task-2

April 3, 2023

[1]:
```
import torch
import numpy as np
import pandas as pd
from tqdm import tqdm
import os
import h5py
import math
import pyarrow.parquet as pq
import torch.nn as nn
import torch.nn.functional as F
from torch.nn import init
from torch.utils.data import Dataset, random_split, DataLoader
from torchvision import transforms
import torch.optim as optim
from torchmetrics.classification import MulticlassAUROC, MulticlassAccuracy
from torch.utils.tensorboard import SummaryWriter
```

[2]:
```
# clearing cuda cache memory
import gc
torch.cuda.empty_cache()
gc.collect()
```

[2]: 22

[3]:
```
# dataset directory
# this directory contains all the datasets related for ML4SCI tests.
os.listdir("../dataset")
```

[3]:
```
['QCDToGGQQ_IMGjet_RH1all_jet0_run0_n36272',
 'QCDToGGQQ_IMGjet_RH1all_jet0_run0_n36272.test.snappy.parquet',
 'QCDToGGQQ_IMGjet_RH1all_jet0_run1_n47540',
 'QCDToGGQQ_IMGjet_RH1all_jet0_run1_n47540.test.snappy.parquet',
 'QCDToGGQQ_IMGjet_RH1all_jet0_run2_n55494',
 'QCDToGGQQ_IMGjet_RH1all_jet0_run2_n55494.test.snappy.parquet',
 'SingleElectronPt50_IMGCROPS_n249k_RHv1.hdf5',
 'SinglePhotonPt50_IMGCROPS_n249k_RHv1.hdf5']
```

[4]:
```
def save_ckpt(imgs,processed_dir,count):
        print("saving...")
        torch.save(imgs,f"{processed_dir}/images-jets{count}-processed.pt")
```

logic to read the image data and corresponding labels

[5]:
```
def read_image_data(dataset_name,count="",start_split=0):
    raw_path = f"../dataset/{dataset_name}/raw/{dataset_name}.test.snappy.
 ↪parquet"
    processed_dir = f"../dataset/{dataset_name}/processed"
    imgs = None
    labels = None
    if f"images-jets{count}-processed.pt" in os.listdir(processed_dir):
        print("loading...")
        imgs = torch.load(f"{processed_dir}/images-jets{count}-processed.pt")
        # load all the label
        # this function returns all the labels
        # hence need truncate if needed seperately.
        labels = torch.load(f"{processed_dir}/labels-jets-processed.pt")
    else:
        dataset = pq.read_table(raw_path,columns=["X_jets","y"]).to_pandas()
        images_raw = dataset["X_jets"].to_numpy()[start_split:]
        labels = dataset["y"][start_split:].to_numpy().astype(np.int64)
        labels = torch.Tensor(labels).to(torch.int32)
        imgs = np.empty([0,125,125,3],dtype=np.float32)
        for inx,img in enumerate(tqdm(images_raw)):
            inx_ = inx+start_split
            img_np = np.stack([np.stack(channel) for channel in img])
            # change the shape to (125,125,3)
            img_np = img_np.transpose()
            imgs = np.vstack((imgs,np.expand_dims(img_np,axis=0)))
            if inx>0 and inx%9068==0:
                imgs = torch.Tensor(imgs)
                save_ckpt(imgs,processed_dir,f"-{str(inx_)}")
        imgs = torch.Tensor(imgs)
        save_ckpt(imgs,labels,processed_dir,"")

    return imgs,labels
```

[6]:
```
# images and labels
img_arrs, labels =␣
 ↪read_image_data("QCDToGGQQ_IMGjet_RH1all_jet0_run0_n36272","")
labels = labels.to(torch.int64)
```

loading...

[7]:
```
# dataset class
# this will ease image/label reading at runtime
```

```
class QuarkGluonDataset(Dataset):
    def __init__(self,split_inx, transform=None,target_transform= None):
        self.img_arrs_split = img_arrs[split_inx]
        self.labels_split = labels[split_inx]
        self.transform = transform
        self.target_transform = target_transform
    def __len__(self):
        return self.labels_split.shape[0]
    def __getitem__(self,idx):
        image=self.img_arrs_split[idx,:,:,:]
        # changing the dim of image to channels, height, width by transposing␣
 ↪the
        # original image tensor.
        image = image.permute(2,1,0)
        label = self.labels_split[idx]
        if self.transform:
            image = self.transform(image)
        if self.target_transform:
            label = self.target_transform(label)
        return image,label
```

[8]:
```
class SeparableConv2d(nn.Module):
    """
    Seperable convolution layer in Xception model, as specified in
    https://arxiv.org/pdf/1610.02357.pdf
    """
    def␣
 ↪__init__(self,in_channels,out_channels,kernel_size=1,stride=1,padding=0,bias=False):
 ↪
        super(SeparableConv2d,self).__init__()

        self.conv1 = nn.
 ↪Conv2d(in_channels,in_channels,kernel_size,stride,padding,groups=in_channels,bias=bias)
        self.pointwise = nn.Conv2d(in_channels,out_channels,1,1,0,1,1,bias=bias)

    def forward(self,x):
        x = self.conv1(x)
        x = self.pointwise(x)
        return x


class Block(nn.Module):
    def␣
 ↪__init__(self,in_channels,out_channels,reps,strides=1,start_with_relu=True,expand_first=Tru
 ↪
        '''
        reps: total number of separable conv layers in the block
```

```
            note that separable conv layers are preceded by relu and followed␣
↪batch normalization.
       start_with_relu: if true start with relu
       expand_first: if True latent embedding dim of the block will be␣
↪expanded to out_channels
                    at the beginning  else latent dim will be expanded at the␣
↪end
       '''
       super(Block, self).__init__()

       if out_channels != in_channels or strides!=1:
           self.skip = nn.Conv2d(in_channels,out_channels,1,stride=strides,␣
↪bias=False)
           self.skipbn = nn.BatchNorm2d(out_channels)
       else:
           self.skip=None

       self.relu = nn.ReLU(inplace=True)
       rep=[]

       filters=in_channels
       if expand_first:
           rep.append(self.relu)
           rep.
↪append(SeparableConv2d(in_channels,out_channels,3,stride=1,padding=1,bias=False))
           rep.append(nn.BatchNorm2d(out_channels))
           filters = out_channels

       for i in range(reps-1):
           rep.append(self.relu)
           rep.
↪append(SeparableConv2d(filters,filters,3,stride=1,padding=1,bias=False))
           rep.append(nn.BatchNorm2d(filters))

       if not expand_first:
           rep.append(self.relu)
           rep.
↪append(SeparableConv2d(in_channels,out_channels,3,stride=1,padding=1,bias=False))
           rep.append(nn.BatchNorm2d(out_channels))

       if not start_with_relu:
           rep = rep[1:]
       else:
           rep[0] = nn.ReLU(inplace=False)

       if strides != 1:
```

```
            rep.append(nn.MaxPool2d(3,strides,1))
        self.rep = nn.Sequential(*rep)

    def forward(self,inp):
        x = self.rep(inp)

        if self.skip is not None:
            skip = self.skip(inp)
            skip = self.skipbn(skip)
        else:
            skip = inp

        x+=skip
        return x



class Xception(nn.Module):
    """
    Xception model, as specified in
    https://arxiv.org/pdf/1610.02357.pdf
    """
    def __init__(self, num_classes=2):
        """ Constructor
        Args:
            num_classes: number of classes
        """
        super(Xception, self).__init__()


        self.num_classes = num_classes

        self.conv1 = nn.Conv2d(3, 32, 3,2, 0, bias=False)
        self.bn1 = nn.BatchNorm2d(32)
        self.relu = nn.ReLU(inplace=True)

        self.conv2 = nn.Conv2d(32,64,3,bias=False)
        self.bn2 = nn.BatchNorm2d(64)

        self.block1=Block(64,128,2,2,start_with_relu=False,expand_first=True)
        self.block2=Block(128,256,2,2,start_with_relu=True,expand_first=True)
        self.block3=Block(256,728,2,2,start_with_relu=True,expand_first=True)

        self.block4=Block(728,728,3,1,start_with_relu=True,expand_first=True)
        self.block5=Block(728,728,3,1,start_with_relu=True,expand_first=True)
        self.block6=Block(728,728,3,1,start_with_relu=True,expand_first=True)
        self.block7=Block(728,728,3,1,start_with_relu=True,expand_first=True)
```

```
        self.block8=Block(728,728,3,1,start_with_relu=True,expand_first=True)
        self.block9=Block(728,728,3,1,start_with_relu=True,expand_first=True)
        self.block10=Block(728,728,3,1,start_with_relu=True,expand_first=True)
        self.block11=Block(728,728,3,1,start_with_relu=True,expand_first=True)

        self.block12=Block(728,1024,2,2,start_with_relu=True,expand_first=False)

        self.conv3 = SeparableConv2d(1024,1536,3,1,1)
        self.bn3 = nn.BatchNorm2d(1536)

        self.conv4 = SeparableConv2d(1536,2048,3,1,1)
        self.bn4 = nn.BatchNorm2d(2048)

        self.fc = nn.Linear(2048, num_classes)

    def forward(self, x):
        x = self.conv1(x)
        x = self.bn1(x)
        x = self.relu(x)

        x = self.conv2(x)
        x = self.bn2(x)
        x = self.relu(x)

        x = self.block1(x)
        x = self.block2(x)
        x = self.block3(x)
        x = self.block4(x)
        x = self.block5(x)
        x = self.block6(x)
        x = self.block7(x)
        x = self.block8(x)
        x = self.block9(x)
        x = self.block10(x)
        x = self.block11(x)
        x = self.block12(x)

        x = self.conv3(x)
        x = self.bn3(x)
        x = self.relu(x)


        x = self.conv4(x)
        x = self.bn4(x)
        x = self.relu(x)
```

```
        x = F.adaptive_avg_pool2d(x, (1, 1))
        x = x.view(x.size(0), -1)
        x = self.fc(x)

        return F.softmax(x,dim=1)

    def __str__(self):
        return "Xception-task2"
```

[9]:
```
# declare the device and the loss function
device = torch.device("cuda:0" if torch.cuda.is_available() else torch.
 ↪device("cpu"))
multicls_criterion = torch.nn.CrossEntropyLoss()
```

[10]:
```
# declare the model
model = Xception(num_classes=2).to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-3)

epochs = 20
```

[11]:
```
# preprocess
preprocess = transforms.Compose([
    transforms.Normalize(mean=[0.5, 0.5,0.5], std=[0.5, 0.5,0.5]),
])

train_inx, valid_inx, test_inx = random_split(range(labels.shape[0]),[0.7,0.2,0.
 ↪1],generator=torch.Generator()
                                                .manual_seed(42))
# random split of train, validation, tests set
# seed it set to 42 for reproducability of results
train_data = QuarkGluonDataset(split_inx=train_inx,transform = preprocess)
valid_data = QuarkGluonDataset(split_inx=valid_inx,transform = preprocess)
test_data = QuarkGluonDataset(split_inx=test_inx,transform = preprocess)

# data loaders
train_dataloader = DataLoader(train_data,batch_size = 64, shuffle = True)
valid_dataloader = DataLoader(valid_data,batch_size = 64, shuffle = True)
test_dataloader = DataLoader(test_data,batch_size = 64, shuffle = True)
```

[12]:
```
# training loop

def train(model, device, loader, optimizer):
    model.train()
    loss_accum = 0
    for step, batch in enumerate(tqdm(loader, desc="Iteration")):
        inputs, labels = batch
        inputs = inputs.to(device)
```

```
        labels = labels.to(device)
        output = model(inputs)
        optimizer.zero_grad()
        loss = multicls_criterion(output, labels)
        loss.backward()
        optimizer.step()

        loss_accum += loss.item()

    return loss_accum / (step + 1)
```

[13]:
```
# evaluation loop

def evaluate(model, device, loader,evaluator="roauc",isTqdm=False,returnLoss=False):
    model.eval()

    preds_list = []
    target_list = []
    loss_accum =0
    iterator = enumerate(loader)
    if isTqdm:
        iterator = enumerate(tqdm(loader))
    for step, batch in iterator:
        inputs, labels = batch
        inputs = inputs.to(device)
        labels = labels.to(device)
        with torch.no_grad():
            output = model(inputs)
            preds_list.extend(output.tolist())
            if returnLoss:
                loss = multicls_criterion(output, labels)
                loss_accum += loss.item()
        target_list += batch[1].tolist()
    if evaluator == "roauc":
        metric = MulticlassAUROC(num_classes=2, average="macro",thresholds=None)
    if evaluator == "acc":
        metric = MulticlassAccuracy(num_classes=2, average="macro")
    # print("AUC-ROC metric score : ",metric(torch.Tensor(preds_list),torch.Tensor(target_list)).item())
    return metric(torch.Tensor(preds_list),torch.Tensor(target_list).to(torch.int64)).item(),loss_accum/(step+1)
```

[14]:
```
# setup for checkpoints saving/loading

checkpoints_path = "../models"
```

```
checkpoints = os.listdir(checkpoints_path)
checkpoint_path = list(filter(lambda i : str(model) in i, checkpoints))
```

[15]:
```
# setup for curve plotting using tensorboard
curves_path = "../tensorboard-plots"
writer = SummaryWriter(log_dir = f"{curves_path}/{str(model)}/exp2")
```

Early stopping criteria

maxPatience : denotes the maximum patience for monotonic increase in validation loss while the
maxTolerance : denotes the maximum patience for increase in validation loss after certain epoch

[16]:
```
# setting maximum patience for early stopping

maxPatience = 3 # patience for monotonic increase
maxTolerance =5 # patience for gradual increase
```

Training the Xception model

[17]:
```
# list of values used for plotting
train_losses = [1000]
val_losses = [1000]

# init values for early stopping and plotting
currentPatience = 0
currentTolerance = 0
toleranceValidScore = -1000.0
starting_epoch = 1

# loading previous checkpoints
if len(checkpoint_path)>0:
    checkpoint = torch.load(f"{checkpoints_path}/{checkpoint_path[0]}")
    model.load_state_dict(checkpoint['model_state_dict'])
    optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
    starting_epoch = checkpoint['epoch']+1
    currentPatience = checkpoint['currentPatience']
    maxPatience = max(checkpoint['prevMaxPatience'],maxPatience)
    currentTolerance = checkpoint['currentTolerance']
    maxTolerance = max(checkpoint['prevMaxTolerance'],maxTolerance)
    toleranceValidScore = checkpoint['toleranceValidScore']
    val_losses = [checkpoint['val_loss']]
    train_losses = [checkpoint['train_loss']]

# training
for epoch in range(starting_epoch, epochs + 1):
    print("=====Epoch {}".format(epoch))
    print('Training...')
    train_loss = train(model, device, train_dataloader, optimizer)
```

```
    print("Evaluating...")
    train_perf_auc,_ = evaluate(model,device,train_dataloader,returnLoss=False)
    valid_perf_auc,val_loss =␣
↪evaluate(model,device,valid_dataloader,returnLoss=True)

    if currentTolerance >0:
        if toleranceValidScore <= val_loss:
            currentTolerance+=1
        else:
            tolerancePoint = f"{checkpoints_path}/
↪{str(model)}-{epoch-(currentTolerance+1)}.pt"
            os.remove(tolerancePoint)
            currentTolerance=0

    if train_losses[-1]>train_loss and val_losses[-1]<=val_loss:
        currentPatience +=1
        if currentTolerance == 0:
            toleranceValidScore = val_losses[-1] # set the starting point a.k.a␣
↪tolerance point
            currentTolerance+=1
    else:
        for pre_inx in range(2,currentPatience+2):
            f = f"{checkpoints_path}/{str(model)}-{epoch-pre_inx}.pt"
            if (currentTolerance != pre_inx) and os.path.exists(f) :
                os.remove(f)
        currentPatience = 0

    train_losses.append(train_loss)
    val_losses.append(val_loss)

    writer.add_scalars('Loss', {"train" : train_loss,
                                "validation"  : val_loss}, epoch)
    writer.add_scalars("AUC",{'train':train_perf_auc,
                              'validation':valid_perf_auc}, epoch)

    # print('Losses: ',{'Train': train_loss, 'Validation': val_loss})
    print('ROC-AUC scores: ',{'Train': train_perf_auc, 'Validation':␣
↪valid_perf_auc})

    # stopping if overfitting

    # stop if the the val loss has increased monotonically
    if currentPatience == maxPatience:
        print("Early stopping training due to overfitting...")
        print(f"obtain results of epoch {epoch-maxPatience}")
        break
```

```
    # stop if the val loss has increased surpassing the tolerance patience
    if currentTolerance == maxTolerance:
        print("Early stopping training due to overfitting...")
        print(f"obtain results of epoch {epoch-(currentTolerance)}")
        break

    # save checkpoint of current epoch
    torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'train_loss':train_loss,
            'val_loss':val_loss,
            'optimizer_state_dict': optimizer.state_dict(),
            'currentPatience':currentPatience,
            'prevMaxPatience':maxPatience,
            'currentTolerance':currentTolerance,
            'prevMaxTolerance':maxTolerance,
            'toleranceValidScore':toleranceValidScore
            }, f"{checkpoints_path}/{str(model)}-{epoch}.pt")

    # delete checkpoint of previous epoch
    if currentPatience == 0:
        if epoch>1:
            prev_ep = f"{checkpoints_path}/{str(model)}-{epoch-1}.pt"
            if os.path.exists(prev_ep):
                os.remove(prev_ep)

print('\nFinished training!')
print('\nROC-AUC Test score: {}'.
 ↪format(evaluate(model,device,test_dataloader)[0]))

# logging and plotting

if currentPatience == maxPatience:
    model_file = f"{checkpoints_path}/{str(model)}-{epoch-maxPatience}.pt"
    if os.path.exists(model_file):
        pre_model = torch.load(model_file)['model_state_dict']
        model.load_state_dict(pre_model)
        test_roc,test_loss =␣
 ↪evaluate(model,device,test_dataloader,returnLoss=True)
        print('\nROC-AUC Test score in {} prior to overfitting: {}'.
 ↪format(epoch-maxPatience,
                                                                    ␣
 ↪test_roc))
        print('\nTest loss in {} prior to overfitting: {}'.
 ↪format(epoch-maxPatience,
```

```
                                                                                ␣
 ↪test_loss))

elif currentTolerance == maxTolerance:
    model_file = f"{checkpoints_path}/{str(model)}-{epoch-maxTolerance}.pt"
    if os.path.exists(model_file):
        pre_model = torch.load(model_file)['model_state_dict']
        model.load_state_dict(pre_model)
        test_roc,test_loss =␣
 ↪evaluate(model,device,test_dataloader,returnLoss=True)
        print('\nROC-AUC Test score in {} prior to overfitting: {}'.
 ↪format(epoch-maxTolerance,
                                                                                ␣
 ↪test_roc))
        print('\nTest loss in {} prior to overfitting: {}'.
 ↪format(epoch-maxTolerance,
                                                                                ␣
 ↪test_loss))

writer.flush()
writer.close()
```

```
=====Epoch 1
Training…

Iteration: 100%|      | 397/397 [02:50<00:00,  2.33it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.7726461887359619, 'Validation': 0.7712996006011963}
=====Epoch 2
Training…

Iteration: 100%|      | 397/397 [03:46<00:00,  1.75it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.7875844240188599, 'Validation': 0.7846232056617737}
=====Epoch 3
Training…

Iteration: 100%|      | 397/397 [04:01<00:00,  1.64it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.7931070327758789, 'Validation': 0.7890264987945557}
=====Epoch 4
Training…

Iteration: 100%|      | 397/397 [04:11<00:00,  1.58it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.7979931831359863, 'Validation': 0.7917254567146301}
=====Epoch 5
```

```
Training…

Iteration: 100%|      | 397/397 [04:04<00:00,  1.62it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.800248384475708, 'Validation': 0.7935057282447815}
=====Epoch 6
Training…

Iteration: 100%|      | 397/397 [04:05<00:00,  1.62it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.8062689900398254, 'Validation': 0.7896032929420471}
=====Epoch 7
Training…

Iteration: 100%|      | 397/397 [04:04<00:00,  1.63it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.8033313155174255, 'Validation': 0.7900791168212891}
=====Epoch 8
Training…

Iteration: 100%|      | 397/397 [04:06<00:00,  1.61it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.8050901889801025, 'Validation': 0.7880290746688843}
=====Epoch 9
Training…

Iteration: 100%|      | 397/397 [04:03<00:00,  1.63it/s]

Evaluating…
ROC-AUC scores:  {'Train': 0.8025626540184021, 'Validation': 0.7715320587158203}
Early stopping training due to overfitting…
obtain results of epoch 6

Finished training!

ROC-AUC Test score: 0.768846333026886

ROC-AUC Test score in 6 prior to overfitting: 0.7768008708953857
```

Evaluating the model on entire dataset

[18]:
```
tot_dataloader = DataLoader(QuarkGluonDataset(split_inx=list(range(labels.
  ↪shape[0])),
                                                              transform =␣
  ↪preprocess))
print('\nROAUC Total score: {}'.
  ↪format(evaluate(model,device,tot_dataloader,isTqdm=True)))
```

```
100%|      | 36272/36272 [10:39<00:00, 56.72it/s]
```

```
ROAUC Total score: (0.7999697923660278, 0.0)
```