LSTM_weighted_sampler

April 2, 2022

0.1 Create Labels for training

[1]:
```
import torch
import torchvision
import torchvision.transforms as transforms
import torch.optim as optim
import time
from itertools import count
import natsort
```

[2]:
```
from torch.utils.data import Dataset, DataLoader, WeightedRandomSampler
import albumentations as A
from albumentations.pytorch import ToTensorV2
import cv2
import glob
import numpy
import random
import pandas as pd
import tqdm
from sklearn import metrics
import matplotlib.pyplot as plt
```

[3]:
```
train_transforms = A.Compose(
    [
        A.Resize(224,224),
        A.Normalize((0.5,0.5,0.5),(0.5,0.5,0.5)),
        ToTensorV2(),
    ]
)
```

[4]:
```
train_image_paths = []
for i in range(1,71):
    filename = '/home/zo2151/assignments/Data/Video%i'%(i,)
    train_image_paths.append(glob.glob(filename + '/*'))
train_image_paths1 = [item for sublist in train_image_paths for item in sublist]
train_image_paths1 = natsort.natsorted(train_image_paths1)
```

[5]:
```
df = pd.read_csv("/home/zo2151/Processed_data.csv")
df1 = df.loc[:,"Phases"].to_numpy()
df2 = df1.tolist()
percentile_list = pd.DataFrame(
    {'Link': train_image_paths1,
     'Label': df2,
    })
```

[6]:
```
percentile_list1 = percentile_list.sample(frac=1, random_state=1)
train_image_paths = percentile_list1.loc[:,"Link"].to_numpy().tolist()
labels = percentile_list1.loc[:,"Label"].to_numpy().tolist()
train_image_paths, valid_image_paths = train_image_paths[:int(0.
↪8*len(train_image_paths))], train_image_paths[int(0.
↪8*len(train_image_paths)):]
train_labels, valid_labels = labels[:int(0.8*len(labels))], labels[int(0.
↪8*len(labels)):]
```

[7]:
```
summary = {i:0 for i in range(14)}
num_classes = 14
total_samples = 0
for i in train_labels:
    total_samples += 1
    summary[i] += 1

class_weights = [total_samples/summary[i] for i in range(num_classes)]
weights = [class_weights[train_labels[i]] for i in range(total_samples)]
sampler = WeightedRandomSampler(torch.DoubleTensor(weights), len(weights))
```

[8]:
```
class SurgicalDataset(Dataset):
    def __init__(self, image_paths, labels, transform=False):
        super(SurgicalDataset, self).__init__()
        self.image_paths = image_paths
        self.transform = transform
        self.labels = labels

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

    def __getitem__(self, idx):
        image_filepath = self.image_paths[idx]
        image = cv2.imread(image_filepath)

        label = self.labels[idx]
        if self.transform is not None:
            image = self.transform(image=image)["image"]

        return image, label
```

[9]:
```
train_dataset = SurgicalDataset(train_image_paths,train_labels,␣
↪train_transforms)
val_dataset = SurgicalDataset(valid_image_paths,valid_labels, train_transforms)
train_loader = DataLoader(
    train_dataset, batch_size=1024, sampler= sampler
)

valid_loader = DataLoader(
    val_dataset, batch_size=1024, shuffle=True
)
```

[10]:
```
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cuda:0
```

0.2 Model with one FC layer substituted as LSTM

[11]:
```
import torch.nn as nn
import torch.nn.functional as F
class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 6, 5)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 5)
        self.lstm1 = nn.LSTM(16 * 53 * 53, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 14)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = torch.flatten(x, 1) # flatten all dimensions except batch
        x, _ = self.lstm1(x)
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x


net = Net()
net.to(device)
```

[11]:
```
Net(
  (conv1): Conv2d(3, 6, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
```

```
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (lstm1): LSTM(44944, 120)
  (fc2): Linear(in_features=120, out_features=84, bias=True)
  (fc3): Linear(in_features=84, out_features=14, bias=True)
)
```

[12]:
```
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(net.parameters(), lr=0.001)
```

[13]:
```
checkpoint = torch.load("/home/zo2151/model3.pt")
net.load_state_dict(checkpoint['model_state_dict'])
optimizer.load_state_dict(checkpoint['optimizer_state_dict'])
```

[14]:
```
for epoch in range(2):  # trained 5 epochs, after 3 epochs, connection lost
    t = time.time()
    running_loss = 0.0
    loop = tqdm.tqdm(train_loader, total = len(train_loader), leave = True)
    for img, label in loop:
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = img.to(device), label.to(device)

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print(f'[{epoch + 1}, {i + 1:5d}] loss: {running_loss / 2000:.3f}')
            running_loss = 0.0
    d = time.time()-t
    print(d)
    torch.save({
            'epoch': 2,
            'model_state_dict': net.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            }, "/home/zo2151/model3.pt")
print('Finished Training')
```

```
100%|                                          |
169/169 [18:53<00:00,  6.71s/it]

1133.9682312011719
```

```
100%|                                                  |
169/169 [18:52<00:00,  6.70s/it]

1132.5939083099365
Finished Training
```

[15]:
```python
classes = [i for i in range(14)]
correct_pred = {classname: 0 for classname in classes}
total_pred = {classname: 0 for classname in classes}
pr = []
pred = []
l = []
# again no gradients needed
t = time.time()
with torch.no_grad():
    for data in valid_loader:
        images, labels = data[0].to(device), data[1].to(device)
        l.append(labels)
        outputs = net(images)
        _, predictions = torch.max(outputs, 1)
        m = F.softmax(outputs, dim=1)
        # collect the correct predictions for each class
        for label, prediction in zip(labels, predictions):
            pred.append(prediction)
            if label == prediction:
                correct_pred[classes[label]] += 1
            total_pred[classes[label]] += 1
        for p in m:
            pr.append(p)
print(time.time()-t)
print(correct_pred)
print(total_pred)
# print accuracy for each class
#for classname, correct_count in correct_pred.items():
    #accuracy = 100 * float(correct_count) / total_pred[classname]
    #print(f'Accuracy for class: {classname:5s} is {accuracy:.1f} %')
```

```
282.01640033721924
{0: 45, 1: 15, 2: 475, 3: 4919, 4: 151, 5: 279, 6: 10749, 7: 2079, 8: 1320, 9:
4583, 10: 229, 11: 8200, 12: 394, 13: 2368}
{0: 52, 1: 18, 2: 587, 3: 5470, 4: 213, 5: 288, 6: 11135, 7: 2177, 8: 3001, 9:
5799, 10: 247, 11: 10405, 12: 404, 13: 3216}
```

[16]:
```python
for i in range(len(l)):
    l[i] = l[i].cpu()
for i in range(len(l)):
    l[i] = l[i].data.numpy()
l = [item for sublist in l for item in sublist]
```

```
for i in range(len(l)):
    pred[i] = pred[i].cpu().data.numpy()
for i in range(len(l)):
    pr[i] = pr[i].cpu().data.numpy()
```

0.3 Some metrics

[17]:
```
metrics.accuracy_score(l, pred)
```

[17]:
```
0.8324653585046033
```

[18]:
```
metrics.f1_score(l, pred, average="macro")
```

[18]:
```
0.8173330759681472
```

[19]:
```
metrics.precision_score(l, pred, average=None)
```

[19]:
```
array([0.83333333, 0.88235294, 0.9082218 , 0.93162879, 0.46461538,
       0.8913738 , 0.83832475, 0.722879  , 0.87301587, 0.85775781,
       0.958159  , 0.95050423, 0.90993072, 0.50946644])
```

[20]:
```
metrics.recall_score(l, pred, average=None)
```

[20]:
```
array([0.86538462, 0.83333333, 0.80919932, 0.89926874, 0.70892019,
       0.96875   , 0.96533453, 0.95498392, 0.43985338, 0.79030867,
       0.92712551, 0.78808265, 0.97524752, 0.73631841])
```

[21]:
```
auc = metrics.roc_auc_score(l, pr, multi_class = "ovr")
```

[22]:
```
auc
```

[22]:
```
0.9855453604105724
```

[23]:
```
auc = metrics.roc_auc_score(l, pr, multi_class = "ovo")
```

[24]:
```
auc
```

[24]:
```
0.9878549535360933
```

[25]:
```
mem_params = sum([param.nelement()*param.element_size() for param in net.
 ↪parameters()])
mem_bufs = sum([buf.nelement()*buf.element_size() for buf in net.buffers()])
mem = mem_params + mem_bufs
```

[26]:
```
mem
```

[26]:
```
86583624
```

[ ]: