

Introduction to Machine Learning, Homework 5

109550155 端木竣偉

• Environment details

我採用的python version為Python 3.9.7

```
(env_test) (base) raytm9999@gpuser3:~/HW5$ python --version  
Python 3.9.7
```

由於原本是在server上train跟inference，requirement.txt高達三百多行，因此手動選出必要的套件版本

```
numpy==1.20.3  
pandas==1.3.4  
tqdm==4.62.3  
torch==1.12.1  
timm==0.6.12  
opencv-python==4.6.0.66
```

開啟新的虛擬環境，並使用pip install -r requirements.txt後，列出以下所安裝的所有套件

```
(env_test) (base) raytm9999@gpuser3:~/HW5$ pip list  
Package            Version  
-----  
certifi             2022.12.7  
charset-normalizer  2.1.1  
filelock            3.8.2  
huggingface-hub     0.11.1  
idna                 3.4  
numpy               1.20.3  
opencv-python       4.6.0.66  
packaging            22.0  
pandas              1.3.4  
Pillow              9.3.0  
pip                 22.3.1  
python-dateutil     2.8.2  
pytz                2022.7  
PyYAML              6.0  
requests            2.28.1  
setuptools          65.6.3  
six                 1.16.0  
timm                0.6.12  
torch               1.12.1  
torchvision         0.13.1  
tqdm                4.62.3  
typing_extensions   4.4.0  
urllib3             1.26.13  
wheel               0.38.4
```

我將model weight放在以下連結：

<https://drive.google.com/drive/folders/1Mt0aC3zHneFT9Xa6Zr-CkFEnwaf7ZvgL?usp=sharing>

save_swin1.pth	我	上午10:25 我	106.1 MB
save_swin2.pth	我	上午10:25 我	106.3 MB
save_swin3.pth	我	上午10:25 我	106.5 MB

打開來應該會看到這三個有點大的權重

• Implementation details

MODEL ARCHITECTURE：

我訓練時是使用三個模型，有三個.py檔，不過都是以swin transformer為框架，只是最後的分類層根據任務不同改為：分為10類、分為兩個36類、分為四個36類。

下圖為task2的模型：

```
#print(timm.list_models('swin*',pretrained=True))
class Model(nn.Module):
    def __init__(self,model_name='swin_tiny_patch4_window7_224',pretrained=True,num_classes=0):
        super().__init__()
        self.layers = timm.create_model(model_name=model_name,pretrained=pretrained,num_classes=num_classes,drop_path_rate = 0.2)
        self.Digit1 = nn.Linear(768, 36)
        self.Digit2 = nn.Linear(768, 36)
    def forward(self, x):
        out=self.layers.forward_features(x)
        #print(out.shape)
        out=out.mean(dim=1)
        #out=out[:,0,:]
        out1 = self.Digit1(out)
        out2 = self.Digit2(out)
        return out1, out2
```

經同儕推薦，使用相當好用的Timm來引入pretrain在imagenet的model，選擇twi transformer的原因是原本使用Resnet、DeiT作為框架，但不管怎麼調學習率跟scheduler，task3都停留在95%，因此改用之前手刃各大榜單的主流模型swin transformer，因為任務不算難且避免overfitting，選擇最小的swin_tiny。

同樣為了避免overfitting，dropout rate設0.2

classifier參考於Timm官方於github的swin transformer.py，使用linear classifier作為最後的分類層，而其input則是來自最後patch embedding feature的mean，想法來源同樣參考於他們原本設計classifier_head的想法。

下圖是task1跟task3的不同之處，如前所述，其餘設定皆和task2相同。

```

        self.digit1 = nn.Linear(768, 10)
    def forward(self, x):
        out=self.layers.forward_features(x)
        #print(out.shape)
        out=out.mean(dim=1)
        #out=out[:,0,:]
        out1 = self.digit1(out)
        return out1

```

```

        self.Digit1 = nn.Linear(768, 36)
        self.Digit2 = nn.Linear(768, 36)
        self.Digit3 = nn.Linear(768, 36)
        self.Digit4 = nn.Linear(768, 36)
    def forward(self, x):
        out=self.layers.forward_features(x)
        #print(out.shape)
        out=out.mean(dim=1)
        #out=out[:,0,:]
        out1 = self.Digit1(out)
        out2 = self.Digit2(out)
        out3 = self.Digit3(out)
        out4 = self.Digit4(out)
        return out1, out2, out3, out4

```

TRANSFER LEARNING :

對於task2跟task3來說，除了使用pretraine過的model來transfer learning以外，我還使用前一個任務的weight來提高正確率。

以下程式碼能去除掉現有Model沒有的架構權重，只更新共有的架構權重。

```

pre_task_weight="save_swin2.pth"
model = Model().to(device)
model_dict=model.state_dict()
pretrained_dict=torch.load(f"{WEIGHT_PATH}/{pre_task_weight}",map_location='cpu')
pretrained_dict = {k: v for k, v in pretrained_dict.items() if k in model_dict}
model_dict.update(pretrained_dict)
model.load_state_dict(model_dict)

```

HYPERARAMETERS&DEEP LEARNING FRAMEWORK :

這三份模型的訓練都使用相同的Framework：

在optimizer選用了Adam：原本在train DeiT時參考作者的作法用SGD，但後來改用Adam覺得效果比較好。

Loss function使用預設的cross entropy。

scheduler則用了CosineLRScheduler，或者叫做Stochastic Gradient Descent with Warm Restarts。

下圖則是task1包含超參數的程式碼：

```

optimizer = torch.optim.Adam(model.parameters(), lr=5e-5,weight_decay = 0.001)
loss_fn = nn.CrossEntropyLoss()
Epoch=15
scheduler=CosineLRScheduler(optimizer,t_initial = Epoch,k_decay=2, lr_min = 5e-7, warmup_t= 5, warmup_lr_init = 1e-6)

```

Learning rate從 10^{-4} ~ 10^{-6} 開始調，選擇中間的 $5e-5$ 。

預防overfitting所以weight_decay=0.001。

Epoch設15，雖然在8~9左右正確率通常都到100%，但再多學能降低validation_loss。

scheduler：

整個週期設成相同Epoch

k_decay設為2調整學習曲線

lr_min= $5e-7$ 怕overfitting設的比較低

一開始從 $1e-6$ 跑五個epoch到達頂峰($5e-5$)，再慢慢往下降

下圖為task2包含超參數的程式碼：

```
pre_task_weight="save_swin1.pth"
model_dict=model.state_dict()
pretrained_dict=torch.load(f"{WEIGHT_PATH}/{pre_task_weight}",map_location='cpu')
pretrained_dict = {k: v for k, v in pretrained_dict.items() if k in model_dict}
model_dict.update(pretrained_dict)
model.load_state_dict(model_dict)
optimizer = torch.optim.Adam(model.parameters(), lr=5e-5,weight_decay = 0.001)
loss_fn = nn.CrossEntropyLoss()
Epoch=35
scheduler=CosineLRScheduler(optimizer,t_initial = Epoch,k_decay=2, lr_min = 5e-7, warmup_t= 5, warmup_lr_init = 1e-6)
```

我先將task1的weight載入作為initial weight

Epoch設35

其他都和task1相同，訓練出來也有99.7%或是100%。

下圖為task3包含超參數的程式碼：

```
pre_task_weight="save_swin2.pth"
model = Model().to(device)
model_dict=model.state_dict()
pretrained_dict=torch.load(f"{WEIGHT_PATH}/{pre_task_weight}",map_location='cpu')
pretrained_dict = {k: v for k, v in pretrained_dict.items() if k in model_dict}
model_dict.update(pretrained_dict)
model.load_state_dict(model_dict)
optimizer = torch.optim.Adam(model.parameters(), lr=2e-5,weight_decay = 0.001)
loss_fn = nn.CrossEntropyLoss()
Epoch=100
scheduler=CosineLRScheduler(optimizer,t_initial = Epoch,k_decay=2, lr_min = 1e-7, warmup_t= 5, warmup_lr_init = 1e-6)
```

我先將task2的weight載入作為initial weight

Epoch設100

和之前不同的是我將learning rate調到 $2e-5$ ，因為這個任務在training set很快就能達到100%，但是val_set卻不夠高，因此我將lr調到更低，並且在scheduler的lr_min也調到 $1e-7$ ，讓訓練比較好逼近local minimum，提高正確率。

接下來我就按照sample code做gradient_descent，下圖是task3的訓練過程：

```

image = image.to(device)
label = label.to(device)
pred1, pred2, pred3, pred4 = model(image)
#print(pred1.shape)
#print(type(pred1))
#print(label[:,0].shape)
#print(type(label[:,0]))
loss = loss_fn(pred1, label[:, 0])
loss += loss_fn(pred2, label[:, 1])
loss += loss_fn(pred3, label[:, 2])
loss += loss_fn(pred4, label[:, 3])
Train_loss += loss
optimizer.zero_grad()
loss.backward()
optimizer.step()
scheduler.step(epoch)

```

唯一有變動的地方是scheduler根據epoch的進展更新
對於其他task，只是要算loss的字數不同而已。

DIFFERENT FROM SAMPLE CODE :

以task3的程式碼為例，我將36種可能（0~9+a~z）放到word這個list，將label轉為tensor，而他們所代表的分類號碼就是在word的index。

另外由於model的要求，我將圖片拉到224x224大小，並且根據將維度轉換成符合input的樣子。

```

word = ['0','1','2','3','4','5','6','7','8','9','a','b','c','d','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s',
        't','u','v','w','x','y','z']
class Task3Dataset(Dataset):
    def __init__(self, data, root, return_filename=False):
        self.data = [sample for sample in data if sample[0].startswith("task3")]
        self.return_filename = return_filename
        self.root = root

    def __getitem__(self, index):
        filename, label = self.data[index]
        img = cv2.imread(f"{self.root}/{filename}")
        img = cv2.resize(img, (224, 224))
        label=torch.tensor([word.index(label[0]),word.index(label[1]),word.index(label[2]),word.index(label[3])])
        img=torch.FloatTensor(img).permute(2, 0, 1)
        if self.return_filename:
            return torch.FloatTensor((img - 128) / 128), filename
        else:
            return torch.FloatTensor((img - 128) / 128), label

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

```

另外為了要追求正確率，把所有數據都放進訓練集，還有根據cuda大小調整batch_size。

```
with open(f'{TRAIN_PATH}/annotations.csv', newline='') as csvfile:
    for row in csv.reader(csvfile, delimiter=','):
        train_data.append(row)

train_ds = Task3Dataset(train_data, root=TRAIN_PATH)
train_dl = DataLoader(train_ds, batch_size=15, num_workers=4, drop_last=True, shuffle=True)
```

INFERENCE PART:

我將weight的資料夾路徑叫做WEIGHT_PATH，TEST_PATH才是測試時會用到的路徑。

```
13  WEIGHT_PATH = "swin_weight"
14  TRAIN_PATH = "captcha-hacker/train"
15  TEST_PATH = "captcha-hacker/test"
16  device = "cuda"
```

接下來TaskDataset的部分和訓練部分相同。

還有將三個模型放在class Model1、class Model2、class Model3。

接著會打開一個新的submission.csv

```
if os.path.exists('submission.csv'):
    csv_writer = csv.writer(open('submission.csv', 'w', newline=''))
    csv_writer.writerow(["filename", "label"])
```

接下來就是一個跑三次的for迴圈

Task123是放任務名稱

Weight123則是放對應模型的權重名稱。

```
model=None
task123=["task1","task2","task3"]
Weight123=["save_swin1.pth","save_swin2.pth","save_swin3.pth"]
for i in range(3):
    print("NOW is runing task:",(i+1))
    test_data = []
    with open(f'{TEST_PATH}/../sample_submission.csv', newline='') as csvfile:
        for row in csv.reader(csvfile, delimiter=','):
            test_data.append(row)

    test_ds = TaskDataset(test_data, root=TEST_PATH, return_filename=True, task_name=task123[i])
    test_dl = DataLoader(test_ds, batch_size=50, num_workers=4, drop_last=False, shuffle=False)
```

根據載入不同的任務、模型、權重。


可以看到model是在此使用WEIGHT_PATH作為路徑，因此要修改對應路徑。


```
if i==0:
    model = Model1().to(device)
elif i==1:
    model = Model2().to(device)
else:
    model = Model3().to(device)
model.load_state_dict(torch.load(f"{WEIGHT_PATH}/{Weight123[i]}",map_location='cpu'))
```

將模型轉成evaluate後根據l開始輸出不同長度的結果，將他們concatenate起來後轉成str寫到submission.csv。

```
model.eval()
for image, filenames in test_dl:
    image = image.to(device)
    if i==0:
        pred = model(image)
        pred = torch.argmax(pred, dim=1)
        for j in range(len(filenames)):
            csv_writer.writerow([filenames[j], str(pred[j].item())])
    elif i==1:
        pred1, pred2 = model(image)
        pred1 = torch.argmax(pred1, dim=1)
        pred2 = torch.argmax(pred2, dim=1)
        for j in range(len(filenames)):
            csv_writer.writerow([filenames[j], str(word[pred1[j].item()+word[pred2[j].item()])])
    else:
        pred1, pred2, pred3, pred4 = model(image)
        pred1 = torch.argmax(pred1, dim=1)
        pred2 = torch.argmax(pred2, dim=1)
        pred3 = torch.argmax(pred3, dim=1)
        pred4 = torch.argmax(pred4, dim=1)
        for j in range(len(filenames)):
            csv_writer.writerow([filenames[j], str(word[pred1[j].item()+word[pred2[j].item()+word[pred3[j].item()+word[pred4[j].item()])])])
```

至此，就能夠從訓練並inference，獲得一樣的結果。

13	109550155		0.99720	4	7h
----	-----------	---	---------	---	----



Your Best Entry!
Your most recent submission scored 0.99720, which is the same as your previous score. Keep trying!

在sever跑的結果跟在虛擬環境的結果都是0.99720，可見前述的環境架設是能正確產出相同結果的。