به نام خدا

تمرین دهم یادگیری عمیق

غزل زمانی‌نژاد

۹۷۵۲۲۱۶۶

برای آموزش تمامی مدل ها از batch size = 24، بهینه ساز Adam با نرخ یادگیری 0.001 و تابع خطا cross entropy استفاده میکنیم. همچنین مقدار نرخ یادگیری را در طی epochها کاهش میدهیم.

همچنین برای بیشتر کردن ظرفیت یادگیری ResnetModel، به آن یک لایه خطی دیگر اضافه میکنیم.

```python
class ResnetModel(nn.Module):
    def __init__(self, original_model, num_classes):
        super(ResnetModel, self).__init__()

        # Everything except the last linear layer
        self.features = nn.Sequential(*list(original_model.children())[:-1])
        self.classifier = nn.Sequential(
            nn.Linear(2048, 512),
            nn.Linear(512, num_classes),
        )

    def forward(self, x):
        f = self.features(x)
        f = f.view(f.size(0), -1)
        y = self.classifier(f)
        return y
```

الف) برای ساختن یک مدل با وزن های تصادفی باید مقدار pretrained در شبکه resnet را false قرار دهیم. تمامی پارامترهای این شبکه trainable هستند به همین دلیل هر دو خط چاپ شده یک مقدار را نمایش میدهند.

```python
# Q1
# resnet model with random initialization
res_mod1 = models.resnet50(pretrained=False)
model1 = ResnetModel(res_mod1, num_classes=len(classes))
pytorch_total_params1 = sum(p.numel() for p in model1.parameters())
pytorch_total_trainable_params1 = sum(p.numel() for p in model1.parameters() if p.requires_grad)
print('first model, pytorch_total_params= ', pytorch_total_params1 )
print('first model, pytorch_total_trainable_params= ', pytorch_total_trainable_params1)
```

```
first model, pytorch_total_params=  24657668
first model, pytorch_total_trainable_params=  24657668
```

سپس مدل را به device موجود انتقال میدهیم. برای سریعتر شدن محاسبات از gpu استفاده میکنیم. مدل را در 25 ایپوک آموزش میدهیم.

```
model1 = model1.to(device)

# Observe that all parameters are being optimized
optimizer_ft1 = optim.Adam(model1.parameters(), lr=0.001)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler1 = lr_scheduler.StepLR(optimizer_ft1, step_size=7, gamma=0.1)
base_model1 = train_model(model1, criterion, optimizer_ft1, exp_lr_scheduler1, num_epochs=25)
```

بدلیل عمیق بودن شبکه و زیاد بودن تعداد پارامترها آموزش مدل آهسته صورت میگیرد.

نتایج ایپوک های پایانی:

```
Epoch 20/24
----------
Iterating through data...
train Loss: 130.6115 Acc: 0.0796
Iterating through data...
val Loss: 150.4317 Acc: 0.0448

Epoch 21/24
----------
Iterating through data...
train Loss: 130.6315 Acc: 0.0783
Iterating through data...
val Loss: 150.7259 Acc: 0.0446

Epoch 22/24
----------
Iterating through data...
train Loss: 130.2091 Acc: 0.0820
Iterating through data...
val Loss: 150.5634 Acc: 0.0449

Epoch 23/24
----------
Iterating through data...
train Loss: 130.0162 Acc: 0.0777
Iterating through data...
```

بدلیل اینکه مدل دارای وزن های تصادفی و دیتاست 196 کلاسه است، مدل نتوانسته عملکرد خوبی داشته باشد. و شدیدا underfit شده است. مدل در صورت پیش بینی تصادفی 0.5٪ خواهد داشت. در اینجا توانسته دقت 7 درصد آموزش و 4 درصد آزمون را بدست بیاورد که از حالت تصادفی بهتر است. با این حال برای یک مسئله یادگیری عمیق دقت نامناسبی است.

ب) در این قسمت وزن های شبکه resnet را فریز میکنیم تا آموزش نبینند. یعنی در یک لوپ مقدار require grad برای تمامی پارامترها را false میکنیم. تعداد پارامترهای قابل آموزش این مدل همان تعداد پارامترهای لایه classifier است.

```python
# Q2
# pretrained resnet model (freezed resnet parameters)
res_mod2 = models.resnet50(pretrained=True)
for param in res_mod2.parameters():
    param.requires_grad = False

model2 = ResnetModel(res_mod2, num_classes=len(classes))
pytorch_total_params2 = sum(p.numel() for p in model2.parameters())
pytorch_total_trainable_params2 = sum(p.numel() for p in model2.parameters() if p.requires_grad)
print('second model, pytorch_total_params= ', pytorch_total_params2)
print('second model, pytorch_total_trainable_params= ', pytorch_total_trainable_params2)
```

```
second model, pytorch_total_params=  24657668
second model, pytorch_total_trainable_params=  1149636
```

مدل را در 15 ایپوک آموزش میدهیم.

```python
model2 = model2.to(device)

# Observe that all parameters are being optimized
optimizer_ft2 = optim.Adam(model2.parameters(), lr=0.001)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler2 = lr_scheduler.StepLR(optimizer_ft2, step_size=7, gamma=0.1)
base_model2 = train_model(model2, criterion, optimizer_ft2, exp_lr_scheduler2, num_epochs=15)
```

نتایج ایپوک های پایانی:

```
Epoch 11/14
----------
Iterating through data...
train Loss: 21.6144 Acc: 0.8341
Iterating through data...
val Loss: 80.7795 Acc: 0.4089

Epoch 12/14
----------
Iterating through data...
train Loss: 20.8447 Acc: 0.8403
Iterating through data...
val Loss: 81.1381 Acc: 0.4100

Epoch 13/14
----------
Iterating through data...
train Loss: 18.4661 Acc: 0.8723
Iterating through data...
val Loss: 79.5789 Acc: 0.4175

Epoch 14/14
----------
Iterating through data...
train Loss: 18.5024 Acc: 0.8694
Iterating through data...
val Loss: 79.4750 Acc: 0.4167

Training complete in 62m 15s
Best val Acc: 0.417485
```

مشاهده میشود این مدل نسبت به مدل قبلی پیشرفت چشمگیری داشته. اما دچار overfit شده و اختلاف میان دقت آموزش و دقت آزمون زیاد است. ممکن است در صورت استفاده از روش های regularization این اختلاف کاهش یابد.

ج) در اینجا باید از شبکه resnet تنها برای استخراج ویژگی استفاده کنیم. پس یک مدل بدین جهت میسازیم.

```
class FeatureExtractor(nn.Module):
    def __init__(self, original_model):
        super(FeatureExtractor, self).__init__()

        # Everything except the last linear layer
        self.features = nn.Sequential(*list(original_model.children())[:-1])

    def forward(self, x):
        f = self.features(x)
        f = f.view(f.size(0), -1)
        return f
```

با استفاده از تابع extract بر روی dataloader حرکت میکنیم و برای تمامی داده ها، ویژگی های resnet را استخراج میکنیم. این ویژگی ها به همراه برچسب اصلی کلاس ها را به دو لیست اضافه میکنیم و لیست ها را برمی گردانیم.

```
def extract(model, phase):
    features = []
    true_labels = []
    for inputs, labels in dataloaders[phase]:
        inputs = inputs.to(device)
        labels = labels.to(device)

        out = model(inputs)

        for i in range(out.cpu().shape[0]):
            features.append(np.array(out[i].cpu()))
            true_labels.append(labels[i].cpu())

    return (features, true_labels)
```

از معماری شبکه و تابع استخراج ویژگی استفاده میکنیم تا ویژگی داده های آموزش و آزمون را بیابیم.

```
# Q3
# pretrained resnet model (freezed resnet parameters)
# with SVM classifier
res_mod3 = models.resnet50(pretrained=True)
for param in res_mod3.parameters():
    param.requires_grad = False

model3 = FeatureExtractor(res_mod3)
model3 = model3.to(device)

features, labels = extract(model3, 'train')
features_val, labels_val = extract(model3, 'val')
```

فرمت آنها را از torch tensor به numpy array تبدیل میکنیم تا توسط svm قابل پذیرش باشد.

```
features = np.array(features)
labels = np.array([x for x in labels])  # convert tensor to number

features_val = np.array(features_val)
labels_val = np.array([x for x in labels_val])

print(features.shape)
print(labels.shape)
print(features_val.shape)
print(labels_val.shape)
```

```
(8144, 2048)
(8144,)
(8041, 2048)
(8041,)
```

از کتابخانه scikit learn یک مدل svc می سازیم و کرنل آن را به rbf ست میکنیم و با استفاده از تابع fit آن را بر روی داده آموزشی train میکنیم.

```
from sklearn import svm

#Create a svm Classifier
svm_classifier = svm.SVC(kernel='rbf')

#Train the model using the training sets
svm_classifier.fit(features, labels)
```

```
SVC()
```

سپس با استفاده از تابع predict پیش بینی مدلی که train شده را بدست می آوریم و دقت ها را چاپ میکنیم.

```
from sklearn import metrics

y_pred = svm_classifier.predict(features)
print("Accuracy on train data:", metrics.accuracy_score(labels, y_pred))
```

```
Accuracy on train data: 0.9680746561886051
```

```
y_pred_val = svm_classifier.predict(features_val)
print("Accuracy:", metrics.accuracy_score(labels_val, y_pred_val))
```

```
Accuracy: 0.3283173734610123
```

نتایج پایانی:

دسته بند svm بر روی داده آموزشی عملکرد بسیار خوبی داشته اما نسبت به دسته بند خطی، بر روی داده آزمون عملکرد ضعیف تری داشته. یعنی یک شبکه عصبی عمیق با دسته بند خطی دقت بیشتری نسبت به روش های دسته بندی کلاسیک کسب کرده است. همچنین میزان overfit شدن شبکه با دسته بند svm از شبکه با دسته بند خطی بیشتر است.

د) برای تنظیم دقیق پیشنهاد میشود:

ابتدا قسمت pretrained مدل فریز شود و تنها لایه های اضافه شده آموزش ببینند. (در قسمت 2 سوال این کار انجام شد)

سپس چند لایه انتهایی مدل pretrained از فریز در بیایند و به همراه لایه های جدید آموزش ببینند.

یعنی در ابتدا مدل 2 (که وزن های آن را ذخیره کرده بودیم) را در مدل load 4 میکنیم. سپس 20 لایه انتهایی را از فریز خارج کرده و همراه با لایه classifier آموزش میدهیم.

```
# Q4
# pretrained resnet model (train last layers of resnet)

# first load model2, then defreeze some layers
res_mod4 = models.resnet50()
model4 = ResnetModel(res_mod4, num_classes=len(classes))
model4.load_state_dict(state_dict2)

layer = 0
for name, module in model4.features.named_modules():
    if layer > 130:
        for p in module.parameters():
            p.requires_grad = False
    layer += 1

```

مدل را در 20 ایپوک آموزش میدهیم.

```
model4 = model4.to(device)

# Observe that all parameters are being optimized
optimizer_ft4 = optim.Adam(model4.parameters(), lr=0.001)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler4 = lr_scheduler.StepLR(optimizer_ft4, step_size=7, gamma=0.1)
base_model4 = train_model(model4, criterion, optimizer_ft4, exp_lr_scheduler4, num_epochs=20)
```

نتایج ایپوک های پایانی:

مشاهده میشود مدل عملکرد بسیار خوبی هم بر روی داده آموزشی و هم داده آزمون دارد. بدلیل بالا بودن ظرفیت شبکه دقت داده آزمون نزدیک به 100 درصد است و شبکه بعد از مدتی به حفظ الگوها پرداخته است. یعنی مقداری overfit شده است.

```
Epoch 16/19
----------
Iterating through data...
train Loss: 0.3161 Acc: 0.9983
Iterating through data...
val Loss: 34.7846 Acc: 0.7350

Epoch 17/19
----------
Iterating through data...
train Loss: 0.3021 Acc: 0.9986
Iterating through data...
val Loss: 34.5990 Acc: 0.7296

Epoch 18/19
----------
Iterating through data...
train Loss: 0.3231 Acc: 0.9980
Iterating through data...
val Loss: 34.5073 Acc: 0.7341

Epoch 19/19
----------
Iterating through data...
train Loss: 0.2915 Acc: 0.9984
Iterating through data...
val Loss: 34.5954 Acc: 0.7350

Training complete in 101m 0s
Best val Acc: 0.734983
```

ه) چاپ کردن درصد 0 های هر لایه مدل 4:

ابتدا وزن های مدل 4 را load میکنیم.

```
1 state_dict4 = torch.load(path4)
```

```
1 res_mod4 = models.resnet50()
2 model4 = ResnetModel(res_mod4, num_classes=len(classes))
3 model4.load_state_dict(state_dict4)
```

```
<All keys matched successfully>
```

سپس از مفهوم hook استفاده میکنیم. یک کلاس برای ذخیره کردن خروجی هر لایه میسازیم. (کلاس SaveOutput) بر روی ماژول های مدل iterate میکنیم و برای ذخیره کردن خروجی ها از متد register_forward_hook استفاده میکنیم.

```
model4 = model4.to(device)

class SaveOutput:
    def __init__(self):
        self.outputs = []

    def __call__(self, module, module_in, module_out):
        self.outputs.append(module_out)

    def clear(self):
        self.outputs = []


save_output = SaveOutput()
hook_handles = []

for layer in model4.modules():
    handle = layer.register_forward_hook(save_output)
    hook_handles.append(handle)
```

یک تصویر را به صورت تصادفی از دیتاست انتخاب میکنیم و آن را به مدل 4 میدهیم.

```
it = iter(dataloaders['train'])
first = next(it)
img = first[0][12]
print(img.shape)

X = img.unsqueeze(dim=0).to(device)
print(X.shape)
```

```
torch.Size([3, 224, 224])
torch.Size([1, 3, 224, 224])
```

```
out = model4(X)
```

```
len(save_output.outputs)
```

```
186
```

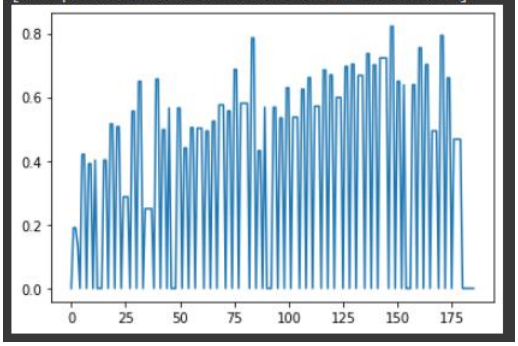
سپس در لیست خروجی هر لایه میچرخیم. تعداد مقادیر صفر و همچنین درصد آن را محاسبه میکنیم و به یک لیست اضافه میکنیم.

```
model4_zeros = []
for layer_output in save_output.outputs:
    x = torch.flatten(layer_output)
    nonzero = torch.count_nonzero(x).cpu()
    zeros = x.shape[0] - nonzero
    model4_zeros.append(zeros / x.shape[0])
```

با استفاده از matplotlib **نمودار را رسم میکنیم.**

```
1 import matplotlib.pyplot as plt
2 plt.plot(model4_zeros)
```

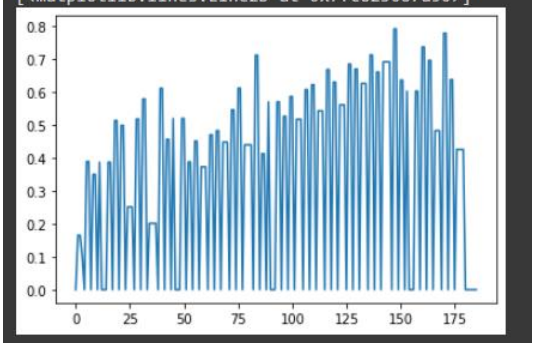
```
[<matplotlib.lines.Line2D at 0x7fc82a2f2e90>]
```



برای مدل 2 نیز همین مراحل را طی میکنیم. نمودار قبل از انجام fine tune**:**

```
1 plt.plot(model2_zeros)
```

```
[<matplotlib.lines.Line2D at 0x7fc823687a50>]
```



قبل از انجام fine tuning **درصد مقادیر غیرصفر اندکی بیشتر بوده اند.**

منابع

https://towardsdatascience.com/the-one-pytorch-trick-which-you-should-know-2d5e9c1da2ca