学习率衰减

在 pytorch 中学习率衰减非常方便，使用 `torch.optim.lr_scheduler`，更多的信息可以直接查看文档

但是我推荐大家使用下面这种方式来做学习率衰减，更加直观，下面我们直接举例子来说明

```
import numpy as np
import torch
from torch import nn
import torch.nn.functional as F
from torch.autograd import Variable
from torchvision.datasets import CIFAR10
from utils import resnet
from torchvision import transforms as tfs
from datetime import datetime
```

```
net = resnet(3, 10)
optimizer = torch.optim.SGD(net.parameters(), lr=0.01, weight_decay=1e-4)
```

这里我们定义好了模型和优化器，可以通过 `optimizer.param_groups` 来得到所有的参数组和其对应的属性，参数组是什么意思呢？就是我们可以将模型的参数分成几个组，每个组定义一个学习率，这里比较复杂，一般来讲如果不做特别修改，就只有一个参数组

这个参数组是一个字典，里面有很多属性，比如学习率，权重衰减等等，我们可以访问以下

```
print('learning rate: {}'.format(optimizer.param_groups[0]['lr']))
print('weight decay: {}'.format(optimizer.param_groups[0]['weight_decay']))
```

```
learning rate: 0.01
weight decay: 0.0001
```

所以我们可以通过修改这个属性来改变我们训练过程中的学习率，非常简单

```
optimizer.param_groups[0]['lr'] = 1e-5
```

为了防止有多个参数组，我们可以使用一个循环

```
for param_group in optimizer.param_groups:
    param_group['lr'] = 1e-1
```

方法就是这样，非常简单，我们可以在任意的位置改变我们的学习率

下面我们具体来看看学习率衰减的好处

```
def set_learning_rate(optimizer, lr):
    for param_group in optimizer.param_groups:
        param_group['lr'] = lr
```

```
# 使用数据增强
def train_tf(x):
    im_aug = tfs.Compose([
        tfs.Resize(120),
        tfs.RandomHorizontalFlip(),
        tfs.RandomCrop(96),
        tfs.ColorJitter(brightness=0.5, contrast=0.5, hue=0.5),
        tfs.ToTensor(),
        tfs.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
    ])
    x = im_aug(x)
    return x

def test_tf(x):
    im_aug = tfs.Compose([
        tfs.Resize(96),
        tfs.ToTensor(),
        tfs.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])
    ])
    x = im_aug(x)
    return x

train_set = CIFAR10('./data', train=True, transform=train_tf)
train_data = torch.utils.data.DataLoader(train_set, batch_size=256, shuffle=True,
num_workers=4)
valid_set = CIFAR10('./data', train=False, transform=test_tf)
valid_data = torch.utils.data.DataLoader(valid_set, batch_size=256, shuffle=False,
num_workers=4)

net = resnet(3, 10)
optimizer = torch.optim.SGD(net.parameters(), lr=0.1, weight_decay=1e-4)
criterion = nn.CrossEntropyLoss()
```

```
train_losses = []
valid_losses = []

if torch.cuda.is_available():
    net = net.cuda()
prev_time = datetime.now()
for epoch in range(30):
    if epoch == 20:
        set_learning_rate(optimizer, 0.01) # 80 次修改学习率为 0.01
    train_loss = 0
    net = net.train()
    for im, label in train_data:
        if torch.cuda.is_available():
            im = Variable(im.cuda())  # (bs, 3, h, w)
            label = Variable(label.cuda())  # (bs, h, w)
        else:
            im = Variable(im)
            label = Variable(label)
        # forward
        output = net(im)
        loss = criterion(output, label)
        # backward
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        train_loss += loss.data[0]
    cur_time = datetime.now()
    h, remainder = divmod((cur_time - prev_time).seconds, 3600)
    m, s = divmod(remainder, 60)
    time_str = "Time %02d:%02d:%02d" % (h, m, s)
    valid_loss = 0
    valid_acc = 0
    net = net.eval()
    for im, label in valid_data:
        if torch.cuda.is_available():
            im = Variable(im.cuda(), volatile=True)
            label = Variable(label.cuda(), volatile=True)
        else:
            im = Variable(im, volatile=True)
            label = Variable(label, volatile=True)
        output = net(im)
        loss = criterion(output, label)
        valid_loss += loss.data[0]
    epoch_str = (
        "Epoch %d. Train Loss: %f, Valid Loss: %f, "
```

```
        % (epoch, train_loss / len(train_data), valid_loss / len(valid_data)))
    prev_time = cur_time

    train_losses.append(train_loss / len(train_data))
    valid_losses.append(valid_loss / len(valid_data))
    print(epoch_str + time_str)
```

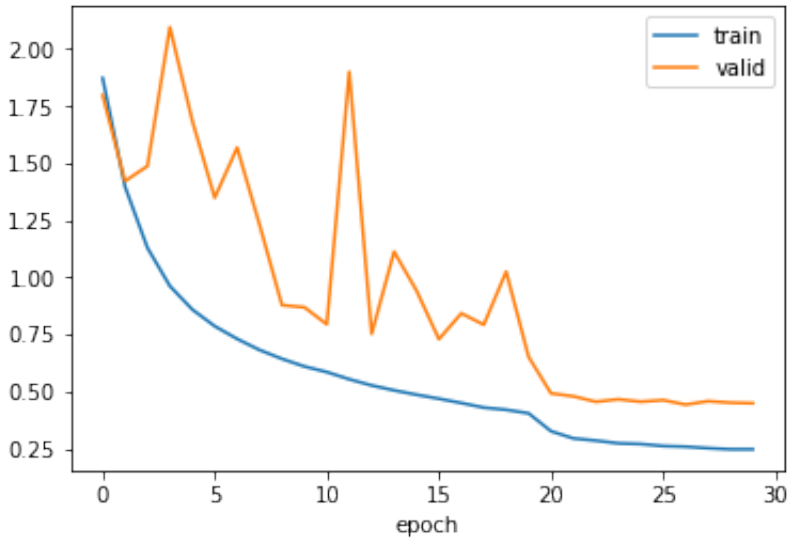
```
Epoch 0. Train Loss: 1.872896, Valid Loss: 1.798441, Time 00:00:26
Epoch 1. Train Loss: 1.397522, Valid Loss: 1.421618, Time 00:00:28
Epoch 2. Train Loss: 1.129362, Valid Loss: 1.487882, Time 00:00:28
Epoch 3. Train Loss: 0.962217, Valid Loss: 2.095880, Time 00:00:28
Epoch 4. Train Loss: 0.859332, Valid Loss: 1.686056, Time 00:00:27
Epoch 5. Train Loss: 0.786428, Valid Loss: 1.348701, Time 00:00:27
Epoch 6. Train Loss: 0.730535, Valid Loss: 1.568454, Time 00:00:27
Epoch 7. Train Loss: 0.682074, Valid Loss: 1.230555, Time 00:00:28
Epoch 8. Train Loss: 0.643144, Valid Loss: 0.878328, Time 00:00:27
Epoch 9. Train Loss: 0.609817, Valid Loss: 0.869068, Time 00:00:27
Epoch 10. Train Loss: 0.585312, Valid Loss: 0.794440, Time 00:00:27
Epoch 11. Train Loss: 0.553877, Valid Loss: 1.900850, Time 00:00:27
Epoch 12. Train Loss: 0.526790, Valid Loss: 0.752651, Time 00:00:27
Epoch 13. Train Loss: 0.505155, Valid Loss: 1.112544, Time 00:00:27
Epoch 14. Train Loss: 0.486104, Valid Loss: 0.942357, Time 00:00:27
Epoch 15. Train Loss: 0.468512, Valid Loss: 0.729420, Time 00:00:27
Epoch 16. Train Loss: 0.449669, Valid Loss: 0.842467, Time 00:00:27
Epoch 17. Train Loss: 0.429664, Valid Loss: 0.792635, Time 00:00:27
Epoch 18. Train Loss: 0.420088, Valid Loss: 1.025345, Time 00:00:27
Epoch 19. Train Loss: 0.404701, Valid Loss: 0.651725, Time 00:00:27
Epoch 20. Train Loss: 0.326198, Valid Loss: 0.491904, Time 00:00:27
Epoch 21. Train Loss: 0.294623, Valid Loss: 0.478969, Time 00:00:27
Epoch 22. Train Loss: 0.284980, Valid Loss: 0.455259, Time 00:00:28
Epoch 23. Train Loss: 0.273168, Valid Loss: 0.465930, Time 00:00:27
Epoch 24. Train Loss: 0.270120, Valid Loss: 0.455458, Time 00:00:27
Epoch 25. Train Loss: 0.261299, Valid Loss: 0.462319, Time 00:00:27
Epoch 26. Train Loss: 0.258373, Valid Loss: 0.442525, Time 00:00:27
Epoch 27. Train Loss: 0.251803, Valid Loss: 0.457620, Time 00:00:27
Epoch 28. Train Loss: 0.247022, Valid Loss: 0.451055, Time 00:00:27
Epoch 29. Train Loss: 0.246816, Valid Loss: 0.448706, Time 00:00:28
```

下面我们画出 loss 曲线

```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
plt.plot(train_losses, label='train')
plt.plot(valid_losses, label='valid')
plt.xlabel('epoch')
plt.legend(loc='best')
```

```
<matplotlib.legend.Legend at 0x7fe8a60d4e48>
```



这里我们只训练了 30 次，在 20 次的时候进行了学习率衰减，可以看 loss 曲线在 20 次的时候不管是 train loss 还是 valid loss，都有了一个陡降。

当然这里我们只是作为举例，在实际应用中，做学习率衰减之前应该经过充分的训练，比如训练 80 次或者 100 次，然后再做学习率衰减得到更好的结果，有的时候甚至需要做多次学习率衰减