链滴

用 fastai 解释什么是 one-cycle-policy

作者：lai-bluejay

原文链接：https://ld246.com/article/1552929502787

来源网站：链滴

fit one cycle

"Manage 1-Cycle style training as outlined in Leslie Smith's paper."

what is one-cycle-policy?

简单来说，one-cycle-policy, 使用的是一种周期性学习率，从较小的学习率开始学习，缓慢提高至高的学习率，然后再慢慢下降，周而复始，每个周期的长度略微缩短，在训练的最后部分，学习率比前的最小值降得更低。这不仅可以加速训练，还有助于防止模型落入损失平面的陡峭区域，使模型更向于寻找更平坦部分的极小值，从而缓解过拟合现象。

one cycle 3步

One-Cycle-Policy大概有三个步骤：

1. 我们逐渐将学习率从lr_max / div_factor提高到lr_max，同时我们逐渐减少从mom_max到mom_min的动量(momentum)。

2. 反向再做一次：我们逐渐将学习率从lr_max降低到lr_max / div_factor，同时我们逐渐增加从mom min到mom_max的动量。

3. 我们进一步将学习率从lr_max / div_factor降低到lr_max /（div_factor x 100），我们保持动力稳在mom_max。

我们来分别简单说明一下：

慢启动的想法并不新鲜：通常使用较低的值来预热训练，这正是one-cycle第一步实现的目标。Leslie建议直接切换到更高的值，而是线性提高的最大值。

他在实验过程中观察到的是，在Cycle中期，高学习率将作为正则化的角色，并使NN不会过拟合。它将阻止模型落在损失函数的陡峭区域，而是找到更平坦的最小值。他在另一篇论文中解释了通过使用一政策，Hessian的近似值较低，表明SGD正在寻找更宽的平坦区域。

然后训练的最后一部分，学习率下降直到消失，将允许我们得到更平滑的部分内的局部最小值。在高习率的同时，我们没有看到loss或acc的显着改善，并且验证集的loss有时非常高。但是当我们最终在后降低学习率时，我们会发现这是很有好处的。

reference

具体的技术细节，可以参考Leslie 的论文。本文接下来的实验脚本都在Cyclical LR and momentums.pynb

建议大家直接在colab运行这个脚本，在chrome中安装open-in-colab extension，一键打开。

本文参考Sylvain Gugger的帖子The 1cycle policy，稍微会有些不同。

插播一句：Dropout 和 BN。

Dropout 和BN在实际使用中，都会被提到加速训练，blabla...那会有什么缺点吗。

一位神秘大佬有一天这样教育菜鸟：

Dropout的缺点

dropout的那个是因为给BP回来的gradient加了一个很大的variance， 虽然sgd训练可以收敛， 但由于方差太大，performance会变差。加了dropout之后，估计的gradient贼差，虽然还是无偏估计但是variance就不可控了。

对卷积层的正则化效果一般

分割线

BN的缺点

BN的是因为改变了loss，所以加了BN之后的网络跟原网络的最优解不是同一个，而且BN之后的网络loss会跟batchsize强相关， 这就是不稳定的原因。

其实BN的缺点在kaiming he的GN的那个paper里面已经说了，但是是从实验上说明的。

只需要进一步的抽象一下，就知道了实验效果的差别是因为loss的差别。

身为PackageMan的我，赶紧掏出小本本记录。

对于其中的方差偏移问题(variance shift)，具体可以参考：知乎专栏

大白话《Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift》和机器之心如何通过方差偏移理解批归一化与Dropout之间的冲突

fastai中代码位置

fastai/callbacks/one_cycle.py

由于one_cycle只是用来帮助加快训练的，但是第一步还是需要找到一个初始的学习率，保证模型的loss在最初是一个较好的状态。我们通过实验来讲解代码。

实验

实验前准备

数据

我们使用cifar10作为实验数据。

```
from fastai.vision import *
# 下载数据并解压
path = untar_data(URLs.CIFAR); path
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

stats = (np.array([ 0.4914 , 0.48216, 0.44653]), np.array([ 0.24703, 0.24349, 0.26159]))

size = 32
batch_size = 512  # 如果性能不好，请调低到2^n，比如16，64

def  get_data(bs):
    ds_tfms = ([*rand_pad(4, 32), flip_lr(p=0.5)], [])
    data = ImageDataBunch.from_folder(path, valid='test', classes=classes,ds_tfms=ds_tfms, b
```

```
=bs).normalize(cifar_stats)
    return data

data = get_data(batch_size)
```

ResNet

在Sylvain的示例中，定义了ResNet和BasicBlock，但我这边建议，跟随最新的pytorch，避免在最的fastai和pytorch使用中报错。

```
def conv3x3(in_planes, out_planes, stride=1):
    """3x3 convolution with padding"""
    return nn.Conv2d(in_planes, out_planes, kernel_size=3, stride=stride,
                     padding=1, bias=False)


def conv1x1(in_planes, out_planes, stride=1):
    """1x1 convolution"""
    return nn.Conv2d(in_planes, out_planes, kernel_size=1, stride=stride, bias=False)

class BasicBlock(nn.Module):
    expansion = 1

    def __init__(self, inplanes, planes, stride=1, downsample=None):
        super(BasicBlock, self).__init__()
        # Both self.conv1 and self.downsample layers downsample the input when stride != 1
        self.conv1 = conv3x3(inplanes, planes, stride)
        self.bn1 = nn.BatchNorm2d(planes)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = conv3x3(planes, planes)
        self.bn2 = nn.BatchNorm2d(planes)
        self.downsample = downsample
        self.stride = stride

    def forward(self, x):
        identity = x

        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)

        out = self.conv2(out)
        out = self.bn2(out)

        if self.downsample is not None:
            identity = self.downsample(x)

        out += identity
        out = self.relu(out)

        return out
```

需要注意的是，在fastai==1.0.48中，构建cnn的函数直接变成了cnn_learner，同时要求bash_arch数是Callable，因此我们需要一定的变化。具体参数查看help(cnn_learner.

Have a look!

参考fastai中resnet18的写法

```
def resnet18(pretrained=False, **kwargs):
    """Constructs a ResNet-18 model.
    Args:
        pretrained (bool): If True, returns a model pre-trained on ImageNet
    """
    model = ResNet(BasicBlock, [2, 2, 2, 2], **kwargs)
    if pretrained:
        model.load_state_dict(model_zoo.load_url(model_urls['resnet18']))
    return model
```

reference: conv2d TypeError

同时，我们不自己写ResNet，而是用fastai直接import的pytorch的定义，保证稳定~~

```
def get_your_model_arch(pretrained=False, **kwargs):
    model_arch = models.ResNet(BasicBlock, [9,9,9,1])
    return model_arch
model_arch = get_your_model_arch
# model2 = models.resnet18() # get exception
model2 = models.resnet18

learn = cnn_learner(data, base_arch=model_arch, metrics=metrics.accuracy)
learn.crit = F.nll_loss
```

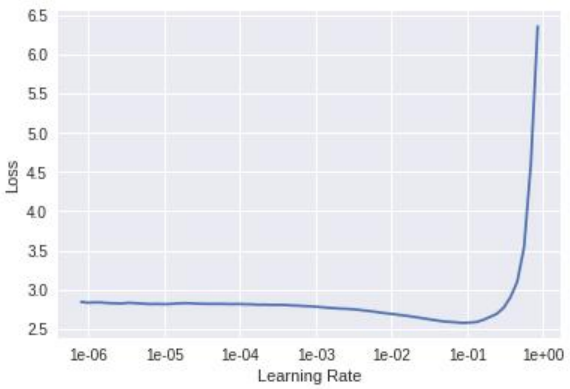
到目前为止，一个CNN就构建好了，大家可以根据ResNet的参数，算一下这个是几层。

lr_find(), 找到你的初始学习率

定义好learner之后，如果要使用one-cycle-policy，那首先我们需要一个最佳的学习率。在fastai中直接使用learn.lr_find就能找到。

找到最佳学习率的思路参考：How Do You Find A Good Learning Rate

```
learn.lr_find(wd=1e-4,end_lr=100)
```



原文链接：用 fastai 解释什么是 one-cycle-policy

我们从图中可以看到，学习率可以设置为0.08，初始的loss大概在2.6.

实验

One-Cycle-Policy大概有三个步骤：

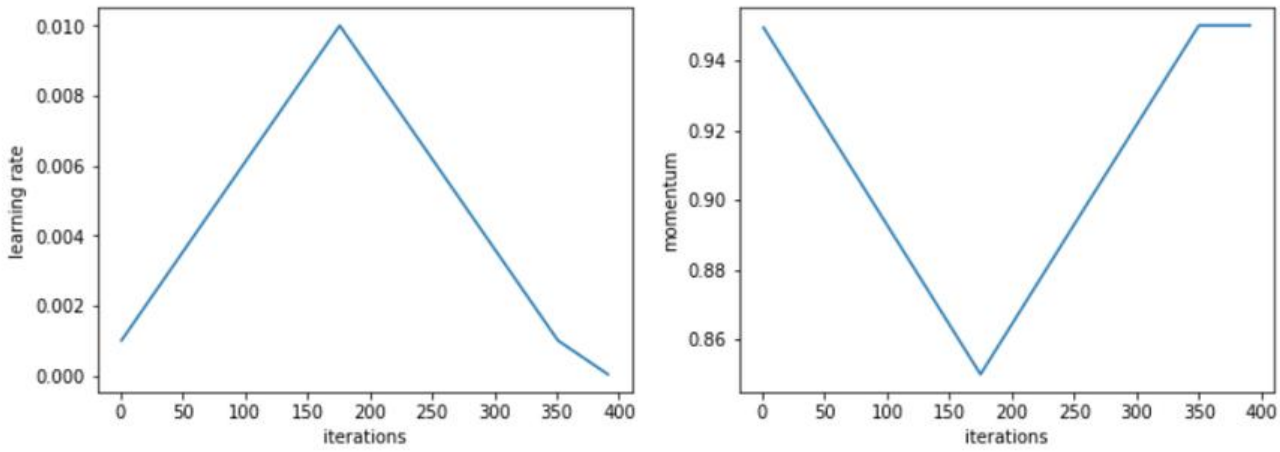
1. 我们逐渐将学习率从lr_max / div_factor提高到lr_max，同时我们逐渐减少从mom_max到mom_mn的动量(momentum)。

2. 反向再做一次：我们逐渐将学习率从lr_max降低到lr_max / div_factor，同时我们逐渐增加从mommin到mom_max的动量。

3. 我们进一步将学习率从lr_max / div_factor降低到lr_max /（div_factor x 100），我们保持动力稳在mom_max。

```
learn.recorder.plot_lr(show_moms=True)
```

大概得到如下图像：



实验1

最大学习率: 0.08, 95轮one_cycle, weight decays:1e-4.

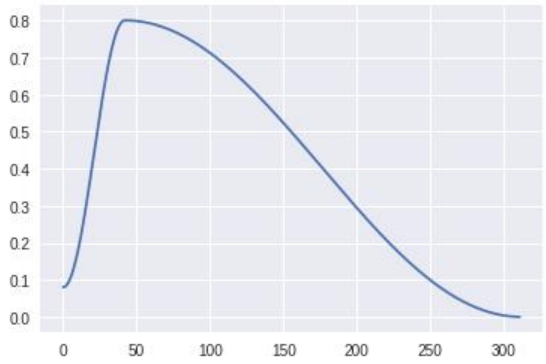
在旧版本中，是use_clr_beta. 新版本中参数进行了修改。

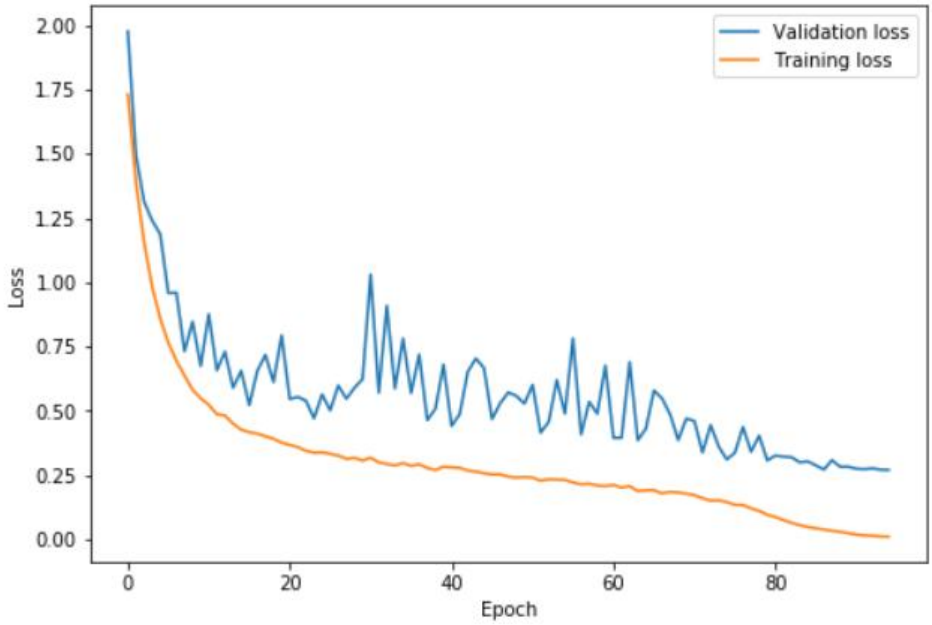
- div_factor:10, pick 1/10th of the maximum learning rate for the minimum learning rate
- pct_start: 0.1368, dedicate 13.68% of the cycle to the annealing at the end (that's 13 epochs over 95)
- moms = (0.95, 0.85)
- maximum momentum 0.95
- minimum momentum 0.85

```
cyc_len = 95
```

```
learn.fit_one_cycle(cyc_len=cyc_len, max_lr=0.08,div_factor=10, pct_start=0.1368,moms=(0.95,
```

0.85), wd=1e-4)





图中可以看到，学习率从0.08上升到0.8，最后下降到了很低的学习率。验证集的loss在中期变得不稳，但在最后降了下来，同时，验证集和训练集的loss差并不大。

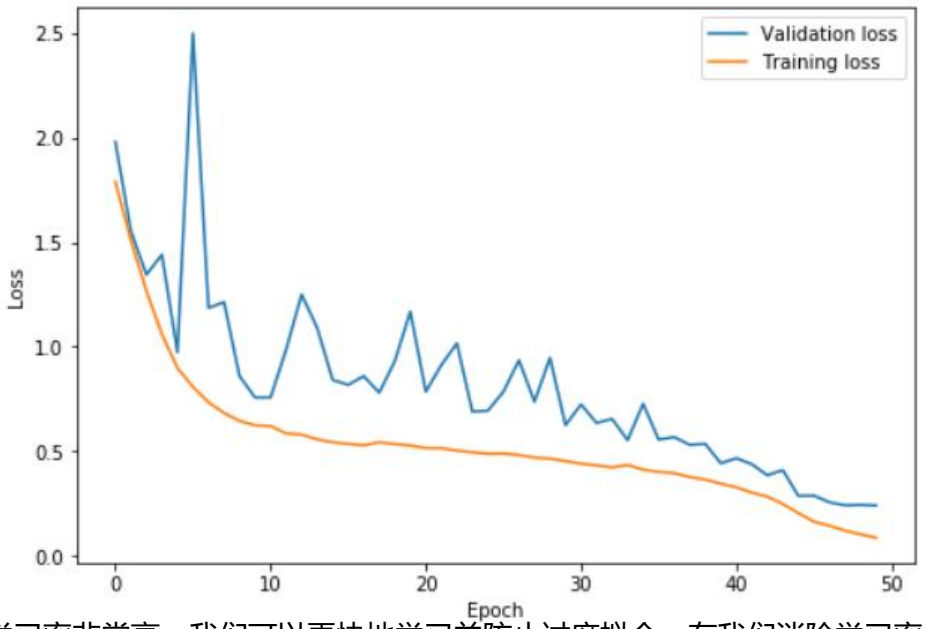
实验2，更高的学习率//更小的cycle

```
# 修改cyc_len和div_factor, pct_start

learn.fit_one_cycle(cyc_len=50, max_lr=0.8,div_factor=10, pct_start=0.5,moms=(0.95, 0.85), w
=1e-4)
```

同时，由于one-cycle学习率衰减的特性，允许我们使用更大的学习率。但为了保证loss不会太大，能会需要更长的cycle来进行

原文链接：用 fastai 解释什么是 one-cycle-policy



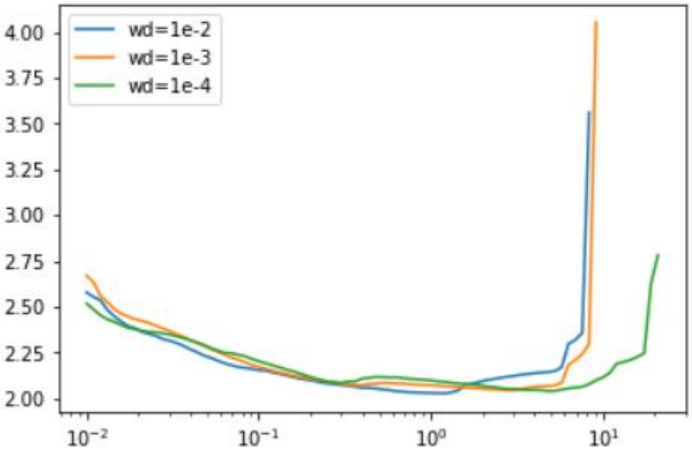
学习率非常高，我们可以更快地学习并防止过度拟合。在我们消除学习率之前，验证损失和训练损失间的差异仍然很小。这是Leslie Smith描述的超收敛现象(super convergence)。

实验3 不变的动量momentum

learn.fit_one_cycle(cyc_len=cyc_len, max_lr=0.08,div_factor=10, pct_start=0.1368,moms=(0.9, .9), wd=1e-4)

在Leslie的实验中，减少动量会带来更好的结果。由此推测，训练时，如果希望SGD快速下降到一个的平坦区域，则新的梯度需要更多的权重。0.85~0.95是经验上的较好值。实验中，发现如果动量使常数，在结果上，会和变化的动量在准确率上接近，但loss会更大一些。时间上，常数动量会快一些。

其他参数的影响：weight decays



(横坐标是学习率，纵坐标是train_loss)

通过对wd的调整，发现不同的wd对学习率和loss是有很大影响的。这也是为什么在lr_find(wd=1e-4)需要和fit_one_cycle(wd=1e-4)相同。

原文链接：用 fastai 解释什么是 one-cycle-policy

结语

one-cycle策略本身还是属于正则化的方法，也可以在模型训练中减少其他正则化方法的使用，因为one-cycle本来可能更有效，也允许在更大的初始学习率中训练更长时间。

原文链接：用 fastai 解释什么是 one-cycle-policy