ResNet

残差网络使得神经网络加入更多的层最起码**不会变差**。

假如第N层从N-1层什么都没学到，但最起码还是能回到N-1层。你可以带着N-1层的有效知识去N+1层训练。

假如你现在的知识储量为100（抽象一下），你新学了一个知识可能会让你对以前的知识理解更加深，也可能让你知识乱套了。

像我考研时候二重积分学到了一个叫雅可比行列式的东西，这个东西是二重积分换元，

用着用着就只会这个办法了，最基础的什么轮换对称性都手生了，这就是知识乱套了。

xxxxxxxxxx net = nn.Sequential(nn.Conv2d(1,6,kernel_size=5),nn.BatchNorm2d(6),
nn.Sigmoid(),nn.MaxPool2d(kernel_size=2,stride=2),
nn.Conv2d(6,16,kernel_size=5),nn.BatchNorm2d(16),
nn.Sigmoid(),nn.MaxPool2d(kernel_size=2,stride=2),
nn.Flatten(),nn.Linear(256,120),nn.BatchNorm1d(120),
nn.Sigmoid(),nn.Linear(120,84),nn.BatchNorm1d(84), nn.Sigmoid(),nn.Linear(84,10))# 使用相同超参数来训练模型d2l.train_ch6(net, train_iter, test_iter, num_epochs, lr, d2l.try_gpu())d2l.plt.show()python

再说白点，这一区块的神经网络对训练有用就保留，没用就相当于跳过。

基于这个特性，所以可以做**很深很深**的神经网络。

$$R(x) = F_2(F_1(x)) + x$$

$$R_n(x) = F_{2,n}(F_{1,n}(R_{n-1}(x))) + R_{n-1}(x)$$

$$R_n(x) = F_{2,n}(F_{1,n}(F_{2,n-1}(F_{1,n-1}(\ldots + R_1(x))))) + R_{n-1}(x)$$

这里的每一层函数可能代表了一个残差块，残差块一般包含卷积池化批量正则激活函数。

Traditonal ResNet Block Code

```
import torch
from torch import nn
from torch.nn import functional as F
from d2l import torch as d2l

# Traditonal ResNet block
class Residual (nn.Module):
    def __init__(self, input_channels, output_channels, use_1x1conv=False,
strides=1): # num_channels为输出channel数
        super().__init__()
        self.conv1 = nn.Conv2d(input_channels, output_channels, kernel_size=3,
padding=1, stride=strides)
        self.conv2 = nn.Conv2d(output_channels, output_channels, kernel_size=3,
padding=1)
        if use_1x1conv:
            self.conv3 = nn.Conv2d(input_channels, output_channels, kernel_size=1,
stride=strides)
        else:
            self.conv3 = None
```

```
        self.bn1 = nn.BatchNorm2d(output_channels)
        self.bn2 = nn.BatchNorm2d(output_channels)
        self.relu = nn.ReLU(inplace=True) # inplace原地操作，不创建新变量，对原变量操
作，节约内存

    def forward(self, X):
        Y = F.relu(self.bn1(self.conv1(X)))
        Y = self.bn2(self.conv2(Y))
        if self.conv3:
            X = self.conv3(X)
        Y += X
        return F.relu(Y)

# 测试Residual
blk = Residual(3,3)
X = torch.rand(4,3,6,6)
Y = blk(X)
print(Y.shape)
blk = Residual(3,6,use_1x1conv=True,strides=2)
print(blk(X).shape) # [4, 6, 3, 3] 因为strides=2

# class Residual为小block，resnet_block 为大block，为Resnet网络的一个stage
# *表示将列表解开，**表示将字典解开
def resnet_block(input_channels,num_channels,num_residuals,first_block=False):
    blk = []
    for i in range(num_residuals):
        # first block 指的是整个结构里的第一个 i=0仅仅是这个block里面第一个
        if i == 0 and not first_block:
            blk.append(Residual(input_channels, num_channels,
use_1x1conv=True,strides=2))
        else:
            blk.append(Residual(num_channels, num_channels))
    return blk

b1 = nn.Sequential(nn.Conv2d(1,64,kernel_size=7,stride=2,padding=3),
                   nn.BatchNorm2d(64),nn.ReLU(),
                   nn.MaxPool2d(kernel_size=3,stride=2,padding=1))
b2 = nn.Sequential(*resnet_block(64,64,2,first_block=True))
b3 = nn.Sequential(*resnet_block(64,128,2)) # b3、b4、b5的首次卷积层都减半
b4 = nn.Sequential(*resnet_block(128,256,2))
b5 = nn.Sequential(*resnet_block(256,512,2))

net =
nn.Sequential(b1,b2,b3,b4,b5,nn.AdaptiveAvgPool2d((1,1)),nn.Flatten(),nn.Linear(512,1
0))
# 观察一下ReNet中不同模块的输入形状是如何变化的
X = torch.rand(size=(1,1,224,224))
for layer in net:
    X = layer(X)
    print(layer.__class__.__name__,'output shape:\t',X.shape)
lr, num_epochs, batch_size = 0.05, 10, 256
train_iter, test_iter = d2l.load_data_fashion_mnist(batch_size, resize=96)
d2l.train_ch6(net, train_iter, test_iter, num_epochs, lr, d2l.try_gpu())
d2l.plt.show()
'''
# 网络各层的输出形状
Conv2d output shape:    torch.Size([1, 64, 112, 112])
MaxPool2d output shape: torch.Size([1, 64, 56, 56])
```

```
Sequential output shape: torch.Size([1, 64, 56, 56])
Sequential output shape: torch.Size([1, 128, 28, 28])
Sequential output shape: torch.Size([1, 256, 14, 14])
Sequential output shape: torch.Size([1, 512, 7, 7])
AdaptiveAvgPool2d output shape:  torch.Size([1, 512, 1, 1])
Flatten output shape:   torch.Size([1, 512])
Linear output shape:    torch.Size([1, 10])
'''
```