Weight decay

常见的处理过拟合的一种方法。

Traditional weight decay code

首先生成多项式

$$y = 0.05 + \sum_{i=1}^{d} 0.01x_i + \epsilon \text{ where } \epsilon \sim \mathcal{N}(0, 0.01^2)$$

```
import torch
from torch import nn
from d2l import torch as d2l
n_train, n_test, num_inputs, batch_size = 20, 100, 200, 5 # 数据越简单，模型越复杂，越容易过拟合。num_inputs为特征维度
true_w, true_b = torch.ones((num_inputs,1)) * 0.01, 0.05
train_data = d2l.synthetic_data(true_w, true_b, n_train) # 生成人工数据集
train_iter = d2l.load_array(train_data, batch_size)
test_data = d2l.synthetic_data(true_w, true_b, n_test)
test_iter = d2l.load_array(test_data, batch_size, is_train=False)
# 初始化模型参数
def init_params():
    w = torch.normal(0,1,size=(num_inputs,1),requires_grad=True)
    b = torch.zeros(1,requires_grad=True)
    return [w,b]

# 定义L2范数惩罚
def l2_penalty(w):
    return torch.sum(w.pow(2)) / 2

def train(lambd):
    w, b = init_params()
    net, loss = lambda X: d2l.linreg(X, w, b), d2l.squared_loss
    num_epochs, lr = 100, 0.003
    animator = d2l.Animator(xlabel='epoch',ylabel='loss',yscale='log',xlim=[5,num_epochs],legend=['train','test'])
    for epoch in range(num_epochs):
        for X, y in train_iter:
            l = loss(net(X),y) + lambd * l2_penalty(w)
            l.sum().backward()
            d2l.sgd([w,b],lr,batch_size)
        if(epoch+1) % 5 == 0:
            if(epoch+1) % 5 ==0:
                animator.add(epoch + 1, (d2l.evaluate_loss(net, train_iter, loss), d2l.evaluate_loss(net,test_iter,loss)))
    print('w的L2范数是',torch.norm(w).item())

train(lambd=0)  # 训练集小，过拟合，测试集损失不下降
d2l.plt.show()
# 使用权重衰退，使得模型在训练过程中不容易过拟合。
train(lambd=30)
d2l.plt.show()
```

Pytorch weight decay code

```
def train_concise(wd):
    net = nn.Sequential(nn.Linear(num_inputs,1))
    for param in net.parameters():
        param.data.normal_()
    loss = nn.MSELoss()
    num_epochs, lr = 100, 0.003
    trainer = torch.optim.SGD([{"params":net[0].weight,"weight_decay":wd},
{"params":net[0].bias}],lr=lr)
    # 惩罚项既可以写在目标函数里面，也可以写在训练算法里面
    # 每一次在更新之前把当前的w乘以衰退因子weight_decay
    animator = d2l.Animator(xlabel='epoch',ylabel='loss',yscale='log',xlim=
[5,num_epochs],legend=['train','test'])
    for epoch in range(num_epochs):
        for X, y in train_iter:
            with torch.enable_grad():
                trainer.zero_grad()
                l = loss(net(X),y)
            l.backward()
            trainer.step()
            if(epoch + 1) % 5 == 0:
                animator.add(epoch + 1, (d2l.evaluate_loss(net, train_iter, loss),
d2l.evaluate_loss(net,test_iter,loss)))
    print('w的L2范数是',net[0].weight.norm().item())
train_concise(0)
d2l.plt.show()
train_concise(3)
d2l.plt.show()
```