第二天

2019年6月8日 9:53

梯度下降

1. 梯度：向量，值是导数，方向是变化最快的方向
2. 导数的计算：▽w =f(w+d)-f(w-d)/2d（d非常小)
3. 更新参数w： w = w-α▽w

链式法则：把复杂的函数用更小的变量去表示包含x的部分

反向传播：先计算最后一层的偏导，之后再计算倒数第二层...

1. 链式法则：
2. 计算图：使用图来表示一个计算，用一个中间变量来表示每一次的计算结果

pytorch实现线性回归

1. tensor 中的require_grad参数
    a. 设置为True，表示会记录该tensor的计算过程
2. tensor中的grad_fn属性
    a. 用来保存计算的过程
3. tensor不保留计算过程
    a. with torch.no_grad():
4. 反向传播：
    a. out.backward()
    b. 导数保存在tensor.grad,默认梯度会累加
5. tensor.data
    a. 获取tensor中值的引用操作
6. tensor.numpy（）
    a. 当tensor中需要计算梯度的时候，grad_fn不为None的时候，tensor.data.numpy()、tensor.detach().numpy()
7. 实现线性回归的逻辑
    a. 准备数据
    b. 初始化参数，进入循环（参数的梯度置为0），计算预测值
    c. 计算loss，loss.backward()计算梯度
    d. 更新参数
8. pytorch通过api完成模型和训练

a. api
    i. nn.Module 构造模型
        1) init：自定义的方法实现的位置
        2) forward：完成一次向前计算的过程
    ii. optimizer优化器类
        1) torch.optim.SGD/Aadm
        2) 流程：
            a) 实例化 Aadm(model.paramters(),lr)
            b) 梯度置为0，optimizer.zero_grad()
            c) 反向传播，计算梯度：loss.backward()
            d) optimizer.step() 参数的更新
    iii. 损失函数
        1) 损失函数对象torch.nn提供
b. 训练的过程：
    i. 实例化模型
    ii. 实例化损失函数
    iii. 实例化优化器类
    iv. 进入循环：
        1) 梯度置为0
        2) 调用模型得到预测值
        3) 调用loss函数，得到损失
        4) loss.backward() 进行梯度计算
        5) optimizer.step()
c. 注意点
    i. model.eval() #把模型置为评估模型
        1) model.training = False
    ii. GPU上运行代码
        1) 自定义的tensor.to(device)
        2) model.to(device)

优化算法：

1. 梯度下降：把所有的数据传入模型，计算平均梯度，更新参数，缺点：慢
2. 随机梯度下降：选一条数据进行参数的更新。缺点：容易受到噪声数据的影响
3. 批梯度下降：每次选择一波数据进行参数的更新。缺点：梯度的变化幅度可能会很大，在最小值附件徘徊
4. 动量法：把历史的梯度考虑进去。更新参数时候使用梯度=历史梯度的指数加权平均
5. adagrad : 更新参数的时候，使用自适应的学习率，= 学习率/历史梯度的平方
6. rmsprop：使用自适应的学习率，= 学习率/历史梯度的平方的指数加权平均
7. adam：动量法和RMSprop的结合版本，既考虑梯度（使用梯度的指数加权平均），又考

虑学习率（让学习率除以历史梯度的平方的指数加权平均）