PyTorch 深度学习实践 第5讲

第5讲  用PyTorch实现线性回归 源代码

B站 刘二大人 ，传送门用PyTorch实现线性回归

PyTorch Fashion(风格)

1、prepare dataset

2、design model using Class  # 目的是为了前向传播forward，即计算y hat(预测值)

3、Construct loss and optimizer (using PyTorch API) 其中，计算loss是为了进行反向传播，optimizer是为了更新梯度。

4、Training cycle (forward,backward,update)

代码说明：

1、Module实现了魔法函数\_\_call\_\_()，call()里面有一条语句是要调用forward()。因此新写的类中需要重写forward()覆盖掉父类中的forward()

2、call函数的另一个作用是可以直接在对象后面加()，例如实例化的model对象，和实例化的linear对象

3、本算法的forward体现是通过以下语句实现的：

y\_pred = model(x\_data)

本实例是批量数据处理，小伙伴们不要被optimizer = torch.optim.SGD(model.parameters(), lr = 0.01)误导了，以为见了SGD就是随机梯度下降。要看传进来的数据是单个的还是批量的。这里的x\_data是3个数据，是一个batch，调用的PyTorch API是 torch.optim.SGD，但这里的SGD不是随机梯度下降，而是批量梯度下降。也就是说，梯度下降算法使用的是随机梯度下降，还是批量梯度下降，还是mini-batch梯度下降，用的API都是 torch.optim.SGD。

由于魔法函数call的实现,model(x\_data)将会调用model.forward(x\_data)函数，model.forward(x\_data)函数中的

y\_pred = self.linear(x)

将会调用torch.nn.Linear类中的forward，至此完成封装，也就是说forward最终是在torch.nn.Linear类中实现的，具体怎么实现，可以不用关心，大概就是y= wx + b。

4、本算法的反向传播，计算梯度是通过以下语句实现的：

loss.backward() # 反向传播，计算梯度

5、本算法的参数(w,b)更新，是通过以下语句实现的：

optimizer.step() # update 参数，即更新w和b的值

6、 每一次epoch的训练过程，总结就是

①前向传播，求y hat （输入的预测值）

②根据y\_hat和y\_label(y\_data)计算loss

③反向传播 backward (计算梯度)

④根据梯度，更新参数

7、传送门 torch.nn.Linear的pytorch文档

|  |
| --- |
| import torch  # prepare dataset  # x,y是矩阵，3行1列 也就是说总共有3个数据，每个数据只有1个特征  x\_data = torch.Tensor([[1.0], [2.0], [3.0]])  y\_data = torch.Tensor([[2.0], [4.0], [6.0]])    #design model using class  """  our model class should be inherit from nn.Module, which is base class for all neural network modules.  member methods \_\_init\_\_() and forward() have to be implemented  class nn.linear contain two member Tensors: weight and bias  class nn.Linear has implemented the magic method \_\_call\_\_(),which enable the instance of the class can  be called just like a function.Normally the forward() will be called  """  class LinearModel(torch.nn.Module):  def \_\_init\_\_(self):  super(LinearModel, self).\_\_init\_\_()  # (1,1)是指输入x和输出y的特征维度，这里数据集中的x和y的特征都是1维的  # 该线性层需要学习的参数是w和b 获取w/b的方式分别是~linear.weight/linear.bias  self.linear = torch.nn.Linear(1, 1)    def forward(self, x):  y\_pred = self.linear(x)  return y\_pred    model = LinearModel()    # construct loss and optimizer  # criterion = torch.nn.MSELoss(size\_average = False)  criterion = torch.nn.MSELoss(reduction = 'sum')  optimizer = torch.optim.SGD(model.parameters(), lr = 0.01) # model.parameters()自动完成参数的初始化操作    # training cycle forward, backward, update  for epoch in range(100):  y\_pred = model(x\_data) # forward:predict  loss = criterion(y\_pred, y\_data) # forward: loss  print(epoch, loss.item())    optimizer.zero\_grad() # the grad computer by .backward() will be accumulated. so before backward, remember set the grad to zero  loss.backward() # backward: autograd，自动计算梯度  optimizer.step() # update 参数，即更新w和b的值    print('w = ', model.linear.weight.item())  print('b = ', model.linear.bias.item())    x\_test = torch.Tensor([[4.0]])  y\_test = model(x\_test)  print('y\_pred = ', y\_test.data) |

————————————————

版权声明：本文为CSDN博主「错错莫」的原创文章，遵循CC 4.0 BY-SA版权协议，转载请附上原文出处链接及本声明。

原文链接：https://blog.csdn.net/bit452/article/details/109677086