用PyTorch类实现Logistic regression

import torch

from torch.autograd import Variable

torch.manual\_seed(2)

x\_data = Variable(torch.Tensor([[1.0], [2.0], [3.0], [4.0]]))

y\_data = Variable(torch.Tensor([[0.0], [0.0], [1.0], [1.0]]))

#初始化

w = Variable(torch.Tensor([-1]), requires\_grad=True)

b = Variable(torch.Tensor([0]), requires\_grad=True)

epochs = 100

costs = []

lr = 0.1

print("before training, predict of x = 1.5 is:")

print("y\_pred = ", float(w.data\*1.5 + b.data > 0))

#模型训练

for epoch in range(epochs):

#计算梯度

A = 1/(1+torch.exp(-(w\*x\_data+b))) #逻辑回归函数

J = -torch.mean(y\_data\*torch.log(A) + (1-y\_data)\*torch.log(1-A)) #逻辑回归损失函数

#J = -torch.mean(y\_data\*torch.log(A) + (1-y\_data)\*torch.log(1-A)) +alpha\*w\*\*2

#基础类进行正则化，加上L2范数

costs.append(J.data)

J.backward() #自动反向传播

#参数更新

w.data = w.data - lr\*w.grad.data

w.grad.data.zero\_()

b.data = b.data - lr\*b.grad.data

b.grad.data.zero\_()

print("after training, predict of x = 1.5 is:")

print("y\_pred =", float(w.data\*1.5+b.data > 0))

print(w.data, b.data)

torch.nn.module写网络结构

import torch

from torch.autograd import Variable

torch.manual\_seed(2)

x\_data = Variable(torch.Tensor([[1.0], [2.0], [3.0], [4.0]]))

y\_data = Variable(torch.Tensor([[0.0], [0.0], [1.0], [1.0]]))

#定义网络模型

#先建立一个基类Module，都是从父类torch.nn.Module继承过来，Pytorch写网络的固定写法

class Model(torch.nn.Module):

def \_\_init\_\_(self):

super(Model, self).\_\_init\_\_() #初始父类

self.linear = torch.nn.Linear(1, 1) #输入维度和输出维度都为1

def forward(self, x):

y\_pred = self.linear(x)

return y\_pred

model = Model() #实例化

#定义loss和优化方法

criterion = torch.nn.BCEWithLogitsLoss() #损失函数，封装好的逻辑损失函数

optimizer = torch.optim.SGD(model.parameters(), lr=0.01) #进行优化梯度下降

#optimizer = torch.optim.SGD(model.parameters(), lr=0.01, weight\_decay=0.001)

#Pytorch类方法正则化方法，添加一个weight\_decay参数进行正则化

#befor training

hour\_var = Variable(torch.Tensor([[2.5]]))

y\_pred = model(hour\_var)

print("predict (before training)given", 4, 'is', float(model(hour\_var).data[0][0]>0.5))

epochs = 40

for epoch in range(epochs):

#计算grads和cost

y\_pred = model(x\_data) #x\_data输入数据进入模型中

loss = criterion(y\_pred, y\_data)

print('epoch = ', epoch+1, loss.data[0])

optimizer.zero\_grad() #梯度清零

loss.backward() #反向传播

optimizer.step() #优化迭代

#After training

hour\_var = Variable(torch.Tensor([[4.0]]))

y\_pred = model(hour\_var)

print("predict (after training)given", 4, 'is', float(model(hour\_var).data[0][0]>0.5))