# 学习代码是 pytorch 深度学习入门与实战 孙玉林 余本国 中国水利水电出版社

# 本地编译器环境

# python 3.10

# torch 2.2.1+cu118 pypi\_0 pypi

# torchaudio 2.2.1+cu118 pypi\_0 pypi

# torchvision 0.17.1+cu118 pypi\_0 pypi

# torch的数据结构

import torch

print(torch.tensor([1.2, 2.4]).dtype)

# torch.float32

torch.set\_default\_tensor\_type(torch.DoubleTensor)

print(torch.tensor([1.2, 2.4]).dtype) # CPU tensor中的DoubleTensor对应dtype中的torch.float64

# torch.float64

a = torch.tensor([1.2, 2.4]) # 强制类型转换

print('a.dtype', a.dtype)

# a.dtype torch.float64

print(a.long().dtype)

# torch.int64

print(a.int().dtype)

# torch.int32

print(a.float().dtype)

# torch.float32

print(a.double().dtype)

# torch.float64

torch.set\_default\_tensor\_type(torch.FloatTensor) # 转回去

print(torch.tensor([1.2, 2.4]).dtype)

# torch.float32

print('get\_default\_dtype', torch.get\_default\_dtype()) # 获取默认的数据类型

# get\_default\_dtype torch.float32

# 基本tensor创建

import torch

A = torch.tensor([[1, 2], [3, 4]])

print(A.numel()) # numel() 方法返回张量中的元素数量 number of elements

# 4

B = torch.tensor((1, 2, 3), dtype=torch.float32, requires\_grad=True) # 注意只有浮点类型可以计算梯度

print(B)

# tensor([1., 2., 3.], requires\_grad=True)

y = B.pow(2).sum()

print(y)

# tensor(14., grad\_fn=<SumBackward0>)

y.backward()

print(B.grad)

# tensor([2., 4., 6.])

# 除了 torch.tensor() 外还有 torch.Tensor() 可构造张量

D = torch.Tensor(2, 3) # 特定尺寸

print(D)

# tensor([[0., 0., 0.],

# [0., 0., 0.]])

print(torch.ones\_like(D)) # 全1

# tensor([[1., 1., 1.],

# [1., 1., 1.]])

print(torch.zeros\_like(D)) # 全0

# tensor([[0., 0., 0.],

# [0., 0., 0.]])

print(torch.rand\_like(D)) # 随机数

# tensor([[0.1927, 0.3265, 0.9107],

# [0.6054, 0.7528, 0.0278]])

E = [[1,2],[3,4]]

print(E)

E = D.new\_tensor(E)

print(E)

print(D) #D 和 E各自是什么

# tensor([[1., 2.],

# [3., 4.]])

# tensor([[-8.8013e+12, 1.1799e-42, 0.0000e+00],

# [ 0.0000e+00, 0.0000e+00, 0.0000e+00]])

print(E.dtype)

print(D.dtype) # tensor.new\_tensor()函数的具体作用，数据类型相同

# torch.float32

# torch.float32

E = D.new\_full((3,3),fill\_value=2) # 3\*3 使用2 填充的张量

print(E)

# tensor([[2., 2., 2.],

# [2., 2., 2.],

# [2., 2., 2.]])

import numpy as np

F = np.ones((3,3))

print(F)

# [[1. 1. 1.]

# [1. 1. 1.]

# [1. 1. 1.]]

Ftensor = torch.as\_tensor(F) # numpy 与 pytorch张量的互换

print(Ftensor)

# tensor([[1., 1., 1.],

# [1., 1., 1.],

# [1., 1., 1.]], dtype=torch.float64)

Ftensor = torch.from\_numpy(F) # numpy 与 pytorch张量的互换

print(Ftensor)

# tensor([[1., 1., 1.],

# [1., 1., 1.],

# [1., 1., 1.]], dtype=torch.float64)

torch.manual\_seed(123)

A = torch.normal(mean=0.0, std=torch.tensor(1.0))

print(A)

# tensor(-0.1115)

A = torch.normal(mean=torch.arange(1 , 5.0), std=torch.arange(1.0 , 5)) #mean随机数的均值，std随机数的标准差

print(A)

# tensor([ 1.1204, 1.2607, 2.2787, -0.7877])

B = torch.rand(3,4) # 在【0，1】的均匀分布

print(B)

# tensor([[0.0756, 0.1966, 0.3164, 0.4017],

# [0.1186, 0.8274, 0.3821, 0.6605],

# [0.8536, 0.5932, 0.6367, 0.9826]])

C = torch.randperm(10) # 随机排序后输出

print(C)

# tensor([9, 1, 7, 6, 3, 4, 5, 8, 0, 2])

D = torch.logspace(start=0.1, end=1.0, steps=5) # 以对数为间隔的张量

print(D)

# tensor([ 1.2589, 2.1135, 3.5481, 5.9566, 10.0000])

# 张量操作

import torch

A = torch.arange(12.0).reshape(3, 4)

print(A)

# tensor([[ 0., 1., 2., 3.],

# [ 4., 5., 6., 7.],

# [ 8., 9., 10., 11.]])

A = A.resize\_(2, 6) # 形状修改 跟reshape()类似

print(A)

# tensor([[ 0., 1., 2., 3., 4., 5.],

# [ 6., 7., 8., 9., 10., 11.]])

B = A.resize\_as(A) # 形状修改 跟reshape()类似

print(B)

# tensor([[ 0., 1., 2., 3., 4., 5.],

# [ 6., 7., 8., 9., 10., 11.]])

x = torch.tensor([[1], [2], [3]])

print("原始张量：")

print(x)

print("原始张量的形状：", x.shape)

# 原始张量：

# tensor([[1],

# [2],

# [3]])

# 原始张量的形状： torch.Size([3, 1])

# 使用squeeze()函数去除维度为1的维度

y = torch.squeeze(x)

print("\n去除维度为1后的张量：")

print(y)

print("去除维度为1后的张量形状：", y.shape)

# 去除维度为1后的张量：

# tensor([1, 2, 3])

# 去除维度为1后的张量形状： torch.Size([3])

C = torch.arange(24).reshape(2, 3, 4)

C = C.reshape(1, 1, 3, 4, 2, 1, 1)

print(C)

print(C.shape)

# torch.Size([1, 1, 3, 4, 2, 1, 1])

C = torch.squeeze(C)

print(C)

print(C.shape)

# torch.Size([3, 4, 2]) #从这个实验可以看出 squeeze()是把维度数量为1 的全部都降维掉了

A = torch.arange(3)

B = A.expand(3, -1)

print(B)

# tensor([[0, 1, 2],

# [0, 1, 2],

# [0, 1, 2]])

D = B.repeat(1, 2, 2) # 重复填充

print(D)

print(D.shape)

# tensor([[[0, 1, 2, 0, 1, 2],

# [0, 1, 2, 0, 1, 2],

# [0, 1, 2, 0, 1, 2],

# [0, 1, 2, 0, 1, 2],

# [0, 1, 2, 0, 1, 2],

# [0, 1, 2, 0, 1, 2]]])

A = torch.arange(12).reshape(1, 3, 4)

B = -A

print(torch.where(A > 5, A, B)) # torch.where() 当A >5 为真时返回x 对应位置值 ， 假时返回y的值

print(A[A > 5])

# tensor([ 6, 7, 8, 9, 10, 11])

print(A)

# tensor([[[ 0, 1, 2, 3],

# [ 4, 5, 6, 7],

# [ 8, 9, 10, 11]]])

print(torch.tril(A,diagonal=0)) # diagonal控制要考虑的对角线 下三角

# tensor([[[ 0, 0, 0, 0],

# [ 4, 5, 0, 0],

# [ 8, 9, 10, 0]]])

print(torch.triu(A,diagonal=1)) # 上三角

# tensor([[[ 0, 1, 2, 3],

# [ 0, 0, 6, 7],

# [ 0, 0, 0, 11]]])

A = A.reshape(3,4)

print(A)

# tensor([[[ 0, 1, 2, 3],

# [ 4, 5, 6, 7],

# [ 8, 9, 10, 11]]])

print(torch.diag(A , diagonal=0))

# tensor([ 0, 5, 10])

print(torch.diag(A,diagonal=1))

# tensor([ 1, 6, 11])

print(torch.diag(A,diagonal=-1))

# tensor([4, 9])

# 张量计算

import torch

A = torch.tensor([10.0])

B = torch.tensor([10.1])

A = torch.tensor(float('nan'))

print(torch.allclose(A, A, equal\_nan=True)) # equal\_nan 为真时 ，判断缺失值nan为接近

print(torch.allclose(A, A, equal\_nan=False))

# True

# False

A = torch.tensor([1, 2, 3, 4, 5, 6])

B = torch.arange(1, 7)

C = torch.unsqueeze(B, dim=0)

print(C)

# tensor([[1, 2, 3, 4, 5, 6]])

print(torch.ge(A, B)) # 判断大于等于

# tensor([True, True, True, True, True, True])

print(torch.ge(A, C))

# tensor([[True, True, True, True, True, True]])

print(torch.gt(A, B)) # 判断大于

# tensor([False, False, False, False, False, False])

print(torch.gt(A, C))

# tensor([[False, False, False, False, False, False]])

A = torch.arange(6.0).reshape(2, 3)

print(A)

# tensor([[0., 1., 2.],

# [3., 4., 5.]])

print(torch.rsqrt(A)) # reciprocal square root 与下一行代码操作结果相同

# tensor([[ inf, 1.0000, 0.7071],

# [0.5774, 0.5000, 0.4472]])

print(1 / (A \*\* 0.5))

# tensor([[ inf, 1.0000, 0.7071],

# [0.5774, 0.5000, 0.4472]])

A = torch.arange(12.0).reshape(2, 2, 3)

B = torch.arange(12.0).reshape(2, 3, 2)

print(A)

# tensor([[[ 0., 1., 2.],

# [ 3., 4., 5.]],

#

# [[ 6., 7., 8.],

# [ 9., 10., 11.]]])

print(B)

# tensor([[[ 0., 1.],

# [ 2., 3.],

# [ 4., 5.]],

#

# [[ 6., 7.],

# [ 8., 9.],

# [10., 11.]]])

print(A[0])

# tensor([[0., 1., 2.],

# [3., 4., 5.]])

print(B[0])

# tensor([[0., 1.],

# [2., 3.],

# [4., 5.]])

# 自动微分

import torch

x = torch.tensor([[1.0, 2.0], [3.0, 4.0]], requires\_grad=True)

y = torch.sum(x \*\* 2 + x \* 2 + 1)

print(x.requires\_grad)

print(y.requires\_grad)

# True

# True

print(x)

# tensor([[1., 2.],

# [3., 4.]], requires\_grad=True)

print(y)

# tensor(54., grad\_fn=<SumBackward0>)

y.backward()

print(x.grad) # 导数结果是 2\*x + 2 ， 即对应梯度

# tensor([[ 4., 6.],

# [ 8., 10.]])

# nn模块 做卷积

import torch

import torch.nn as nn

import matplotlib.pyplot as plt

import numpy as np

from PIL import Image

myim = Image.open("./123.jpg")

myimgray = np.array(myim.convert("L"), dtype=np.float32) # 转为灰度图像，将结果转换为float32

plt.figure(figsize=(6, 6))

plt.imshow(myimgray, cmap=plt.cm.gray)

plt.axis("off")

plt.show()

imh, imw = myimgray.shape

myimgray\_t = torch.from\_numpy(myimgray.reshape((1, 1, imh, imw)))

print(myimgray\_t.shape)

# torch.Size([1, 1, 512, 512]) batch, channel , height, width 转换格式才能继续做卷积操作

kersize = 5

ker = torch.ones(kersize, kersize, dtype=torch.float32) \* -1 # 注意创建张量的代码写法，这里元素全部为-1

ker[2, 2] = 24 # 5 \* 5 的矩阵的中心元素[2,2]为24

ker = ker.reshape((1, 1, kersize, kersize)) # 转换格式才能继续做卷积操作

conv2d = nn.Conv2d(1, 2, (kersize, kersize), bias=False) # 定义卷积 ，channel为2

conv2d.weight.data[0] = ker # 将定义的卷积层 conv2d 的第一个卷积核的权重设置为之前创建的 ker 张量。在卷积神经网络中，卷积层的权重参数就是卷积核，它用来提取输入数据的特征

imconv2dout = conv2d(myimgray\_t) # 卷积操作

imconv2dout\_im = imconv2dout.data.squeeze() # 压缩

print(imconv2dout\_im.shape)

# torch.Size([2, 508, 508])

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.imshow(imconv2dout\_im[0], cmap=plt.cm.gray)

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(imconv2dout\_im[1], cmap=plt.cm.gray)

plt.axis("off")

plt.show()

maxpool2 = nn.MaxPool2d(2, stride=2) # 最大池化定义pool of square window of size=2

pool2\_out = maxpool2(imconv2dout)

pool2\_out\_im = pool2\_out.squeeze()

print(pool2\_out.shape)

# torch.Size([1, 2, 254, 254])

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.imshow(pool2\_out\_im[0].data, cmap=plt.cm.gray)

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(pool2\_out\_im[1].data, cmap=plt.cm.gray)

plt.axis("off")

plt.show()

avgpool2 = nn.AvgPool2d(2, stride=2) # 平均值池化

pool2\_out = avgpool2(imconv2dout)

pool2\_out\_im = pool2\_out.squeeze()

print(pool2\_out.shape)

# torch.Size([1, 2, 254, 254])

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.imshow(pool2\_out\_im[0].data, cmap=plt.cm.gray)

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(pool2\_out\_im[1].data, cmap=plt.cm.gray)

plt.axis("off")

plt.show()

Adaavgpool2 = nn.AdaptiveAvgPool2d((100, 200)) # 自适应平均值池化

pool2\_out = Adaavgpool2(imconv2dout)

pool2\_out\_im = pool2\_out.squeeze()

print(pool2\_out.shape)

# torch.Size([1, 2, 100, 200])

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.imshow(pool2\_out\_im[0].data, cmap=plt.cm.gray)

plt.axis("off")

plt.subplot(1, 2, 2)

plt.imshow(pool2\_out\_im[1].data, cmap=plt.cm.gray)

plt.axis("off")

plt.show()

# 激活函数

x = torch.linspace(-6, 6, 100)

sigmoid = nn.Sigmoid() # 定义Sigmoid激活函数

ysigmoid = sigmoid(x)

tanh = nn.Tanh() # 定义Tanh激活函数

ytanh = tanh(x)

relu = nn.ReLU() # 定义ReLU激活函数

yrelu = relu(x)

softplus = nn.Softplus() # 定义Softplus激活函数

ysoftplus = softplus(x)

plt.figure(figsize=(14, 3))

plt.subplot(1, 4, 1)

plt.plot(x.data.numpy(), ysigmoid.data.numpy(), "r-")

plt.title("Sigmoid")

plt.grid()

plt.subplot(1, 4, 2)

plt.plot(x.data.numpy(), ytanh.data.numpy(), "r-")

plt.title("Tanh")

plt.grid()

plt.subplot(1, 4, 3)

plt.plot(x.data.numpy(), yrelu.data.numpy(), "r-")

plt.title("Relu")

plt.grid()

plt.subplot(1, 4, 4)

plt.plot(x.data.numpy(), ysoftplus.data.numpy(), "r-")

plt.title("Softplus")

plt.grid()

plt.show()