深度学习框架PyTorch

实验目标

通过本案例的学习和课后作业的练习：

1. 创建张量和张量的基本操作；
2. 构造一个卷积神经网络实现对手写数字的识别。

你也可以将本案例相关的 ipynb 学习笔记分享到 AI Gallery Notebook (https://marketplace.huaweicloud.com/markets/aihub/notebook/list/) 版块获得成长值 (https://marketplace.huaweicloud.com/markets/aihub/article/detail/?content_id=9b8d7e7a-a150-449e-ac17-2dcf76d8b492)，分享方法请查看此文档 (https://marketplace.huaweicloud.com/markets/aihub/article/detail/?content_id=8afec58a-b797-4bf9-acca-76ed512a3acb)。

Pytorch介绍

PyTorch 是最新的深度学习框架之一，由 Facebook 的团队开发，并于 2017 年在 GitHub 上开源。

PyTorch 很简洁、易于使用、支持动态计算图而且内存使用很高效，因此越来越受欢迎。

PyTorch和TensorFlow的比较

- PyTorch本质上是Numpy的替代者，而且支持GPU、带有高级功能，可以用来搭建和训练深度神经网络。如果你熟悉Numpy、Python以及常见的深度学习概念（卷积层、循环层、 SGD等），会非常容易上手PyTorch。
- 而TensorFlow可以看成是一个嵌入Python的编程语言。你写的TensorFlow代码会被Python编译成一张图，然后由TensorFlow执行引擎运行。很多新手就会因为这个增加的间接层而困扰。也正是因为同样的原因，TensorFlow有一些额外的概念需要学习，例如会话、图、变量作用域（variable scoping）、占位符等。另外还需要更多的样板代码才能让一个基本的模型运行。所以TensorFlow的上手时间，肯定要比PyTorch长。
- pytorch的计算图是动态的，可以根据计算需要实时改变计算图,对比静态的 Tensorflow, 它能更有效地处理一些问题。

本案例推荐的理论学习视频：

- 《AI基础课程--常用框架工具》 深度学习框架PyTorch (https://education.huaweicloud.com/courses/course-v1:HuaweiX+CBUCNXE081+Self-paced/courseware/364dbe7d5197438ebb9772f60a7b8723/8e2e5d0c816743809691da152424afe4/)

注意事项

1. 本案例推荐使用AI引擎：**Pytorch-1.0.0**；
2. 本案例可以使用CPU运行，但推荐切换为使用 GPU 运行，请查看《ModelArts JupyterLab 硬件规格使用指南》(https://marketplace.huaweicloud.com/markets/aihub/article/detail/?content_id=638c55c5-816d-421c-9b5f-90c804b1fa6b)了解切换硬件规格的方法；
3. 如果您是第一次使用 JupyterLab，请查看《ModelArts JupyterLab使用指导》(https://marketplace.huaweicloud.com/markets/aihub/article/detail/?content_id=03676d0a-0630-4a3f-b62c-07fba43d2857)了解使用方法；
4. 如果您在使用 JupyterLab 过程中碰到报错，请参考《ModelArts JupyterLab常见问题解决办法》(https://marketplace.huaweicloud.com/markets/aihub/article/detail/?content_id=9ad8ce7d-06f7-4394-80ef-4dbf6cfb4be1)尝试解决问题。

实验步骤

1. 创建张量

In [1]:
```
import torch

x = torch.empty(5, 3)  # 未初始化的张量
print(x)
```

```
tensor([[2.0004e+15, 4.5863e-41, 2.0004e+15],
        [4.5863e-41, 1.6457e+19, 1.4585e-19],
        [6.7421e+22, 5.0761e+31, 1.3556e-19],
        [7.2053e+22, 4.7428e+30, 1.8580e-19],
        [2.4755e-12, 3.7282e-08, 1.3567e-19]])
```

全0的张量

In [2]:
```
x = torch.zeros(5, 3, dtype=torch.long)
print(x)
```

```
tensor([[0, 0, 0],

        [0, 0, 0],

        [0, 0, 0],

        [0, 0, 0],

        [0, 0, 0]])
```

全1张量

In [3]:
```
x = torch.ones(5, 3, dtype=torch.long)
print(x)
```

```
tensor([[1, 1, 1],

        [1, 1, 1],

        [1, 1, 1],

        [1, 1, 1],

        [1, 1, 1]])
```

直接通过数据创建张量

In [4]:
```
x = torch.tensor([5.5, 3])
print(x)
```

```
tensor([5.5000, 3.0000])
```

创建随机数据的张量

In [5]:
```
x = torch.rand(5, 3) # 随机数据的矩阵
print(x)
```

```
tensor([[0.5132, 0.0148, 0.7240],

        [0.9469, 0.7166, 0.3563],

        [0.8657, 0.1148, 0.9135],

        [0.0744, 0.0366, 0.6522],

        [0.9637, 0.1378, 0.9998]])
```

2. 张量的操作

判断是否是张量

```
In [6]: torch.is_tensor(x)
```

```
Out[6]: True
```

切片操作

```
In [7]: x[0, 2]
```

```
Out[7]: tensor(0.7240)
```

查看张量维度

```
In [8]: print(x.size())
```

```
        torch.Size([5, 3])
```

张量间的合并。Cat方法可以将两个数据进行合并，但是要保证合并的方向上维度是相等的

```
In [9]: a = torch.ones((6, 3))
        torch.cat((x, a), 0)
```

```
Out[9]: tensor([[0.5132, 0.0148, 0.7240],
                [0.9469, 0.7166, 0.3563],
                [0.8657, 0.1148, 0.9135],
                [0.0744, 0.0366, 0.6522],
                [0.9637, 0.1378, 0.9998],
                [1.0000, 1.0000, 1.0000],
                [1.0000, 1.0000, 1.0000],
                [1.0000, 1.0000, 1.0000],
                [1.0000, 1.0000, 1.0000],
                [1.0000, 1.0000, 1.0000],
                [1.0000, 1.0000, 1.0000]])
```

张量运算

In [10]:
```
x = torch.ones(5, 3)
y = torch.rand(5, 3)

print(y)
print(x + y)
```

```
tensor([[0.2254, 0.4511, 0.9180],

        [0.7136, 0.3726, 0.6671],

        [0.9530, 0.2009, 0.8919],

        [0.0727, 0.9779, 0.3924],

        [0.9939, 0.8354, 0.9298]])

tensor([[1.2254, 1.4511, 1.9180],

        [1.7136, 1.3726, 1.6671],

        [1.9530, 1.2009, 1.8919],

        [1.0727, 1.9779, 1.3924],

        [1.9939, 1.8354, 1.9298]])
```

使用add方法实现加法操作

In [11]:
```
print(torch.add(x, y))
```

```
tensor([[1.2254, 1.4511, 1.9180],

        [1.7136, 1.3726, 1.6671],

        [1.9530, 1.2009, 1.8919],

        [1.0727, 1.9779, 1.3924],

        [1.9939, 1.8354, 1.9298]])
```

张量转置

In [12]:
```
x = torch.rand(5, 3)

print(x)
print(torch.t(x))
```

```
tensor([[0.2151, 0.5589, 0.9562],

        [0.8276, 0.5394, 0.8051],

        [0.9407, 0.4313, 0.0213],

        [0.5521, 0.6539, 0.0443],

        [0.5406, 0.9499, 0.7963]])

tensor([[0.2151, 0.8276, 0.9407, 0.5521, 0.5406],

        [0.5589, 0.5394, 0.4313, 0.6539, 0.9499],

        [0.9562, 0.8051, 0.0213, 0.0443, 0.7963]])
```

张量之间的比较

In [13]:
```
x = torch.eye(3)
y = torch.ones((3, 3))

print(x)
print(y)
print(torch.eq(x, y) )
print(torch.equal(x, y))
```

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

        [0., 1., 0.],

        [0., 0., 1.]])

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

        [1., 1., 1.],

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

tensor([[1, 0, 0],

        [0, 1, 0],

        [0, 0, 1]], dtype=torch.uint8)

False
```

3. 手写数字识别

导入工具包

In [14]:
```
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms
```

创建超参数

In [15]:
```
# 超参数
num_epochs = 1          # 训练轮数
num_classes = 10        # 类别
batch_size = 100        # batch大小
learning_rate = 0.001   # 学习率

# 选择使用的硬件
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cpu
```

从华为云OBS下载数据

In [16]:
```
# 下载MNIST数据集
!wget -N https://modelarts-labs-bj4.obs.cn-north-4.myhuaweicloud.com/course/hwc_edu/python_module_framework/datasets/pytorch_data/MNIST_data.zip
# 下载测试图片
!wget -N https://modelarts-labs-bj4.obs.cn-north-4.myhuaweicloud.com:443/course/hwc_edu/python_module_framework/datasets/pytorch_data/num.png
```

```
--2021-07-02 16:18:59--  https://modelarts-labs-bj4.obs.cn-north-4.myhuaweicloud.com/course/hwc_edu/python_module_framework/datasets/pytorch_data/MNIST_data.zip

Resolving proxy-notebook.modelarts-dev-proxy.com (proxy-notebook.modelarts-dev-proxy.com)... 192.168.0.172

Connecting to proxy-notebook.modelarts-dev-proxy.com (proxy-notebook.modelarts-dev-proxy.com)|192.168.0.172|:8083... connected.

Proxy request sent, awaiting response... 200 OK

Length: 21525536 (21M) [application/zip]

Saving to: ‘MNIST_data.zip’



MNIST_data.zip      100%[===================>]  20.53M   135MB/s    in 0.2s



2021-07-02 16:18:59 (135 MB/s) - ‘MNIST_data.zip’ saved [21525536/21525536]



--2021-07-02 16:18:59--  https://modelarts-labs-bj4.obs.cn-north-4.myhuaweicloud.com/course/hwc_edu/python_module_framework/datasets/pytorch_data/num.png

Resolving proxy-notebook.modelarts-dev-proxy.com (proxy-notebook.modelarts-dev-proxy.com)... 192.168.0.172

Connecting to proxy-notebook.modelarts-dev-proxy.com (proxy-notebook.modelarts-dev-proxy.com)|192.168.0.172|:8083... connected.

Proxy request sent, awaiting response... 200 OK

Length: 612 [image/png]

Saving to: ‘num.png’



num.png             100%[===================>]     612  --.-KB/s    in 0s



2021-07-02 16:18:59 (10.7 MB/s) - ‘num.png’ saved [612/612]
```

解压MNIST数据集

In [17]:
```
import os
if not os.path.exists('MNIST_data'):
    os.system("unzip MNIST_data.zip")
```

In [18]:
```
# 加载 MNIST 数据集
train_dataset = torchvision.datasets.MNIST(root='./MNIST_data',
                                           train=True,
                                           transform=transforms.ToTenso
r(),
                                           download=False)  # 如果将down
load设置为True可下载MNIST数据

test_dataset = torchvision.datasets.MNIST(root='./MNIST_data',
                                          train=False,
                                          transform=transforms.ToTensor
(),
                                          download=False)

train_loader = torch.utils.data.DataLoader(dataset=train_dataset,
                                           batch_size=batch_size,
                                           shuffle=True)

test_loader = torch.utils.data.DataLoader(dataset=test_dataset,
                                          batch_size=batch_size,
                                          shuffle=False)
```

创建模型

In [19]:
```
class ConvNet(nn.Module):

    def __init__(self, num_classes=10):
        super(ConvNet, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(1, 16, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(16),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))

        self.layer2 = nn.Sequential(
            nn.Conv2d(16, 32, kernel_size=5, stride=1, padding=2),
            nn.BatchNorm2d(32),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        self.fc = nn.Linear(7 * 7 * 32, num_classes)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.reshape(out.size(0), -1)
        out = self.fc(out)
        return out


model = ConvNet(num_classes).to(device)
model
```

Out[19]:
```
ConvNet(
  (layer1): Sequential(
    (0): Conv2d(1, 16, kernel_size=(5, 5), stride=(1, 1), padding=(2,
2))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ce
il_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1), padding=
(2, 2))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_
running_stats=True)
    (2): ReLU()
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ce
il_mode=False)
  )
  (fc): Linear(in_features=1568, out_features=10, bias=True)
)
```

优化器和损失函数

In [20]:
```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

开始训练

CPU训练一个epoch耗时约2分钟，GPU训练一个epoch耗时约5秒钟

In [21]:
```
total_step = len(train_loader)
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)

        outputs = model(images)
        loss = criterion(outputs, labels)

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i+1) % 100 == 0:
            print ('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                   .format(epoch+1, num_epochs, i+1, total_step, loss.i
tem()))
```

```
Epoch [1/1], Step [100/600], Loss: 0.2603

Epoch [1/1], Step [200/600], Loss: 0.0757

Epoch [1/1], Step [300/600], Loss: 0.1397

Epoch [1/1], Step [400/600], Loss: 0.0653

Epoch [1/1], Step [500/600], Loss: 0.0465

Epoch [1/1], Step [600/600], Loss: 0.0500
```

在测试集上的表现

In [22]:
```
model.eval()  # 预测模式
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_loader:
        images = images.to(device)
        labels = labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print('Test Accuracy of the model on the 10000 test images: {} %'.forma
t(100 * correct / total))
```

```
Test Accuracy of the model on the 10000 test images: 98.44 %
```

保存模型到本地文件

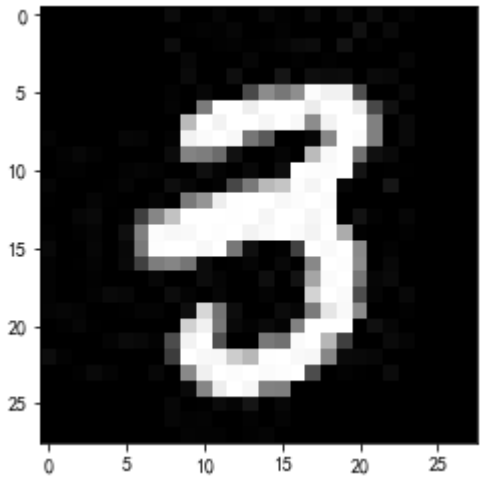
In [23]:
```
torch.save(model.state_dict(), 'model.ckpt')
```

查看测试图片

In [24]:
```
import numpy as np
import cv2
import matplotlib.pyplot as plt
%matplotlib inline

# 图片路径
img_path = "./num.png"
img_file = cv2.imread(img_path, 0)
img_file = cv2.resize(img_file, (28, 28))
plt.imshow(img_file, 'gray')
plt.show()

data_test = img_file
data_test = np.float32(data_test.reshape(1, 28 * 28))
print(data_test.shape)
```




```
(1, 784)
```

预测测试图片

In [25]:
```
py = model(torch.from_numpy(data_test.reshape(1, 1, 28, 28)).to(devic
e))
predicted = torch.max(py, 1)  # 获取分类结果

print(predicted[1])
```

```
tensor([3])
```

可以看到最终预测值为3，是准确的，同学们可以上传自己写的数字看看预测结果如何。

以上是 Pytorch 的基本使用，受限于篇幅原因，本案例未完全覆盖 Pytorch 中的全部操作，欢迎你将更全面的 Pytorch 学习笔记分享到 AI Gallery Notebook (https://marketplace.huaweicloud.com/markets/aihub/notebook/list/) 版块获得成长值 (https://marketplace.huaweicloud.com/markets/aihub/article/detail/?content_id=9b8d7e7a-a150-449e-ac17-2dcf76d8b492)，分享方法请查看此文档 (https://marketplace.huaweicloud.com/markets/aihub/article/detail/?content_id=8afec58a-b797-4bf9-acca-76ed512a3acb)。

作业

本案例“获取测试数据”的这一步是从外部下载了一张图片，其实在下载 `MNIST` 数据集时 `test_dataset` 对象已经加载了测试图片，请你利用本课程中学到的知识和已掌握的知识，完成以下编程题：

1. 获取MNIST数据集中的测试数据集test_dataset对象，从 `test_dataset` 对象中取出一张图片，并保存到本地。 (https://marketplace.huaweicloud.com/markets/aihub/notebook/detail/?id=0956b622-682b-4227-97f2-7bea97e598d8)