

Artificial Intelligence

人工智能实验

深度学习

中山大学计算机学院
2024年春季

目录

1. 理论课内容回顾

1.1 PyTorch介绍

1.2 CNN 网络训练实例

2. 实验任务

2.1 中药图片分类任务

3. 作业提交说明

1.1 PyTorch 介绍

□ PyTorch 安装

- 官网: <https://pytorch.org>
- CPU 版安装（无 Nvidia 显卡）：直接在官网主页选择配置，然后复制生成的 Command 粘贴到终端运行（注意需提前激活 Python 虚拟环境）。
- 建议：安装1.7.0及以上版本。

PyTorch Build	Stable (2.3.0)		Preview (Nightly)	
Your OS	Linux	Mac	Windows	
Package	Conda	Pip	LibTorch	Source
Language	Python		C++ / Java	
Compute Platform	CUDA 11.8	CUDA 12.1	ROCm 6.0	CPU
Run this Command:	<pre>pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118</pre>			

1.1 PyTorch 介绍

GPU 版安装：先在终端使用 `nvidia-smi` 命令查看当前显卡支持的 CUDA 版本

```
(base) PS C:\Users\yanghl> nvidia-smi
Mon May 23 19:42:28 2022

+-----+
| NVIDIA-SMI 512.77      Driver Version: 512.77      CUDA Version: 11.6  |
+-----+-----+
| GPU   Name                TCC/WDDM  Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf    Pwr:Usage/Cap |      Memory-Usage     | GPU-Util  Compute M. |
|                                           MIG M.               |
+-----+-----+
|  0  NVIDIA GeForce ... WDDM  00000000:01:00.0  On  |           N/A         |
| 40%   35C   P8     12W / 160W | 982MiB / 6144MiB     |    11%    Default   |
|                                           N/A                 |
+-----+-----+
```

然后在主页选择 CUDA xx.x 生成 Command 命令，PyTorch CUDA 的版本不能高于显卡支持的 CUDA 版本。

Linux 和 MacOS 同理，在主页选择对应 OS 选项即可。

1.1 PyTorch 介绍

安装完成后验证是否安装成功：在终端键入 `python`，进入 `python` 交互环境。

- CPU 版直接执行 `import torch`，如无报错即安装成功。
- GPU 版执行 `import torch` 后，执行 `torch.cuda.is_available()`，如返回 `True` 说明 GPU 版 PyTorch 安装成功。

```
(pytorch) PS C:\Users\yanghl> python
Python 3.8.5 (default, Sep 3 2020, 21:29:08) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> torch.cuda.is_available()
True
```

PyTorch 安装到此结束，更多内容可参考官方文档：

<https://pytorch.org/docs/stable/index.html>

1.1 PyTorch 介绍——Tensor

- tensor 是 PyTorch 的基本数据类型，在使用 torch 框架进行操作时，对象一般都要求是 tensor 类型。
- 要初始化一个 tensor，通常有以下三种方式：

1. 直接初始化

```
beta = torch.tensor([[1., 2., 3.], [4., 5., 6.]], dtype=torch.float32)
print(beta)

[2] ✓ 0.6s

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

2. 通过原始数据转化

```
alpha = [[1., 2., 3.], [4., 5., 6.]]
beta = torch.tensor(alpha)
print(beta)

[3] ✓ 0.3s

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

3. 通过 numpy 数据转化

```
alpha = np.array([[1., 2., 3.], [4., 5., 6.]])
beta = torch.tensor(alpha)
print(beta)

[5] ✓ 0.5s

... tensor([[1., 2., 3.],
           [4., 5., 6.]], dtype=torch.float64)
```

1.1 PyTorch 介绍——Tensor

- 当初始化时未指定数据类型时，`torch.tensor()` 将会根据数据本身的类型自行判断，如：

```
beta = torch.tensor([[1., 2., 3.], [4., 5., 6.]])
print(beta.dtype)

beta = torch.tensor([[1, 2, 3], [4, 5, 6]])
print(beta.dtype)
```

[6] ✓ 0.3s

... torch.float32
torch.int64

- 也可以通过 `torch.ones()`, `torch.zeros()` 等创建指定大小的全 0 或者全 1 张量：

```
torch.ones((2, 3))
```

[8] ✓ 0.5s

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

```
torch.zeros((2, 3))
```

[9] ✓ 0.5s

... tensor([[0., 0., 0.],
[0., 0., 0.]])

1.1 PyTorch 介绍——Tensor

- 输入神经网络的数据需保证 `tensor(xxx, requires_grad=True)`

e.g.

```
x = torch.tensor([1.0, 2.0])
y1 = x ** 2
y2 = y1 * 2
y3 = y1 + y2

print(y1, y1.requires_grad)
print(y2, y2.requires_grad)
print(y3, y3.requires_grad)


y3.backward(torch.ones(y3.shape))
print(x.grad)
```

[10] 0.2s

```
... tensor([1., 4.]) False
     tensor([2., 8.]) False
     tensor([ 3., 12.]) False

</>
-----
RuntimeError                                Traceback (most recent call last)
<ipython-input-10-ee276a4ce8a7> in <module>
      8 print(y2, y2.requires_grad)
```

`requires_grad` 为 `True` 时会自动记录梯度，这里 $y3 = 2x^2 + x^2$ ，求导为 $4x + 2x = 6x$ ，对应结果为 6 和 12。

若 `requires_grad` 为 `False`， 梯度反向传播时会直接报错。

```
x = torch.tensor([1.0, 2.0], requires_grad=True)
y1 = x ** 2
y2 = y1 * 2
y3 = y1 + y2

print(y1, y1.requires_grad)
print(y2, y2.requires_grad)
print(y3, y3.requires_grad)

y3.backward(torch.ones(y3.shape))
print(x.grad)
```

✓ 0.1s

```
tensor([1., 4.], grad_fn=<PowBackward0>) True
tensor([2., 8.], grad_fn=<MulBackward0>) True
tensor([ 3., 12.], grad_fn=<AddBackward0>) True
tensor([ 6., 12.])
```


1.1 PyTorch 介绍—— torch 常用操作

维度变换

- `torch.view()`或者`torch.reshape()` 维度重置（但总数要一致），若根据已有维度可推算出剩下的维度
- `torch.reshape()`也可以重置维度
- `torch.squeeze(dim)` 若不指定维度，则会将 tensor 中为1的dim压缩，若指定则只会压缩对应的维度（必须为1）

```
temp = torch.rand((4, 4, 6))
print(temp.shape)
print(temp.view(4, 24).shape)
print(temp.view(4, -1).shape)

✓ 0.6s

torch.Size([4, 4, 6])
torch.Size([4, 24])
torch.Size([4, 24])
```

```
temp = torch.rand((4, 1, 6, 1))
print(temp.shape)
print(temp.squeeze().shape)
print(temp.squeeze(1).shape)
print(temp.squeeze(-1).shape)

✓ 0.4s

torch.Size([4, 1, 6, 1])
torch.Size([4, 6])
torch.Size([4, 6, 1])
torch.Size([4, 1, 6])
```

1.1 PyTorch 介绍—— torch 常用操作

维度变换

- `torch.unsqueeze(dim)` 维度扩展

因为神经网络一般默认 batch 输入，所以测试数据时，如果输入为单个数据，需要对数据进行 `unsqueeze` 处理，即将其看成 `batch=1` 的特殊情况

- `torch.cat(List[tensor, tensor], dim)`

向量拼接，需指定维度

```
temp = torch.rand((4, 6))
print(temp.shape)
print(temp.unsqueeze(0).shape)
```

✓ 0.5s

```
torch.Size([4, 6])
torch.Size([1, 4, 6])
```

```
a = torch.tensor([[1, 2], [3, 4]])
b = torch.tensor([[5, 6], [7, 8]])
c = torch.cat([a, b], dim=0)
c
```

✓ 0.5s

```
tensor([[1, 2],
        [3, 4],
        [5, 6],
        [7, 8]])
```

```
a = torch.tensor([[1, 2], [3, 4]])
b = torch.tensor([[5, 6], [7, 8]])
c = torch.cat([a, b], dim=-1)
c
```

✓ 0.4s

```
tensor([[1, 2, 5, 6],
        [3, 4, 7, 8]])
```

1.1 PyTorch 介绍—— torch.nn

□ torch.nn

- 自定义神经网络类的基本框架：继承 `nn.Module` 神经网络基本类，该类实例化后输入数据将自动调用 `forward` 前向计算。

```
from torch import nn

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        ...

    def forward(x):
        ...
        return ...
```

```
net = Net()
out = net(x)
```

1.1 PyTorch 介绍—— torch.nn

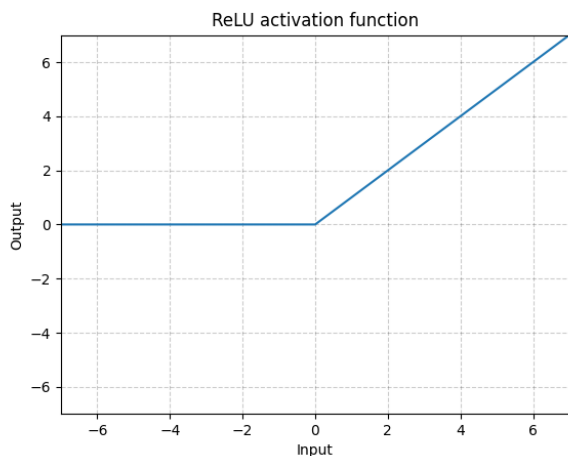
- 全连接层 `nn.Linear(in_dim, out_dim, bias=True)`

```
from torch import nn
m = nn.Linear(20, 30)
input = torch.randn(128, 20)
output = m(input)
print(output.size())
```

✓ 0.7s

`torch.Size([128, 30])`

- 激活函数 `nn.ReLU()`



```
m = nn.ReLU()
input = torch.tensor([-2.4, 1.8])
print(input)
output = m(input)
print(output)
```

✓ 0.3s

`tensor([-2.4000, 1.8000])`
`tensor([0.0000, 1.8000])`

1.1 PyTorch 介绍—— torch.nn

- 卷积神经网络 `nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, bias=True)`
 - NOTE: PyTorch 卷积网络输入默认格式为 (N, C, H, W)**，其中 **N** 为 batch 大小（输入默认batch处理），**C** 为图像通道数（黑白1维，彩色RGB三维），**H**和**W**分别为图像的高度和宽度。
- `Conv2d` 的前两个参数分别为输入和输出的通道数，`kernel_size` 为卷积核大小，`stride`为步长默认为1，`padding` 为填充默认0。
- 一般情况下，计算公式为：

- Input: $(N, C_{in}, H_{in}, W_{in})$ or (C_{in}, H_{in}, W_{in})
- Output: $(N, C_{out}, H_{out}, W_{out})$ or $(C_{out}, H_{out}, W_{out})$, where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

1.1 PyTorch 介绍—— torch.nn

网络训练一般步骤

实例化网络 `net = Net()` 后，计算得到 Loss，并定义网络优化器

`optim = nn.optim.Adam(net.parameters(), lr=lr)`

在更新前，需清除上一步的梯度，即

`optim.zero_grad()`

然后 Loss 反向传播：

`loss.backward()`

最后优化器更新：

`optim.step()`

1.2 CNN 网络训练实例

卷积神经网络（CNN）

什么是卷积神经网络？

➤ 卷积

- 不再是对图像中每一个像素做处理，而是对图片上每一小块像素区域做处理，加强了图片中像素的连续性，从而处理的一个图形而不是单个像素点

➤ 神经网络

- 神经网络是一种计算模型，由大量的神经元以及层与层之间的激活函数组成。

1.2 CNN 网络训练实例

卷积神经网络（CNN）

假设用手写数字识别作为样例：

- 1.读入训练集和测试集中的数字图片信息以及对图片预处理
- 2.用pytorch搭建神经网络（包括卷积和全连接神经网络）
- 3.将一个batch的训练集中的图片输入至神经网络，得到所有数字的预测分类概率（总共10个数字,0123456789）
- 4.根据真实标签和预测标签，利用交叉熵损失函数计算loss值，并进行梯度下降
- 5.根据测试集计算准确率，如果准确率没收敛，跳转回步骤3
- 6.画出loss、测试集准确率的曲线图

参考视频：<https://www.bilibili.com/video/BV1Vx411j7kT?p=19>

参考代码：

https://github.com/MorvanZhou/PyTorch-Tutorial/blob/master/tutorial-contents/401_CNN.py

1.2 CNN 网络训练实例

卷积神经网络（CNN）

步骤2：用pytorch搭建神经网络（包括卷积和全连接神经网络）

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(          # input shape (1, 28, 28)
            nn.Conv2d(                        # input height
                in_channels=1,                # n_filters
                out_channels=16,              # filter size
                kernel_size=5,                # filter movement/step
                stride=1,                     # if want same width and length of this image after con2d, padding=(kernel_size-1)/2 if stride=1
                padding=2,                    # output shape (16, 28, 28)
            ),
            nn.ReLU(),                        # activation
            nn.MaxPool2d(kernel_size=2),      # choose max value in 2x2 area, output shape (16, 14, 14)
        )
        self.conv2 = nn.Sequential(          # input shape (1, 28, 28)
            nn.Conv2d(16, 32, 5, 1, 2),      # output shape (32, 14, 14)
            nn.ReLU(),                       # activation
            nn.MaxPool2d(2),                 # output shape (32, 7, 7)
        )
        self.out = nn.Linear(32 * 7 * 7, 10) # fully connected layer, output 10 classes

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = x.view(x.size(0), -1)           # flatten the output of conv2 to (batch_size, 32 * 7 * 7)
        output = self.out(x)
        return output, x                    # return x for visualization
```

1.2 CNN 网络训练实例

卷积神经网络（CNN）

步骤3：将一个batch的训练集中的图片输入至神经网络，得到所有数字的预测分类概率

步骤4：根据真实标签和预测标签，利用交叉熵损失函数计算loss值，并进行梯度下降

```
for epoch in range(EPOCH):
    for step, (x, y) in enumerate(train_loader):  # gives batch data, normalize x when iterate train_loader
        b_x = Variable(x)  # batch x
        b_y = Variable(y)  # batch y

        output = cnn(b_x)[0]  # cnn output
        loss = loss_func(output, b_y)  # cross entropy loss
        optimizer.zero_grad()  # clear gradients for this training step
        loss.backward()  # backpropagation, compute gradients
        optimizer.step()  # apply gradients

    if step % 100 == 0:
        test_output, last_layer = cnn(test_x)
        pred_y = torch.max(test_output, 1)[1].data.squeeze()
        accuracy = (pred_y == test_y).sum().item() / float(test_y.size(0))
        print('Epoch: ', epoch, '| train loss: %.4f' % loss.data[0], '| test accuracy: %.2f' % accuracy)
```

2. 实验任务

□ 中药图片分类任务

- 利用pytorch框架搭建神经网络实现中药图片分类，其中中药图片数据分为训练集train和测试集test，训练集仅用于网络训练阶段，测试集仅用于模型的性能测试阶段。训练集和测试集均包含五种不同类型的中药图片：baihe、dangshen、gouqi、huaihua、jinyinhua。
- 具体见超算习堂“实验作业六”附件中给出的训练集和测试集。
- 要求：
 - 搭建合适的网络框架，利用训练集完成网络训练，统计网络模型的训练准确率和测试准确率，画出模型的训练过程的loss曲线、准确率曲线。
 - 可以使用`pytorch`框架、`numpy`库、`matplotlib`库以及python标准库

3. 作业提交说明

- ❑ 提交到课程网站（超算习堂）中对应的“实验作业六”，并注意网站上公布的截止日期
- ❑ 提交格式：提交一个命名为“学号_姓名.zip”的压缩包，压缩文件下包含两部分：code文件夹和实验报告pdf文件
- ❑ 实验报告是pdf格式，命名为：学号_姓名.pdf
- ❑ code文件夹：存放实验代码，一般有多个代码文件的话需要有readme
- ❑ “学号_姓名” 样例：20*****_wangxiaoming
- ❑ 如果需要更新提交的版本，则在后面加_v1，_v2。如第一版是”学号_姓名.zip”，第二版是”学号_姓名_v1.zip”，依此类推
- ❑ 截止日期：**2024年5月27日晚24点**