Artificial Intelligence
人工智能实验

深度学习

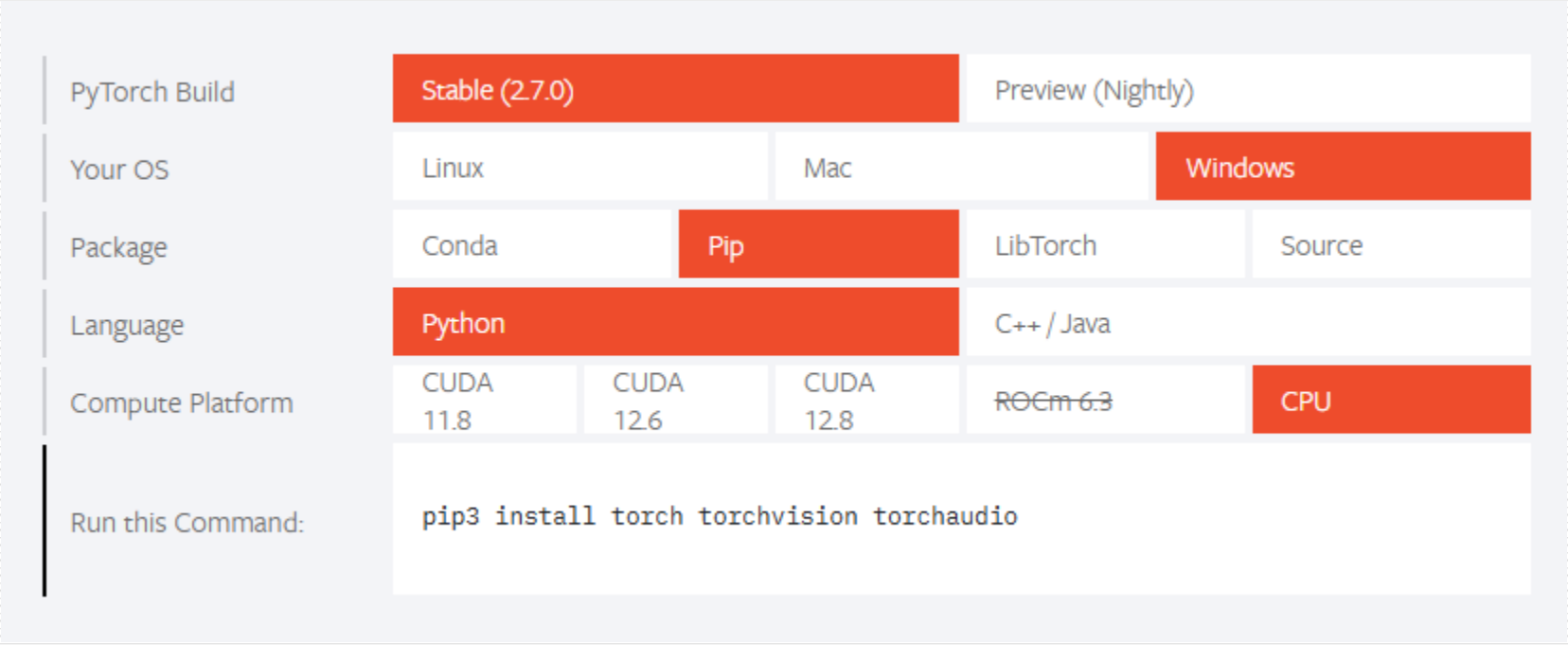
中山大学计算机学院
2025年春季

目录

1.1 PyTorch 介绍

PyTorch 安装

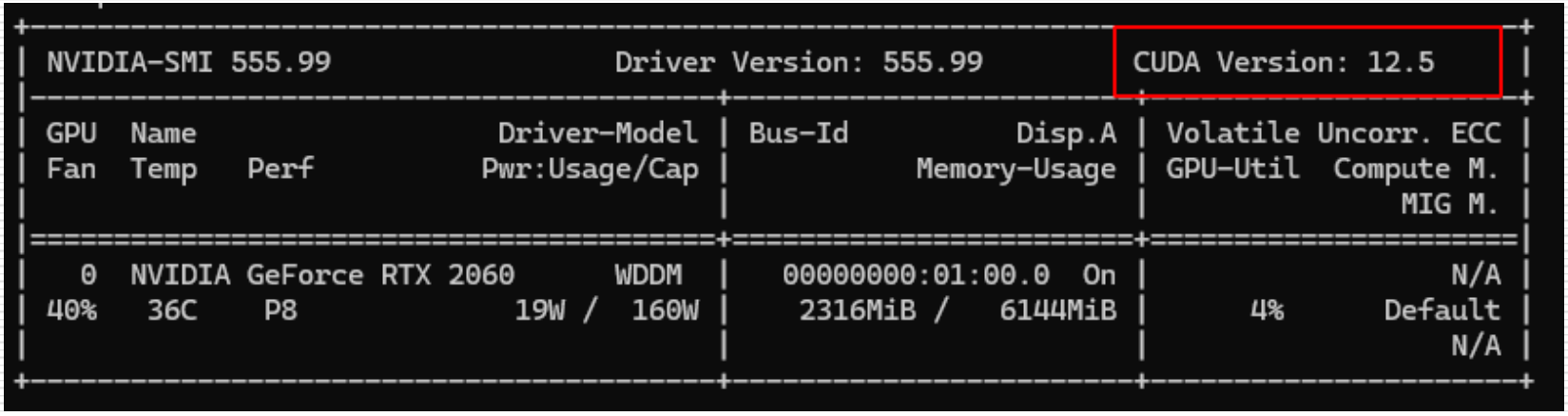
- 官网：https://pytorch.org
- CPU 版安装（无 Nvidia 显卡）：直接在官网主页选择配置，然后复制生成的 Command 粘贴到终端运行（注意需提前激活 Python 虚拟环境）。
- 建议：安装1.7.0及以上版本。



1.1 PyTorch 介绍

PyTorch 安装

- 官网：https://pytorch.org
- GPU 版安装：先在终端使用 nvidia-smi 查看当前显卡支持的 CUDA 版本



- 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。

1.1 PyTorch 介绍

PyTorch 安装

- 官网：https://pytorch.org
- 例如，版本 pytorch==1.12.0 的安装命令

Linux and Windows

```
# ROCM 5.1.1 (Linux only)
pip install torch==1.12.0+rocm5.1.1 torchvision==0.13.0+rocm5.1.1 torchaudio==0.12.0 --extra-index-url  http
# CUDA 11.6
pip install torch==1.12.0+cu116 torchvision==0.13.0+cu116 torchaudio==0.12.0 --extra-index-url https://down
# CUDA 11.3
pip install torch==1.12.0+cu113 torchvision==0.13.0+cu113 torchaudio==0.12.0 --extra-index-url https://down
# CUDA 10.2
pip install torch==1.12.0+cu102 torchvision==0.13.0+cu102 torchaudio==0.12.0 --extra-index-url https://down
# CPU only
pip install torch==1.12.0+cpu torchvision==0.13.0+cpu torchaudio==0.12.0 --extra-index-url https://download.
```

1.1 PyTorch 介绍

PyTorch 安装

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

```
(python38) C:\Users\Administrator>python
Python 3.8.13 (default, Mar 28 2022, 06:59:08) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch as th
>>> th.cuda.is_available()
True
>>>
```

- CPU 版直接执行 import torch，如无报错即安装成功。
- 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

- 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

```
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
```

- 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：

```
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)

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

```
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>
```

- 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。

```
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

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，可使用 -1 替代
- torch.reshape()也可以重置维度

```
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])
```

- 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, 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

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

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

✓ 0.5s

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

- torch.cat(List[tensor, tensor], dim) 向量拼接，需指定维度

```
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

- 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

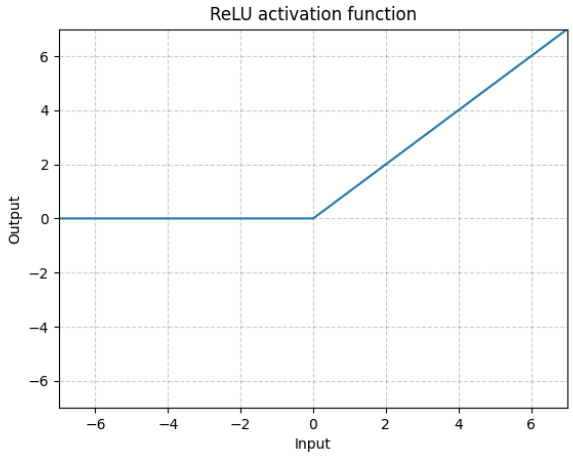
- 全连接层
- nn.Linear(in_dim, out_dim, bias=True)

```
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)
- 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

- 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。

1.1 PyTorch 介绍

网络训练一般步骤

- 实例化网络 net = Net() 后，定义网络优化器
  - **optim = nn.optim.Adam(net.parameters(), lr=lr)**
- 计算得到 Loss，
  - **loss=MSE(a,b)**
- 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

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

手写数字识别作为样例

- 步骤2：用pytorch搭建神经网络（包括卷积和全连接神经网络）

```python
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.conv1 = nn.Sequential(         # input shape (1, 28, 28)
            nn.Conv2d(
                in_channels=1,              # input height
                out_channels=16,            # n_filters
                kernel_size=5,              # filter size
                stride=1,                   # filter movement/step
                padding=2,                  # if want same width and length of this image after con2d, padding=(kernel_size-1)/2 if stride=1
            ),                              # 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

手写数字识别作为样例

- 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，具体见给出的数据集和测试集。
- 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。
  - 本次作业可以使用pytorch`库、 numpy库、matplotlib库以及python标准库。
  - 测试集不可用于模型训练。
  - 不能使用开源的预训练模型进行训练。

3. 作业提交说明

- 压缩包命名为：“学号_姓名_作业编号”，例：20250512_张三_实验5。
- 每次作业文件下包含两部分：code文件夹和实验报告PDF文件。
  - code文件夹：存放实验代码；
  - PDF文件格式参考发的模板。
- 如果需要更新提交的版本，则在后面加_v2，_v3。如第一版是“学号_姓名_作业编号.zip”，第二版是“学号_姓名_作业编号_v2.zip”，依此类推。
- 截至日期：**2024年5月12日晚24点**。
- 提交邮箱：zhangyc8@mail2.sysu.edu.cn。