系列文章

OpenVINO™ 工具套件

intel.

PyTorch + OpenVINO 开发实战系列教程

第一篇

目录

概述

大家好，本章是主要介绍一下深度学习框架 Pytorch 的的历史与发展，主要模块构成与基础操作代码演示。重点介绍 Pytorch 的各个组件、编程方式、环境搭建、基础操作代码演示。本章对有 Pytorch 开发经验的读者来说可以直接跳过；对初次接触 Pytorch 的读者来说，通过本章学习认识 Pytorch 框架，搭建好 Pytorch 的开发环境，通过一系列的基础代码练习与演示建立起对深度学习与 Pytorch 框架的感性认知。

本书内容以 Python 完成全部代码构建与程序演示。本章的主要目标是帮助初次接触 Python 与 Pytorch 的读者搭建好开发环境，认识与理解 Pytorch 框架中常见的基础操作函数、学会使用它们完成一些基础的数据处理与流程处理，为后续内容学习打下良好基础。

好了，下面就让我们来一起开启这段 Pytorch 框架的深度学习破冰之旅。

1. Pytorch 介绍与基础知识

1.1 Pytorch 介绍

Pytorch 是开放源代码的机器学习框架，目的是加速从研究原型到产品开发的过程。其 SDK 主要基于 Python 语言，而 Python 语言作为流行的人工智能开发语言一直很受研究者与开发者的欢迎。其模型训练支持 CPU 与 GPU、支持分布式训练、云部署、针对深度学习特定领域有不同的丰富的扩展库。

1.1.1 Pytorch 历史

Pytorch 在 2016 年由 facebook 发布的开源机器学习（深度学习）框架，Pytorch 最初的来源历史可以追溯到另外两个机器学习框架，第一个是 torch 框架，第二个是 Chainer，实现了 Eager 模式与自动微分，Pytoch 集成了这两个框架的优点， 把 Python 语言作为框架的首选编程语言，所以它的名字是在 torch 的前面加上 Py 之后的 Pytorch。由于 Pytorch 吸取了之前一些深度学习框架优点，开发难度大大降低、很容易构建各种深度学习模型并实现分布式的训练，因此一发布就引发学术界的追捧热潮，成为深度学习研究者与爱好者的首选开发工具。在 pytorch 发布之后两年的 2018 年 facebook 又把 caffe2 项目整合到 pytorch 框架中，这样 pytorch 就进一步整合原来 caffe 开发者生态社区，因为其开发效率高、特别容易构建各种复杂的深度学习模型网络，因此很快得到大量人工智能开发者的认可与追捧，也成为工业界最受欢迎的深度学习框架之一。

Pytorch 发展至今，其版本跟功能几经迭代，针对不同的场景任务分裂出不同的分支扩展库，比如针对自然语言处理（NLP）的 torchtext、针对计算机视觉的 torchvision、针对语音处理的 torchaudio，这些库支持快速模型训练与演示应用，可以帮助开发者快速搭建原型演示。此外在移动端支持、模型部署的压缩、量化、服务器端云化部署、推理端 SDK 支持等方面 Pytorch 也在不断的演化改进。

在操作系统与 SDK 支持方面，Pytorch 从最初的单纯支持 Python 语言到如今支持 Python/C++/Java 主流编程语言，目前已经支持 Linux、Windows、MacOS 等主流的操作系统、同时全面支持 Android 与 iOS 移动端部署。

在版本发布管理方面，Pytorch 分为三种不同的版本分别是稳定版本 (Stable Release)、Beta 版本、原型版本（Prototype）。其中稳定版本长期支持维护没有明显的性能问题与缺陷，理论上支持向后兼容的版本；Beta 版本是基于用户反馈的改动版本，可能有 API/SDK 函数改动，性能有进一步需要提升的空间；原型版本是新功能还不可以，需要开发不能通过 pip 方式直接安装。

1.1.2 Pytorch 的模块与功能

Pytorch 当前支持绝大数的深度学习常见的算子操作，基于相关的功能模块可以快速整合数据、构建与设计模型、实现模型训练、导出与部署等操作。这些功能的相关模块主要有如下:

1）torch.nn 包，里面主要包含构建卷积神经网络的各种算子操作，主要包括卷积操作（Conv2d、Conv1d、Conv3d）激活函数、序贯模型 (Sequential)、功能函数 (functional)、损失功能、支持自定义的模型类（Module）等。通过它们就可以实现大多数的模型结构搭建与生成。

2）torch.utils 包，里面主要包括训练模型的输入数据处理类、pytorch 自带的模型库、模型训练时候可视化支持组件、检查点与性能相关的组件功能。重要的类有数据集类（Dataset），数据加载类 (DataLoader)、自定义编程的可视化支持组件 tensorboard 相关类。

3）torch 开头的一些包与功能，主要包括支持模型导出功能的 torch.onnx 模块、优化器 torch.optim 模块、支持 GPU 训练 torch.cuda 模块，这些都是会经常用的。

4）此外本书当中还会重点关注的 torchvison 库中的一些常见模型库与功能函数，主要包括对象检测模块与模型库、图象数据增强与预处理模块等。

以上并不是 pytorch 框架中全部模块与功能说明，作者这里只列出了跟本书内容关联密切必须掌握的一些模块功能，希望读者可以更好的针对性学习，掌握这些知识。

1.1.3 Pytorch 框架现状与趋势

Pytorch 是深度学习框架的后起之秀，它参考了市场上早期框架包括 torch、caffe、tensorflow 的经验教训，从一开始设

计就特别注重开发者体验与生产效率提升，一经发布就引发追捧热潮，可以说“出道即巅峰”。Pytorch 虽然来自脸书实验室，但是它也吸引外部公司包括特斯拉、优步、亚马逊、微软、阿里等积极支持，其平缓的学习曲线，简洁方便的函数与模型构建在短时间内吸引了大量学术研究者与工业界开发者的追捧。当前无论是在学术界还是工业界 Pytorch 已经是主流深度学习框架之一，而且大有后来居上之势，因此随着人工智能赋能各行各业，Pytorch 框架必然会更加得到开发者的青睐，成为人工智能 (AI) 开发者必备技能之一。同时 Pytorch 也会在部署跟推理方面会更加完善与方便，加强支持移动端，嵌入式端等应用场景，相信掌握 Pytorch 框架的开发技术人才也会得到丰厚回报。

1.2 环境搭建

Pytorch 的开发环境搭建十分的简洁，它的依赖只有 Python 语言 SDK，只要有了 Python 语言包支持，无论是在 windows 平台、ubuntu 平台还是 Mac 平台都靠一条命令行就可以完成安装。首先是安装 Python 语言包支持，当前 Pytorch 支持的 Python 语言版本与系统对应列表如下：

表 -1（参考 Pytorch 官网与 Github）

系统	Python3.6	Python3.7	Python3.8
Linux CPU/GPU	支持	支持	支持
Windows CPU/GPU	支持	支持	支持
Linux (aarch64) CPU	支持	支持	支持
Mac (CPU)	支持	支持	支持

当前最新稳定版本是 Pytorch 1.9.0、长期支持版本是 Pytorch 1.8.2(LTS), 此外 Python 语言支持版本 3.6 表示支持 3.6.x 版本，其中 x 表示 3.6 版本下的各个小版本，依此类推 3.7、3.8 同样如此。本书代码演示以 Python3.6.5 版本作为 Python 支持语言包。它在 Windows 系统下的安装过程非常简单，只需如下几步：

1. 下载 Python3.6.5 安装包，地址为：

https://www.python.org/ftp/python/3.6.5/python-3.6.5-amd64.exe

2. 下载之后，双击 exe 文件安装，显示的界面如下：

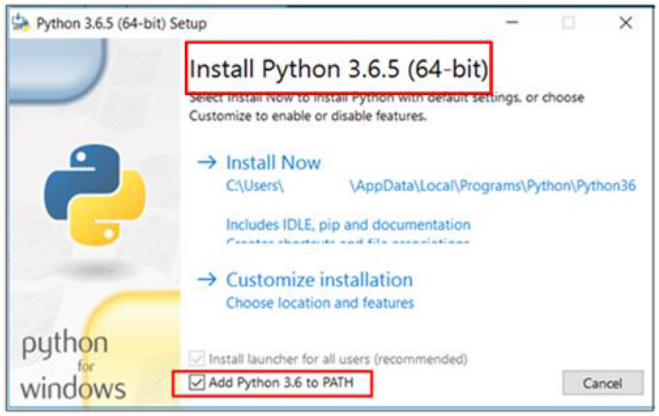

图 1-1（Python3.6.5 安装界面）

注意：图 1-1 中的矩形框，必须手动选择上“add Python3.6 to PATH”之后再点击【Install Now】默认安装完成即可。

3. 安装好 Python 语言包支持以后可以通过命令行来验证测试安装是否成功，首先通过 cmd 打开 Window 命令行窗口，然后输入 Python，显示如下：

图 1-2（验证 Python 命令行模式）

如果显示图 1-2 所示的信息表示已经安装成功 Python 语言包支持；如果输入 Python 之后显示信息为**“'python' 不是内部或外部命令，也不是可运行的程序”**则说明第二步中没有勾选上“add Python3.6 to PATH”，此时请手动把 python.exe 所在路径添加到 Windows 系统的环境变量中去之后再次执行即可。

4. 安装好 Python 语言包支持之后，只要运行下面的命令行即可完成 Pytorch 框架的安装，GPU 支持版本的命令行如下（需要 GPU 显卡支持）：

pip install torch==1.9.0+cu102 torchvision==0.10.0+cu102 torchaudio===0.9.0 -f https://download.pytorch.org/whl/torch_stable.html

CPU 支持版本的命令行如下（没有 GPU 显示支持）：
pip install torch torchvision torchaudio

5. 在执行第三步的基础上，在命令行中输入下面两行代码，执行结果如下：

```
>>> import torch
>>> torch._ _version_ _
'1.9.0+cu102'
```

其中第一行表示导入 pytorch 的包支持，第二行表示版本查询，第三行是执行结果（GPU 版本）。

现在很多开发者喜欢使用 Ubuntu 开发系统，在 Ubuntu 系统下如下正确安装与配置 Pytorch，第一步同样是安装 python 语言依赖包 Python3.6，主要是执行一系列的安装命令行，具体步骤如下：

1. 导入第三方软件仓库
sudo add-apt-repository ppa:jonathonf/python-3.6

2. 更新与安装 python3.6
sudo apt-get update
sudo apt-get install python3.6

3. 删除默认 python 版本设置
zhigang@ubuntu:/usr/bin$ sudo rm python

4. 把安装好的 3.6 设置为默认版本
zhigang@ubuntu:/usr/bin$ sudo ln -s python3.6 /usr/bin/python

5. 检查与验证
zhigang@ubuntu:~$ python -V
Python 3.6.5

成功完成上述五个步骤的命令行执行就完成了 Python 语言包依赖安装，然后安装 Pytorch 框架，CPU 版本执行命令行如下：
pip3 install torch==1.9.0+cpu torchvision==0.10.0+cpu torchaudio==0.9.0 -f https://download.pytorch.org/whl/torch_stable.html

GPU 版本执行命令行如下：
pip3 install torch torchvision torchaudio

然后执行与 Windows 下相同的命令行完成 pytorch 安装校验测试。这样我们就完成了 Pytorch 的环境搭建，这里有个很特别的地方需要注意，就是 Pytorch 的 GPU 版本需要 CUDA 驱动支持与 CUDA 库的安装配置支持。关于这块的安装强烈建议参照英伟达官方网站的安装指导与开发者手册。

1.3 Pytorch 基础术语与概念

很多人开始学习深度学习框架面临的第一个问题就是专业术语理解跟基本的编程概念与传统面向对象编程不一样，这个是初学者面临的第一个学习障碍。在主流的面向对象编程语言中，结构化代码最常见的关键字是 if、else、while、for 等关键字，而在深度学习框架中编程模式主要是基于计算图、张量数据、自动微分、优化器等组件构成。面向对象编程运行的结果是交互式可视化的，而深度学习通过训练模型生成模型文件，然后再使用模型预测，本质数据流图的方式工作。所以学习深度学习首先必须厘清深度学习编程中计算图、张量数据、自动微分、优化器这些基本术语概念，下面分别解释如下：

- 张量

张量是深度学习编程框架中需要理解最重要的一个概念，张量的本质是数据，在深度学习框架中一切的数据都可以看成张量。深度学习中的计算图是以张量数据为输入，通过算子运算，实现对整个计算图参数的评估优化。但是到底什么是张量？可以看下面这张图：

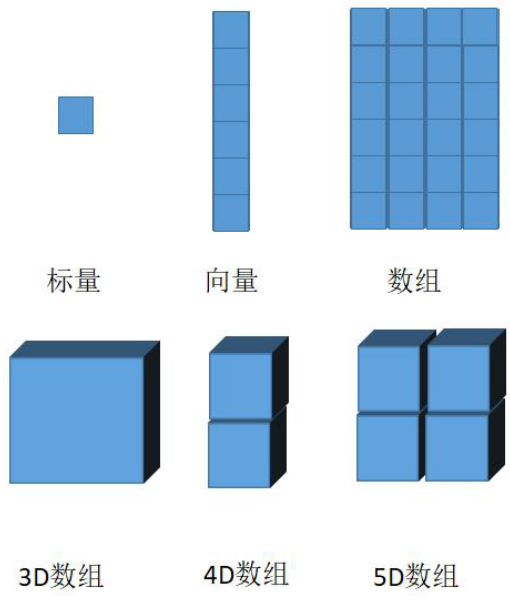

图 1-3（张量表示）

上图 1-3 中标量、向量、数组、3D、4D、5D 数据矩阵在深度学习框架中都被称为张量。可见在深度学习框架中所有的数据都是张量形式存在，张量是深度学习数据组织与存在一种数据类型。

- 算子 / 操作数

深度学习主要是针对张量的数据操作、这些数据操作从简单到复杂、多数都是以矩阵计算的形式存在，最常见的矩阵操作就是加减乘除、此外卷积、池化、激活、也是模型构建中非常有用的算子 / 操作数。Pytorch 支持自定义算子操作，可以通过自定义算子实现复杂的网络结构，构建一些特殊的网络模型。张量跟算子 / 操作数一起构成了计算图，它们是也是计算图的基本组成要素。

- 计算图

深度学习是基于计算图完成模型构建，实现数据在各个计算图节点之间流动，最终输出，因此计算图又被称为数据流图。根据构建计算图的方式不同还可以分为静态图与动态图，Pytorch 默认是基于动态图的方式构建计算图，动态图采用类似 python 语法，可以随时运行，灵活修改调整；而静态图则是效率优先，但是在图构建完成之前无法直接运行。可以看出动态图更加趋向于开发者平时接触的面向对象的编程方式，也更容易被开发者理解与接受。下图是一个简单的计算图示例：

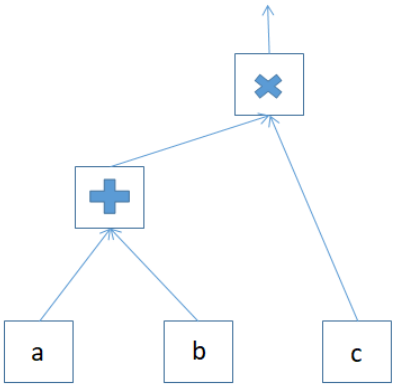

图 1-4（计算图示意）

图 1-4 中最底层三个节点表示计算图的输入张量数据节点(a、b、c) 、剩下节点表示操作、带箭头的线段表示数据的流向。

- 自动微分

使用 Pytorch 构建神经网络（计算图）模型之后，一般都是通过反向传播进行训练，使用反向传播算法对神经网络中每个参数根据损失函数功能根据梯度进行参数值的调整。为了计算这些梯度完成参数调整，深度学习框架中都会自带一个叫做自动微分的内置模块，来自动计算神经网络模型训练时候的各个参数梯度值并完成参数值更新，这种技术就是深度学习框架中的自动微分。

1.4 Pytorch 基础操作

前面我们已经安装并验证好了 Pytorch 框架，解释了深度学习框架中一些常见术语与基本概念。本节重点介绍 Pytorch 中一些基本的数据定义与类型转换、算子操作、通过它们帮助读者进一步了解 Pytorch 开发基础知识，为后续章节学习打下良好基础。在正式开始这些基础操作之前，我们首先需要有一个合适的集成开发环境 (IDE)，本书的源代码是基于 Python 实现，演示的集成开发环境（IDE）是 PyCharm。

1.4.1 PyCharm 的安装与配置

首先是从 Pycharm 官方网站上下载 Pycharm，版本有专业版与社区版之分，社区版免费使用而专业版则需要付费使用。Pycharm 官方网站如下：

https://www.jetbrains.com/pycharm/

点击就可以下载专业版试用或者社区免费版，默认安装之后就可以通过桌面图标双击打开如下：

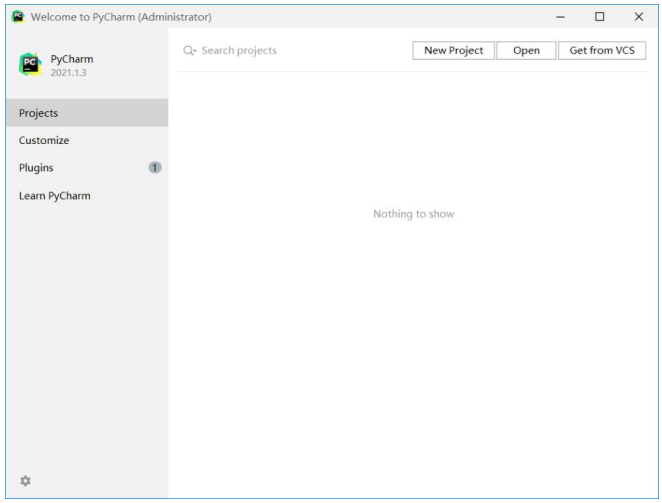

图 1-5（Pycharm 导航页面）

点击【New Project】，输入项目名称，显示如下：

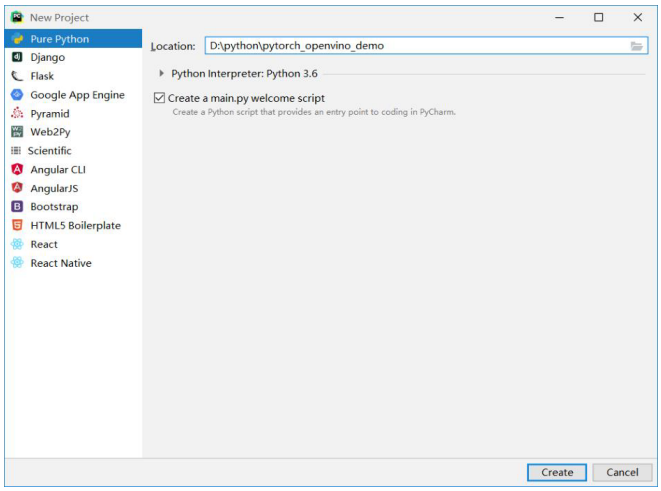

图 1-6（创建新项目）

点击【Create】按钮完成项目创建，选择文件 (File)-> 设置 (Setting) 选项：

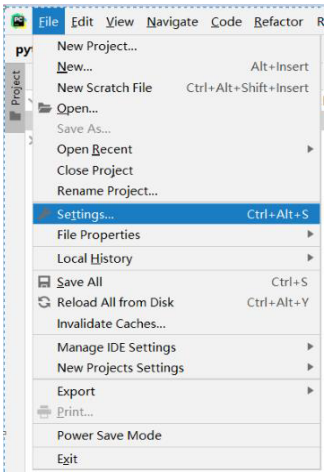

图 1-7（设置选项）

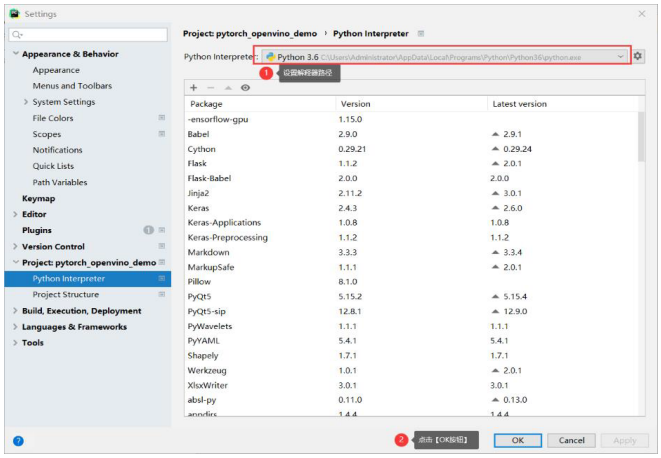

图 1-8（设置系统 Python 解释器）

完成之后，在项目中创建一个空的 python 文件命名为 main.py，然后直接输入下面两行测试代码：

```
import torch
print(torch.__version__)
```

执行测试（作者笔记本）：

1.9.0+cu102

这样我们就完成了 PyCharm IDE 开发环境配置与项目创建。

1.4.2 张量定义与声明

张量在 Pytorch 深度学习框架中表示的数据，有几种不同的方式来创建与声明张量数据，一种是通过常量数值来直接声明为 tensor 数据，代码如下：

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

运行结果

```
tensor([[2., 3.],
        [4., 5.]]) torch.float32
```

其中 torch.Tensor 是 torch.FloatTensor 的别名，所以默认的数据类型是 flaot32，这点从 a.dtype 的打印结果上也得了印证。此外 torch.Tensor 函数还支持从 Numpy 数组直接转换为张量数据，这种定义声明张量数据的代码如下：

```
b = torch.tensor(np.array([[1,2],[3,4],[5,6],[7, 8]]))
print(b)
```

运行结果：

```
tensor([[1, 2],
        [3, 4],
        [5, 6],
        [7, 8]], dtype=torch.int32)
```

根据数据类型的自动识别，转换为 torch.int32 的数据类型。除了直接声明常量数组的方式，Pytorch 框架还支持类似 Matlab 方式的数组初始化方式，可以定义数组的维度，然后初始化为零，相关的演示代码如下：

```
c = torch.zeros([2, 4], dtype=torch.float32)
print(c)
```

运行结果：

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

初始化了一个两行四列值全部为零的数组。torch.zeros 表示初始化全部为零，torch.ones 表示初始化全部为 1，torch.ones 的代码演示如下：

```
d = torch.ones([2, 4], dtype=torch.float32)
print(d)
```

运行结果：

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

上面都是创建常量数组的方式。在实际的开发中，经常需要随机初始化一些张量变量，创建张量并随机初始化的方式主要通过 torch.rand 函数实现。用该函数创建张量的代码演示如下：

```
v1 = torch.rand((2, 3))
print("v1 = ", v1)

torch.initial_seed()
v2 = torch.rand((2, 3))
print("v2 = ", v2)

v3 = torch.randint(0, 255, (4, 4))
print("v3 = ", v3)
```

运行结果：

```
v1 =  tensor([[0.5257, 0.4236, 0.0409],
      [0.3271, 0.6173, 0.8080]])
v2 =  tensor([[0.9671, 0.6011, 0.3136],
      [0.7428, 0.9261, 0.6602]])
v3 =  tensor([[181, 170,  49,  82],
      [209,  25,   0, 210],
      [ 65, 220,  93,  11],
      [133, 102,  64, 230]])
```

其中 v1 是直接输出、v2 首先随机初始化种子之后再输出、v3 是函数 torch.randint 创建的随机数组，它的前面两个值 0 跟 255 表示整数的取值范围为 0~255 之间，最后一个 (4,4) 表示创建 4x4 大小的数组。

1.4.3 张量操作

通过前面一小节，我们已经学会了在 Pytorch 中如何创建张量，本节将介绍张量常见的算子操作，通过这些操作进一步加深对 Pytorch 的理解。

- 计算图操作

还记得图 1-4 的计算图吗？这里我们就通过代码构建这样一个计算图，完成一系列基于张量的算子操作，演示代码如下：

```
a = torch.tensor([[2., 3.], [4., 5.]])
b = torch.tensor([[10, 20], [30, 40]])
c = torch.tensor([[0.1], [0.2]])
x = a + b
y = torch.matmul(x, c)
print("y: ", y)
```

运行结果如下：

```
y:  tensor([[ 5.8000],
            [12.4000]])
```

上面得代码中 x 是 a 加 b 的结果，y 是 a 加 b 之和与 c 的矩阵乘法的最终输出结果。

- 数据类型转换

在实际的开发过程中，我们经常需要在不同类型的数据张量中切换，因此数据类型转换函数也是必修的，代码演示如下：

```
m = torch.tensor([1.,2.,3.,4.,5.,6], dtype=torch.float32)
print(m, m.dtype)
print(m.int())
print(m.long())
print(m.double())
```

运行结果如下：

```
tensor([1., 2., 3., 4., 5., 6.]) torch.float32
tensor([1, 2, 3, 4, 5, 6], dtype=torch.int32)
tensor([1, 2, 3, 4, 5, 6])
tensor([1., 2., 3., 4., 5., 6.], dtype=torch.float64)
```

可见，在 PyTorch 中实现数据类型转换非常的便捷， m.int() 表示转换伟 32 位整型、m.double() 表示转换位 64 位浮点数类型、m.long() 表示 64 位整型。

- 维度转换

在 Pytorch 开发中另外一个常见的基础操作就是图象维度转换与升降维度操作，Pytorch 中实现对张量数据的维度转换与升降的代码演示如下：

```
a = torch.arange(12.)
a = torch.reshape(a, (3, 4))
print("a: ", a)
a = torch.reshape(a, (-1, 6))
print("a: ", a)
a = torch.reshape(a, (-1, ))
print("a: ", a)
a = torch.reshape(a, (1, 1, 3, 4))
print("a: ", a)
```

运行结果如下：

```
a:  tensor([[ 0.,  1.,  2.,  3.],
            [ 4.,  5.,  6.,  7.],
            [ 8.,  9., 10., 11.]])
a:  tensor([[ 0.,  1.,  2.,  3.,  4.,  5.],
            [ 6.,  7.,  8.,  9., 10., 11.]])
a:  tensor([ 0.,  1.,  2.,  3.,  4.,  5.,  6.,  7.,  8.,  9., 10., 11.])
a:  tensor([[[[ 0.,  1.,  2.,  3.],
            [ 4.,  5.,  6.,  7.],
            [ 8.,  9., 10., 11.]]]])
```

其中 a 是声明的张量数据，torch.reshape(a, (3, 4)) 意思转换为 3 行 4 列的二维数组、torch.reshape(a, (-1, 6)) 表示转为每行 6 列的二维数组，其中 -1 表示从列数推理得到行数、torch.reshape(a, (-1,)) 表示直接转换为一行、torch.reshape(a, (1, 1, 3, 4)) 表示转为 1x1x3x4 的四维张量。除了 torch.reshape 函数之外，还有另外一个基于 tensor 的维度转换方法 tensor.view(), 它的用法代码演示如下：

```
x = torch.randn(4, 4)
print(x.size())
x = x.view(-1, 8)
print(x.size())
x = x.view(1, 1, 4, 4)
print(x.size())
```

运行结果如下：

```
torch.Size([4, 4])
torch.Size([2, 8])
torch.Size([1, 1, 4, 4])
```

其中 torch.randn(4, 4) 是创建一个 4x4 的随机张量；x.view(-1, 8) 表示转换为每行八列的，-1 表示自动计算行数；x.view(1, 1, 4, 4) 表示转换为 1x1x4x4 的四维张量。其中 torch.size 表示输出数组维度大小。

- 其它属性操作

通道交换与寻找最大值是 Pytorch 中处理张量数据常用操作之一，这两个相关函数名称分别是 torch.transpose 与 torch.argmax，支持张量直接操作。代码演示如下：

```
x = torch.randn(5, 5, 3)
print(x.size())
x = x.transpose(2, 0)
print(x.size())
x = torch.tensor([2., 3., 4.,12., 3., 5., 8., 1.])
print(torch.argmax(x))
x = x.view(-1, 4)
print(x.argmax(1))
```

运行结果如下：

```
torch.Size([5, 5, 3])
torch.Size([3, 5, 5])
tensor(3)
tensor([3, 2])
```

运行结果的第一行对应是声明的张量 x 的维度信息、第二行是调用 transpose 方法完成通道交换之后 x 输出的维度信息；第三行是针对 x 调用 argmax 得到最大值对应索引（12 对应索引值为 3）、第四行是进行维度变换之后针对二维数据（2x4）的第二个维度调用 argmax 得到的输出，分别是 12 与 8 对应的索引值 [3,2]。

- CPU 与 GPU 运算支持

Pytorch 支持 CPU 与 GPU 计算，默认创建的 tensor 是 CPU 版本的，要想使用 GPU 版本，首先需要检测 GPU 支持，然后转换为 GPU 数据，或者直接创建为 GPU 版本数据，代码演示如下：

```
gpu = torch.cuda.is_available()
for i in range(torch.cuda.device_count()):
```

```
        print(torch.cuda.get_device_name(i))
if gpu:
        print(x.cuda())
        y = torch.tensor([1, 2, 3, 4], device="cuda:0")
        print("y: ", y)
```

以上代码查询 CUDA 支持，如果支持打印 GPU 名称并把变量 x 变成 GPU 支持数据，并打印输出。运行结果如下：

```
GeForce GTX 1050 Ti
tensor([[ 2.,  3.,  4., 12.],
        [ 3.,  5.,  8.,  1.]], device='cuda:0')
y:  tensor([1, 2, 3, 4], device='cuda:0')
```

这里 x 默认是 CPU 类型数据，y 是直接创建的 GPU 类型数据。

以上都是一些最基础跟使用频率较高的 Pytorch 基础操作，了解并掌握这些函数有助于进一步学习本书后续章节知识，更多关于 Pytorch 基础操作的函数知识与参数说明，读者可以直接参见官方的开发文档。

1.5 线性回归预测

上一小节介绍了 Pytorch 框架各种基础操作，本节我们学习一个堪称是深度学习版本的 Hello World 程序，帮助读者理解模型训练与参数优化等基本概念，开始我们学习 Pytorch 框架编程的愉快旅程。

1.5.1 线性回归过程

很坦诚的说，有很多资料把线性回归表述的很复杂、一堆公式推导让初学者望而生畏，无法准确快速理解线性回归，这里作者将通过一个码农的独特视角来解释线性回归概念。线性回归的本质就是根据给出二维数据集来拟合生成一条直线，简单的图示如下：

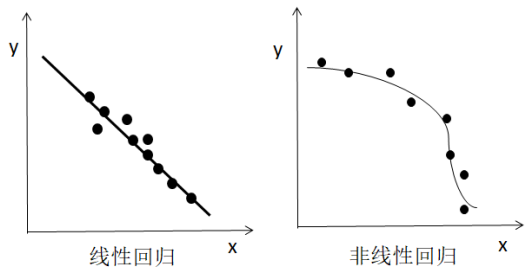

图 1-9（线性回归与非线性回归）

图 1-9 左侧是图中得圆点表示 xy 二维坐标点数据集，直线是根据线性回归算法生成的。右侧则是一个根据坐标点数据集生成一个非线性回归的例子。现在我们已经可以很直观的了解什么线性回归了，脑子里面可能会有个大大的疑问（？），线性回归是怎么找到这条直线的？答案就是通过 Pytorch 构建一个简单的计算图来不断学习，最终得到一个足够逼近真实直线参数方程，这个过程也可以被称为线性回归的学习 / 训练过程。最终根据得到的参数就可以绘制回归直线。那这个计算图到底是怎么样的？答案就是很简单的数学知识，最常见的直线方程如下：

$$y = kx + b$$ （公式 1-1）

假设我们有二维的坐标点数据集：

x: 1,2,0.5,2.5,2.6,3.1

y: 3.7,4.6,1.65,5.68,5.98,6.95

我们通过随机赋值初始 k、b 两个参数，根据公式 1-1，x 会生成一个对应输出 $\hat{y}$，它跟真实值 y 之间的差值我们称为损失，最常见的为均值平方损失（MSE），表示如下：

$$MSE = \frac{1}{n}\sum(y-\hat{y})^2$$ （公式 1-2）

然后我们可以通过下面的公式来更新 k、b 两个参数：

更新参数 (k, b)= 参数 (k, b)- 学习率 * 对应参数梯度（公式 1-3）

其中学习率通常用η表示，对应的每个参数梯度则根据深度学习框架的自动微分机制得到的，这样就实现了线性回归模型模型的构建与训练过程，最终根据输入的迭代次数运行输出就获取了回归直线的两个参数。完成了线性回归的求解。

1.5.2 线性回归代码演示

通过前面一小节的学习读者应该了什么是线性回归、线性回归是如何工作的，现在我们已经迫不及待的想在 Pytorch 中通过代码来验证我们上面的理论解释了。Pytorch 提供了丰富的函数组件可以帮助我们快速搭建线性回归模型并完成训练预测。

第一步：构建数据集

```
x = np.array([1,2,0.5,2.5,2.6,3.1], dtype=np.float32).
reshape((-1, 1))
```

```
y = np.array([3.7,4.6,1.65,5.68,5.98,6.95], dtype=np.float32).reshape(-1, 1)
```

第二步：根据公式 1-1 构建线性回归模型

```
class LinearRegressionModel(torch.nn.Module):
def __init__(self, input_dim, output_dim):
super(LinearRegressionModel, self).__init__()
self.linear = torch.nn.Linear(input_dim, output_dim)

def forward(self, x):
out = self.linear(x)
return out
```

LinearRegressionModel 是一个自定义的类，继承了 torch.nn.Module，其中 torch.nn.Linear 就表示构建了公式 1-1 的线性模型，重载方法 forward，表示根据模型计算返回预测结果。

第三步：创建损失功能与优化器

```
input_dim = 1
output_dim = 1
model = LinearRegressionModel(input_dim, output_dim)
criterion = torch.nn.MSELoss()

learning_rate = 0.01
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

其中 MSELoss 跟公式 1-2 表述一致、优化器 optimizer 完成自动梯度求解并根据公式 1-3 的方式来更新每个参数。

第四步：开始迭代训练过程

```
for epoch in range(100):
epoch += 1
# Convert numpy array to torch Variable
inputs = torch.from_numpy(x).requires_grad_()
labels = torch.from_numpy(y)

# Clear gradients w.r.t. parameters
optimizer.zero_grad()

# Forward to get output
outputs = model(inputs)

# Calculate Loss
loss = criterion(outputs, labels)

# Getting gradients w.r.t. parameters
loss.backward()

# Updating parameters
optimizer.step()

print('epoch {}, loss {}'.format(epoch, loss.item()))
```

这部分的代码注释都很清楚了，这里就不再赘述。

第五步：根据训练得到的参数，使用模型预测得到回归直线并显示，代码如下：

```
predicted = model(torch.from_numpy(x).requires_grad_()).data.numpy()

# Plot true data
plt.plot(x, y, 'go', label='True data', alpha=0.5)

# Plot predictions
plt.plot(x, predicted_y, '--', label='Predictions', alpha=0.5)

# Legend and plot
plt.legend(loc='best')
plt.show()
```

运行结果如下：

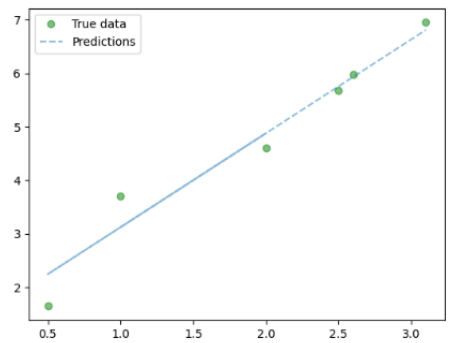

图 1-10

图 1-10 中点表示训练数据，直线表示线性回归预测。

总结线性回归这个入门级别例子演示，我们从中可以学习到一些基本组件函数，它们就是模型创建、损失函数、学习率、优化器，它们都是 Pytroch 开发中必须掌握的基本组件函数与方法，学会使用它们可以事半功倍，减少代码量，提升开发效率。

1.6 小结

本章是 Pytorch 框架与基础知识的介绍，通过本章学习了解 Pytorch 框架的历史与发展、理解深度学习框架常见的术语与词汇含义；安装 Python SDK、掌握 Pytorch 安装命令行、Pycharm IDE 安装与配置。学会使用 Pytorch 创建基本的张量、能够完成常见的张量数据操作、最后通过一个具有代表性的线性回归的例子，帮助大家真正打开 Pytroch 框架开发的大门。

本章的目标是帮助初学者厘清深度学习框架基本概念、基础组件与基础数据操作、同时通过案例激发起大家进一步学习的兴趣。

如欲了解更多 OpenVINO™ 开发资料，

请扫描下方二维码，我们会把最新资料及时推送给您。