dro 、rsome库学习与实践总结

目录

1. dro 库是什么？

一个功能强大全面的 用于**分布鲁棒优化 (Distributionally Robust Optimization)**Python 库。

- **核心思想**: 训练出的模型不仅在训练数据上表现良好，更能抵抗未知的数据分布变化，从而在现实世界中表现更稳定。
- **三大主要内容**:
  - **模型框架**: 支持线性模型、神经网络和树集成模型。
  - **DRO 算法**: 实现了多种主流的 DRO 方法（如 Wasserstein, CVaR, f-散度等）。
  - **数据工具**: 内置数据生成器，方便实验和复现。

2. 核心模块解析：模型 (dro.model)

库的核心，根据基础学习器分为三大类：

- **dro.linear_model (线性模型)**
  - 功能最丰富的模块，支持 SVM, Logistic 回归等。
  - 几乎涵盖所有主流 DRO 算法，如 WassersteinDRO, CVaRDRO, KLDRO 等。
- **dro.neural_model (神经网络)**
  - 将 DRO 应用于 PyTorch 神经网络。
  - 核心是 WNNDRO，通过**对抗训练**实现鲁棒性。
  - 支持内置的 resnet 等架构，也支持**注入用户自定义模型**。
- **dro.tree_model (树模型)**
  - 为 XGBoost 和 LightGBM 提供了 DRO 封装。

个人认为dro主要用于机器学习的任务，但是这不是固定的，可以通过dro的自定义板块灵活调整

3. 实践案例：鲁棒的 MNIST 数字识别

- **目标**: 使用 dro.neural_model.WNNDRO 训练一个能抵抗输入扰动的 MNIST 分类器
- **模型**: WNNDRO + 内置 ResNet 架构

- **数据**: 标准 MNIST 手写数字数据集

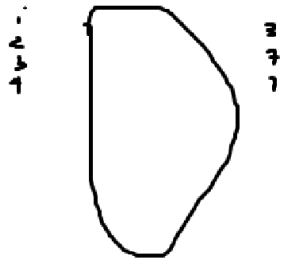
4. 问题与解决方案（基本语法问题而已，库还用得不熟练）

- **问题 1: 对抗攻击模块需要 4D 图像数据**
  - **报错**: broadcast error
  - **原因**: WNNDRO 的对抗攻击模块需要 [批次, 通道, 高, 宽] 格式的图像，但我传了展平后的向量
  - **解决**: 保持数据的四维图像形状 (N, 1, 28, 28)
- **问题 2: 内置 ResNet 的输入要求**
  - **报错**: DataValidationError
  - **原因**: dro 的 ResNet 模型要求输入是 **3 通道**且尺寸**不小于 32x32**
  - **解决**:
    1. 将 MNIST 图像从 28x28 **放大到 32x32**
    2. 将单通道灰度图**复制 3 次**，模拟成三通道 RGB 图像

5. dro 与标准 PyTorch 的对比

库	**dro 库 (WNNDRO)**	**标准 PyTorch**
核心思想	**分布鲁棒优化 (DRO)**	经验风险最小化 (ERM)
鲁棒性	通过对抗训练（也就是上面问题提到的对抗攻击），**天然具备鲁棒性**	默认不具备，需要额外实现
代码抽象	**高度封装**，调用 model.fit() 即可	**完全控制**，需手动编写训练循环
易用性	非常高，适合快速应用和实验	灵活度高，但代码量更大
准确率（5个epoch）	97.78%	99.12%

可以看到通过**pytorch**构建的模型在同样数据集同样训练次数情况下准确率比通过**dro**构建的模型准确率**更高**，这说明pytorch的模型更好吗？并不是，因为dro的训练过程有**抵抗输入扰动**，而pytorch没有，直接上结果吧：



我把这张阴间图片分别拿给dro和pytorch来认，结果如下：

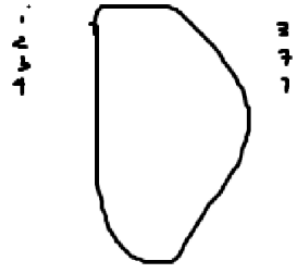
```
DRO 预测结果: 0
PyTorch 预测结果: 2
```

再附上一些大模型的回答（当个乐看，不具备参考性）：

99cbbcbd25755a6d04cb...
PNG 9.09KB

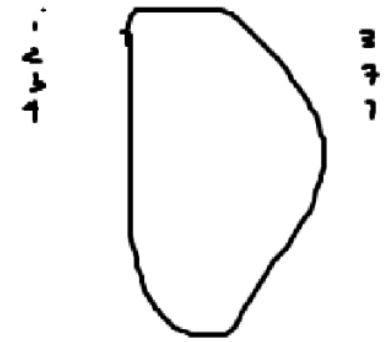
这是几

这是数字6。



这是几，直接回答一个数字，不要回答多余内容

3



这是几，这是数字识别任务，直接回答我一个数字

显示思路

371

这就是dro的作用。

6. 总结

- dro 库是一个设计精良的 DRO 工具箱，极大地**简化了鲁棒模型的开发**
- 使用 dro 的关键在于**理解并匹配模型对输入数据的格式要求**，尤其是处理图像数据时
- 通过 dro，我们可以轻松地为现有的机器学习模型（无论是线性模型、神经网络还是树模型）增加**分布鲁棒性**，使其在面对不确定的现实世界数据时更加可靠
- 虽然封装的东西多，但是并不死板，相反灵活性很高，用好自定义模型和自定义损失函数可以搞出花来

自定义模型可以通过update_user_mode方法实现，自定义损失可以通过MethodType方法和改变类的初始化来实现，前面提到dro主要用于机器学习的任务，其实也可以通过自定义损失函数来解决一般优化问题。

7. rsome概要

这部分目前我没有多说的，因为我没咋用rsome，这个库语法跟numpy有相似之处，并且有种所见即所得的感觉。跟一般的优化模型类似，这个也主要包含变量、约束、目标函数三个部分，先构建模型：
from rsome import ro

model = ro.Model('My model')

变量：两种类型，决策变量(dvar)和随机变量(rvar)，比如
x = model.dvar(3, vtype='I') vtype表示变量类型，I表示整数，B表示0-1变量，C表示连续变量，也是默认情况
y = model.dvar((3, 5), 'B') vtype可以不写出来
z = model.dvar((2, 3, 4, 5)) 创建了一个4维的连续变量

约束条件：用model.st()来设置，可以设置多个，用逗号隔开即可：
model.st(A @ x <= c) 设置单个约束
model.st(y.sum() >= 1,y.sum(axis=1) >= 0,A*y + x >= 0) 设置多个约束

目标函数：最大化model.max(b @ x)和最小化model.min(b @ x)