Neural Network Type 2 Classify Toolkit

Nanoka
Tianjin Chengjian University
zhengbindesu@gmail.com

Abstract

本文详细介绍了项目 Neural Network Type 2 Classify Toolkit 的具体内容
Github 链接：https://github.com/hhhhc-da/NeuralNetworkToolkit 同时，本项目已不再维护，还请读者自行解决其中的问题。

1. 前置准备

首先，我们需要先准备一套自己的 Python 环境，这样我们才有后面的工作。 首先我们先进入 Anaconda 官网（https://www.anaconda.com/download/）里，点击 Skip registration，选择 Windows Python 3.x 64-Bit Graphical Installer，之后会自动开启浏览器下载任务。 选择一个合适的位置安装好 Anaconda 后，我们需要把 Anaconda 的环境变量配置好，如图 Figure 1 所示。

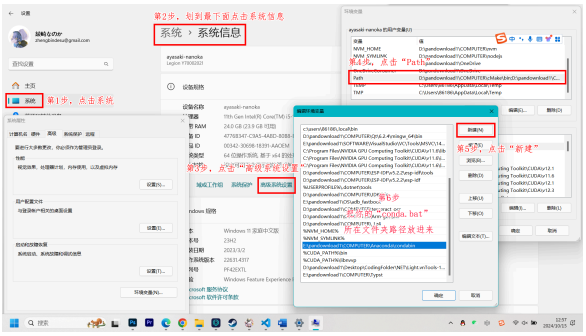

Figure 1: 文件结构图

然后我们同时按住 *win+R* 打开运行，然后输入 *cmd*，系统就会帮你打开命令行界面，全名叫命令提示符（cmd.exe）。输入 *conda –version* 就会显示 Anaconda 的版本信息，这时候就算安装成功了。 接下来我们安装 Python 环境，假如我们要创建一个叫 **pytorch** 的环境，那我们输入 *conda create -n* ***pytorch*** *python=3.10*，然后回车。系统会提示你需要安装一些包，直接回车确认即可，然后安装好后就可以继续下一步内容了。

2. Python 十分钟入门

Python 的文件后缀为 **.py**，所以我们可以直接创建一个 **.txt** 文件后，改名为后缀为 **.py** 的文件。我们创建一个 Helloworld.py, 输入 notepad Helloworld.py，系统会提示你找不到文件，自动创建到本文件夹下。 之后在记事本内输入 *print(“Hello world.”)* 这个命令，然后 *Ctrl+S* 保存即可退出记事本。那我们安装的环境还没有开启，自然是不能运行 Python 的，所以我们要在命令提示符内运行 *conda activate* ***pytorch***，之后系统就会在你的语句前加一个 *(pytorch)* 来表示你开启了环境，之后输入 *python –version* 可以查看 python 版本。 查看到版本之后（比如我的是 3.10.15），我们输入 *python Helloworld.py* 就可以看到命令提示符中打印出了 *Hello world!*，剩下的内容你需要进行系统的学习，才可以完全掌握（小提示：你在输入到 *python He* 的时候，按一下键盘 **Tab**，系统会自动帮你补全文件）。同时，Anaconda 会自动给你生成一个 *base* 环境，你也可以直接使用这个进行开发，使用方法类似，输入 *conda activate base* 即可。

3. 文件结构

我们来看一下整套 Toolkit 的文件图，如图 Figure 2 所示。 其中 GUI 文件夹下都是有关小猿口算的内容，NeuralNetwork 文件夹下的内容才是神经网络的 Toolkit。按照文件来说明，*make_data.py* 是整个项目最开始需要使用的，这个文件可以快速截取屏幕内容，并将其存储在 **./traindata/unform/image** 中。如果你要做普通的二分类任务，那么你在有数据的时候也可以直接拖到这个文件夹下，如果是 **.csv** 或者其他类型的，还需要自己适配对应的生成文件。我们直接找一个猫狗识别训练集，然后把所有的图片放到 **./traindata/unform/image** 中即可，然后我们在文件系统的工作就完成了（这部分我放在群文件）。

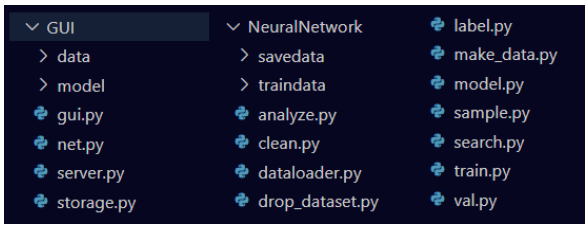

Figure 2: 文件结构图

4. 数据预处理

4.1. 数据标注

如果你感兴趣的话，顺手还能做一个 Kaggle。我们放进去之后，注意 Kaggle 的 0、1 分别表示什么，同时还要注意提交格式（Kaggle 原文：***Your submission should have a header. For each image in the test set, predict a label for its id (1 = dog, 0 = cat)***），输入 *python label.py* 开始打标签，类似于图

Figure 3，左边是 0，右边是 1，请注意区分好，打标签错了也不要慌，直接根据输出找到对应文件改了就行。

Figure 3: label.py 绘制的打标签工具

4.2. 平衡分析

打好标签后，我们粗略的分析一下这个数据集，我们可以看到使用 *python analyze.py* 后输出了类似于图 Figure 4。

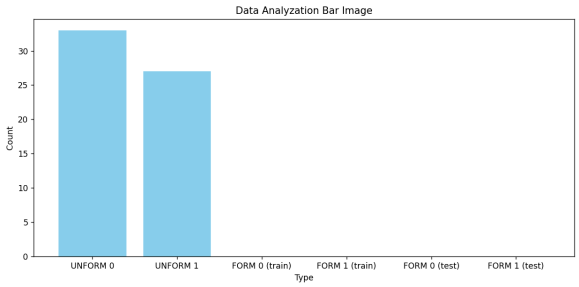

Figure 4: 数据集标签分析统计图

可以看出，数据集不算倾斜，如果出现了一边很多、另一边很少的情况，就要考虑你的神经网络会不会正确的区分特征了。（举个例子：如果你有 100 张人像，1 张猫咪照片，那么网络只会输出人类类型，因为在这 100 张图片中学习人类的所有特征，对最后的优化器来说，提升比区分人类和猫咪的区别要大得多!）据我的经验，这个样子应该是没什么问题的，即使我只放了 60 张数据。

4.3. 数据切分

下一步输入 *python sample.py* 来进行数据集分割和采样，可以看到输出如图 Figure 5 所示。

```
52  traindata\unform\image\57.jpg  traindata\unform\label\57.txt
53  traindata\unform\image\58.jpg  traindata\unform\label\58.txt
54  traindata\unform\image\59.jpg  traindata\unform\label\59.txt
55   traindata\unform\image\6.jpg   traindata\unform\label\6.txt
56  traindata\unform\image\60.jpg  traindata\unform\label\60.txt
57   traindata\unform\image\7.jpg   traindata\unform\label\7.txt
58   traindata\unform\image\8.jpg   traindata\unform\label\8.txt
59   traindata\unform\image\9.jpg   traindata\unform\label\9.txt
名称完全匹配，开始切分数据集
训练集 Length: 48 & 48，测试集 Length: 12 & 12，测试集比例: 0.2
Done.
```

Figure 5: 数据随机切割输出

此时，我们的数据集已经准备完毕，如果你的目标要求不是很高，只要做一个二分类的神经网络，那么基本上已经快结束了。

4.4. 初步训练

输入 *python train.py* 就可以看到结果了，模型损失折线图如图 Figure 6 所示，训练时也会展示训练进度，训练比较大的模型时可能会比较有用，模型进度条展示图如图 Figure 7 所示。

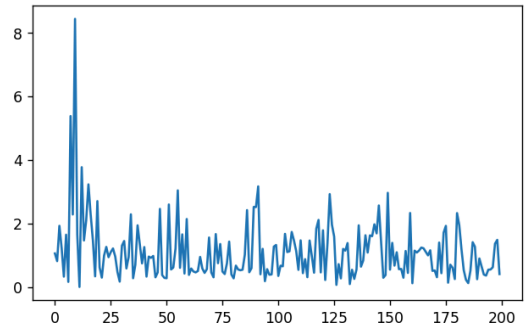

Figure 6: 模型训练损失折线图

```
EPOCH 4: 100%| 10/10 [00:00<00:00, 56.68it/s, loss=0.833]
EPOCH 5: 100%| 10/10 [00:00<00:00, 54.83it/s, loss=1.31]
EPOCH 6: 100%| 10/10 [00:00<00:00, 55.29it/s, loss=0.651]
EPOCH 7: 100%| 10/10 [00:00<00:00, 54.83it/s, loss=0.8]
EPOCH 8: 100%| 10/10 [00:00<00:00, 55.74it/s, loss=0.959]
EPOCH 9: 100%| 10/10 [00:00<00:00, 56.69it/s, loss=1.14]
TEST: 100%| 3/3 [00:00<00:00, 75.25it/s, loss=1.04]
EPOCH 10: 100%| 10/10 [00:00<00:00, 56.37it/s, loss=1.04]
EPOCH 11: 100%| 10/10 [00:00<00:00, 53.74it/s, loss=1.04]
EPOCH 12: 100%| 10/10 [00:00<00:00, 55.13it/s, loss=1.21]
EPOCH 13: 100%| 10/10 [00:00<00:00, 54.53it/s, loss=0.902]
EPOCH 14: 100%| 10/10 [00:00<00:00, 55.43it/s, loss=1.56]
EPOCH 15: 100%| 10/10 [00:00<00:00, 54.83it/s, loss=0.904]
EPOCH 16: 100%| 10/10 [00:00<00:00, 57.33it/s, loss=0.973]
EPOCH 17: 100%| 10/10 [00:00<00:00, 54.79it/s, loss=0.8]
EPOCH 18: 100%| 10/10 [00:00<00:00, 56.69it/s, loss=0.978]
EPOCH 19: 100%| 10/10 [00:00<00:00, 56.37it/s, loss=0.727]
TEST: 100%| 3/3 [00:00<00:00, 200.69it/s, loss=0.856]
模型文件已保存到: savedata\resnet18.pth
Done.
```

Figure 7: 模型训练示意图

4.5. 抽样分析

输入 *python val.py* 采样测试数据，程序会自动弹窗显示测试样本和预测标签，如图 Figure 8 所示。

Figure 8: 数据抽样预测

看得出模型效果一塌糊涂（给我都写笑了）。

5. 神经网络进阶

5.1. 数据归一化

模型训练不出来有多种因素，数据不够充分、数据分布不典型、数据预处理方式不佳、学习率太高、迭代器选择错误等都可能导致模型训练失败。那么这时候我们就要一一排查，我们首先要想到的，就是数据规不规范。神经网络处理的值大多数是区间 **[0, 1]** 内的数据，所以我们首先检查 *dataloader.py*。文件内容如图 Figure 9 所示。这一部分详细介绍了图片从哪里加载，并创建了类内的公有成员供后续使用，然后通过预处理后，将所有内容保存在 **self.images** 和 **self.labels** 中， 后续预处理的代码如图 Figure 10 所示（这里有一个细节，就是我本打算使用软标签作为类别以增强模型泛化能力的，但是后续对接 **scikit-learn** 的时候遇到了困难，所以便放弃了）。

```python
class MyDataset(Dataset):
    def __init__(self, root_dir="traindata", mode="train"):
        Dataset.__init__(self)

        self.mode = mode
        self.root_dir = root_dir
        self.image_dir = os.path.join(self.root_dir, "image", self.mode)
        self.label_dir = os.path.join(self.root_dir, "label", self.mode)

        self.image_file = os.listdir(self.image_dir)
        self.length = len(self.image_file)

        # 检查，如果 analyze 过了可以省略
        if len(self.image_dir) != len(self.label_dir) and len(self.image_dir) > 0:
            raise RuntimeError("Fatal Error: 文件个数不匹配")

        # 将所有图片数据加载，因为有黑白两种照片，不适合标准化
        # 我们使用 Min-Max 归一化

        self.images = []
        self.labels = []
```

Figure 9: 数据预处理 Part.1

```python
for image_path in self.image_file:
    image = cv2.imread(os.path.join(self.image_dir, image_path))
    image = cv2.resize(image, (128, 128))
    image = image.transpose(2, 0, 1)

    self.images.append(np.array(image, dtype=np.float32)/255.0)

    with open(os.path.join(self.label_dir, image_path[:-4] + ".txt"), "r") as f:
        l = int(f.read())
        f.close()

        self.labels.append(l)

        # if l == 0:
        #     self.labels.append(np.array([1,0], dtype=np.float32))
        # elif l == 1:
        #     self.labels.append(np.array([0,1], dtype=np.float32))
        # else:
        #     raise RuntimeError("Unknown label")

# 转换为 torch.FloatTensor
self.images = torch.tensor(self.images, dtype=torch.float32)
self.labels = torch.tensor(self.labels, dtype=torch.long)
```

Figure 10: 数据预处理 Part.2

之后，我们还要规定好数据集怎么取数据，怎么确定数据长度，如图 Figure 11 所示（顺手还做了一个对接用的 API，向其他库传递数据时使用）。

```python
def __len__(self):
    return self.length

def __getitem__(self, idx):
    return self.images[idx], self.labels[idx]

def data(self):
    return np.array(self.images, dtype=np.float32), np.array(self.labels, dtype=np.long)
```

Figure 11: 数据预处理 Part.3

科普一个小知识，那就是图片文件可以直接除 255.0 进行归一化，这是 **Min-Max-Scale** 最大-最小值归一化，一般来说做图像会考虑这个，比较简单。当然，也有其他的归一化方法，如 **Stdandard-Scale** 标准归一化，可以去百度了解一下，但是因为处理图像数据比较庞大需要用到 **ReLU** 所以一般不考虑这个。 同样的，神经网络还有很多约束方式，如：正则化、AdamW 迭代正则化、图像维度变换（变换为 NCHW 格式）、模糊加噪、数据增强、图片降噪、YUV 图像直方图均衡化、马尔科夫网参数预训练等多种技术， 这里我都没有介绍（因为我比较忙）。

5.2. 网格搜索

顾名思义，网格搜索就是你给出几个可能的超参数值，然后让机器自动帮你每个都跑一遍，然后看哪个效果最好，给出超参数集合如图 Figure 12 所示。然后我们运行 *python searchcv.py* 就可以全自动搜索最佳参数了，我这里用的是宏平均 **F1-Score** 作为评价标准，结果如图 Figure 13 所示。

```python
# 设置超参数网格
param_grid = {
    'learning_rate': [0.05, 0.01, 0.005, 0.001],
    'batch_size': [2, 5, 10, 15, 20],
    'episode': [5, 10, 15, 20, 30]  # 可以根据需要调整
}
```

Figure 12: 超参数网格

```
最佳参数: {'batch_size': 10, 'episode': 15, 'learning_rate': 0.005}
最佳得分: 0.7865160986758083

所有参数组合的得分:
                                               params  ...  rank_test_score
0   {'batch_size': 2, 'episode': 5, 'learning_rate...  ...               93
1   {'batch_size': 2, 'episode': 5, 'learning_rate...  ...                5
2   {'batch_size': 2, 'episode': 5, 'learning_rate...  ...               62
3   {'batch_size': 2, 'episode': 5, 'learning_rate...  ...               84
4   {'batch_size': 2, 'episode': 10, 'learning_rat...  ...               78
..                                                ...  ...              ...
95  {'batch_size': 20, 'episode': 20, 'learning_ra...  ...                6
96  {'batch_size': 20, 'episode': 30, 'learning_ra...  ...                8
97  {'batch_size': 20, 'episode': 30, 'learning_ra...  ...               15
98  {'batch_size': 20, 'episode': 30, 'learning_ra...  ...               83
99  {'batch_size': 20, 'episode': 30, 'learning_ra...  ...               87

[100 rows x 4 columns]
x_test.shape: (12, 3, 128, 128), y_test.shape: (12,)

混淆矩阵:
[[6 1]
 [1 4]]

分类报告:
              precision    recall  f1-score   support

           0       0.86      0.86      0.86         7
           1       0.80      0.80      0.80         5
```

Figure 13: GridSearchCV 输出网格搜索的结果

其中，我们观察混淆矩阵，四个从左往右从上到下分别是 **TP**、**FP**、**TN**、**FN**，表示正例被预测为正例的个数、正例被预测为负例的个数、负例被预测为正例的个数、负例被预测为负例的个数，对角线上支持数据比较多、反对角线上支持数据比较少就说明效果不错，看得出这么点数据还不够网络塞牙缝的。之后最上面显示了最佳的超参数组合，我们直接搬运到 train.py 就可以更好的训练模型了。