Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

数据处理：飞一般的感觉

硕士研究生 徐泽豪

2022年12月25日

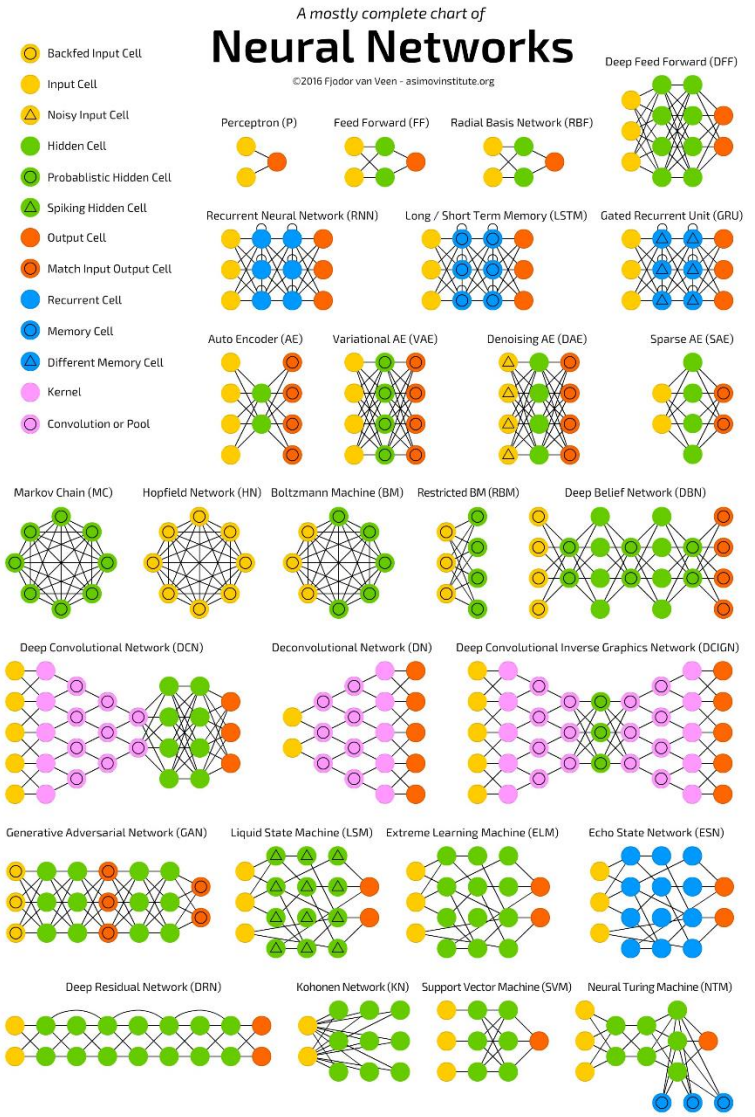
内容提要

- 背景简介
- 数据读取优化
  - 文件格式转化
- 数据处理优化
  - 向量化处理
- 应用总结
- 参考文献

背景简介

- 预期收获
  - 了解数据读取优化的基本方法
  - 了解数据处理优化的基本方法
  - 理解数据优化的基本原理
  - 了解数据优化的前沿发展

背景简介　研究背景



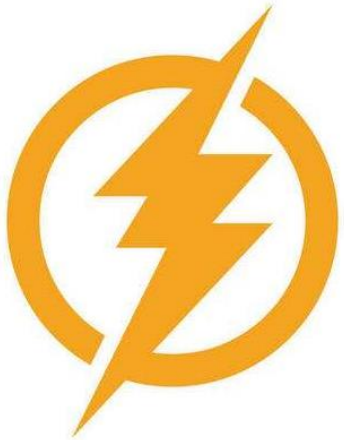
- 研究背景
  - 神经网络研究火热，模型众多
  - 实验室大量课题项目与神经网络有关
- 存在问题
  - 训练反馈时间长
  - 实验室CPU/GPU资源紧张
- 解决方案
  - 综合利用CPU与GPU资源
  - 提升数据的读写速度

解决方案

- 如何优雅地进行模型训练?
  - 瓶颈主要体现在GPU显存不足
    - 梯度累加
    - 混合精度训练
    - Gradient Checkpointing

沈宇辉	如何优雅地进行模型训练	2022-05-04 17:00:00
王帅鹏	如何优雅地阅读和复用代码	2019-12-29 19:00:00
李橙	如何优雅的开发	2019-12-22 19:00:00

- 如何简单粗暴地降低模型训练时间?
  - 综合利用CPU与GPU资源
    - 优化数据读取
    - 优化数据处理



数据读取优化

数据读取优化

T	降低数据读取时间与内存占用以高效利用GPU
I	6种数据格式
P	1.读写不同格式的数据集 2.记录读写时间与文件大小 3.采用混合精度降低内存占用
O	处理后的数据

P	针对不同数据集和功能需求选择数据格式
C	数据为行列结构的数据帧格式
D	降低内存占用后的数据处理
L	

数据读取优化

EC_20220529-20220604.csv
EC_20220605-20220611.csv
EC_20220612-20220618.csv
EC_20220619-20220625.csv
EC_20220626-20220702.csv

- 最常用的数据格式.csv
  - 以纯文本形式存储表格数据
  - 该文件是一个字符序列，不含必须像二进制数字那样被解读的数据
  - 是一种通用的、相对简单的文件格式

- Pandas 读取
  - df = pandas.read_csv()

- 应用广泛就一定是最好的吗?
  - Python专用的文件类型
  - 读取速度
  - 占用空间

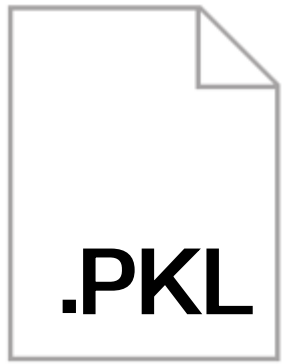
数据读取优化

- .csv
  - 以纯文本形式存储表格数据
  - 1GB文本包含的内容并不一定是1GB

整数：1,000,000,000 < 2^{30}

C++：4字节
文本：10字节

- .pkl 格式（.pickle）
  - python中独有的序列化模块
  - 将内存中对象转换为二进制编码的bytes字符进行保存
  - 能够保存python的复杂的数据类型，包括列表、元组、自定义类等

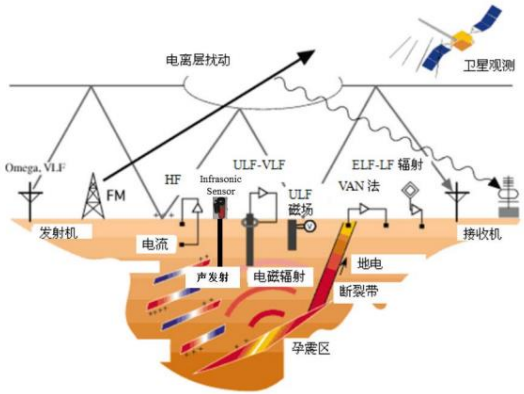


数据读取优化

- 数据来源
  - 小组培训考核： AETA地震预测AI算法大赛

StationID	TimeStamp	magn@var	magn@po	magn@ske	magn@kur	magn@ab	magn@ab	magn@ab	magn@ab	magn@en	magn@en	magn@po	magn@po	magn@po	magn@po
19	1.67E+09	0.595526	0.595684	0.037021	-1.61074	1.102909	0.708637	1.027494	0.99723	0.009918	0.254206	0.000158	9.21E-07	1.71E-06	7.19E-06
19	1.67E+09	0.576698	0.576854	0.037721	-1.61139	1.093383	0.696477	1.009418	0.977805	0.013893	0.252333	0.000158	1.53E-06	2.31E-06	7.96E-06
19	1.67E+09	0.570916	0.571073	0.038506	-1.61522	1.07107	0.694034	1.002968	0.973417	0.00887	0.242467	0.000158	1.00E-06	1.78E-06	7.36E-06
19	1.67E+09	0.572482	0.572639	0.037195	-1.61784	1.085808	0.694862	0.998768	0.967605	0.013336	0.254418	0.000158	1.87E-06	2.68E-06	8.27E-06
19	1.67E+09	0.548904	0.549061	0.038666	-1.61985	1.05922	0.681065	0.981142	0.953879	0.007955	0.233116	0.000158	1.58E-06	2.40E-06	7.87E-06
19	1.67E+09	0.598196	0.598353	0.036164	-1.61106	1.114197	0.707389	1.021906	0.990705	0.014083	0.260308	0.000158	1.23E-06	1.90E-06	7.49E-06
19	1.67E+09	0.55697	0.557126	0.038617	-1.61098	1.078758	0.684977	0.99213	0.958717	0.012553	0.245189	0.000158	1.80E-06	2.56E-06	7.92E-06
19	1.67E+09	0.524969	0.525126	0.03862	-1.61676	1.037357	0.666229	0.959617	0.929578	0.012715	0.228966	0.000157	5.63E-07	1.32E-06	6.79E-06
19	1.67E+09	0.508513	0.508669	0.039751	-1.61819	1.023369	0.65484	0.944466	0.91484	0.010578	0.221929	0.000159	2.55E-06	3.50E-06	8.97E-06

- 数据构成
  - 川滇实验场的电磁观测数据，共计 95 个特征，由整型与浮点型数据组成



数据读取优化

- .csv VS .pkl 效果对比

数据类型	空间占用	缩减比例	内存占用	读取用时
1.csv	2.82M	8.15%	2.57M	14.04ms
1.pkl	2.59M		2.59M	1.0ms
2.csv	178.6M	41.37%	104.67M	1315.82ms
2.pkl	104.67M		104.67M	101.76ms
3.csv	7.06G	43.40%	4087.78M	56.592s
3.pkl	3.99G		4087.78M	4.263s

– 分“雏量级、羽量级、轻量级”数据集进行对比
– 占用空间缩小40%左右，读取速度提升了12倍左右

数据读取优化

- 降低内存占用（混合精度）
  - 方法来源：AETA/Kaggle
  - 将浮点数和整数转换为它们的最小子类型

  GradScaler 和 autocast

- Python资源回收
  - 引用计数（基础模块）
  - 分代垃圾（gc, generational cyclic）
  - del 删除的是变量，而非数据对象
  - gc.collect()清除内存

```python
for col in df.columns:
    col_type = df[col].dtypes
    if col_type != object:
        c_min = df[col].min()
        c_max = df[col].max()
        if str(col_type)[:3] == 'int':
            if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                df[col] = df[col].astype(np.int8)
            elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                df[col] = df[col].astype(np.int16)
            elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                df[col] = df[col].astype(np.int32)
            elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                df[col] = df[col].astype(np.int64)
        else:
            if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                df[col] = df[col].astype(np.float16)
            elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                df[col] = df[col].astype(np.float32)
            else:
                df[col] = df[col].astype(np.float64)
```

```python
final_df.to_csv(f'./data/{type}.csv')
final_df.to_pickle(Merged_Data_Path[type])  # 导出为pkl文件
del final_df
gc.collect()
```

数据读取优化

- .csv VS .pkl 降低内存占用效果对比

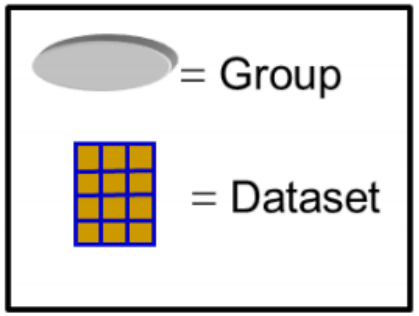
数据类型	读取用时	压缩用时	内存占用	处理后内存占用	降低比例
1.csv	14.04ms	31.25ms	2.57M	0.96M	62.50%
1.pkl	1.0ms	31.25ms	2.59M	1.46M	43.75%
2.csv	1315.82ms	921.99ms	104.67M	27.90M	73.35%
2.pkl	101.76ms	1092.00ms	104.67M	27.90M	73.35%
3.csv	56.592s	39.533s	4087.78M	1173.34M	71.30%
3.pkl	4.263s	44.414s	4087.78M	1211.19M	70.37%

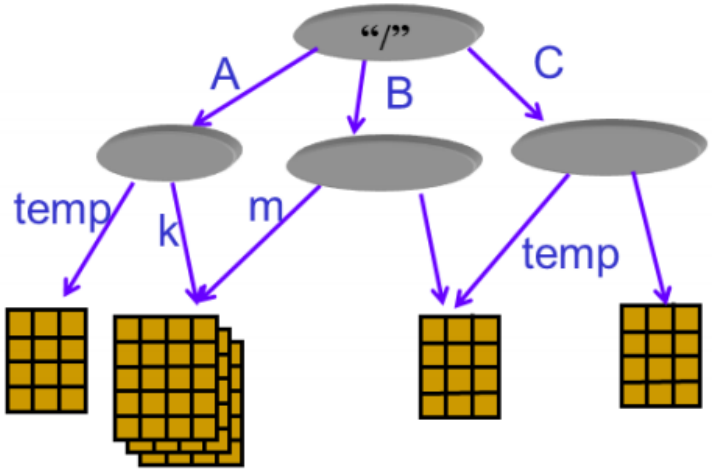
– 分“雏量级、羽量级、轻量级”数据集进行对比

– 对于由整型与浮点型数据组成的数据集可将内存占用空间缩小70%左右

数据读取优化

- 层级数据格式.hdf5
  - 一个文件当中可以存放不同种类的数据集，这些数据集使用group进行管理
  - 其管理方式类似文件管理系统，不同的文件位于不同的目录下
  - 目录即hdf5中的group, 描述数据集dataset的分类信息，通过group 有效的将多种dataset 进行管理和区分
  - 文件即hdf5中的dataset, 表示具体的数据





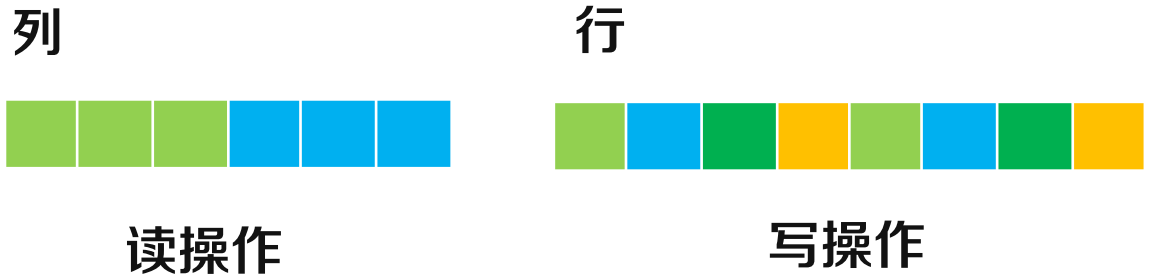
数据读取优化

- .feather
  - 速度更快、压缩后更加轻量级的二进制保存格式
  - Feather 是 Apache Arrow 项目中的一种数据格式，但是由于其性能优异也将其转移到了Python中
  - Python 和 R 之间数据快速交互而设计的，尽可能提高数据在内存中转换的效率（Julia）
  - 将内存中的数据几乎原封不动写入到文件中，因此具有超高的写入速度，且文件体积几乎不能再次缩小

数据读取优化

- .parquet
  - Apache Parquet是面向分析型业务的列式存储格式
  - 列式存储有着更高的压缩比，相同类型的数据为一列存储在一起方便压缩，不同列可以采用不同的压缩方式
  - 结合Parquet的嵌套数据类型，可以通过高效的编码和压缩方式降低存储空间提高I/O效率

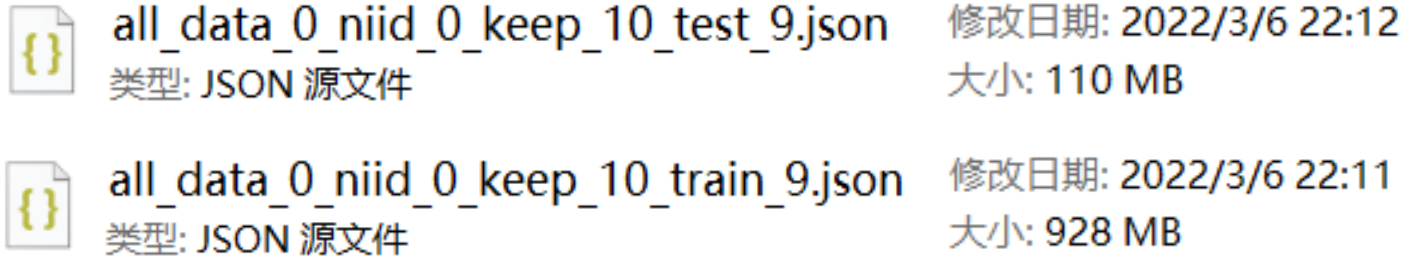
地点	经度	纬度	震级
四川宜宾市珙县	104.77	28.17	3.7
贵州毕节市赫章县	104.63	27.16	4.5
云南昆明市东川区	103.13	26.02	4.2



数据读取优化

- JavaScript对象表示.json
  - 层次结构清晰，易于人阅读和编写，同时也易于机器解析和生成，是理想的数据交换语言，以简单文本格式存储，可以在任何文本编辑器中查看

- MLSys 2020 FedProx
  - MNIST数据集
  - 非独立同分布处理



- 问题：json文件可以被各种语言读取，适用范围广，但它效率高吗?

数据读取优化

- 六种数据格式效果对比（178M.csv文件为基准）

数据类型	保存时间	保存倍率	读取时间	读取倍率	文件大小	大小倍率
csv	11286.02ms	1	1335.30ms	1	178.6M	1
hdf	303.57ms	0.0269	136.03ms	0.1019	106.5M	0.5974
pkl	82.11ms	0.007	38.01ms	0.0285	104.5M	0.5963
feather	73.86ms	0.006	107.02ms	0.0801	99.9M	0.5601
parquet	953.77ms	0.084	172.04ms	0.1288	114.1M	0.6401
json	1908.95ms	0.1691	6503.56ms	4.85	259.9M	1.4579

- .pkl
  - 对于由整型与浮点型数据组合的数据集综合得分最高
- .json
  - 读取速度，文件大小等指标反而弱于csv

数据读取优化

- .hdf5
  - 文件更大（尤其是纯文本）、存取速度比feather慢；但hdf5更类似SQL，可以按条件指定行、列进行存取，而feather只能如同csv一般在读取时指定列名读取
- .pkl
  - 用于序列化，存储中间变量数据
- .feather
  - 用于存储，压缩空间，写入速度快
- .parquet
  - 工程上用于分区过滤与列修剪（Spark）
  - 大幅节省磁盘I/O，减轻服务器的压力

纯文本数据集效果

（重复字符串）

数据处理优化

数据处理优化

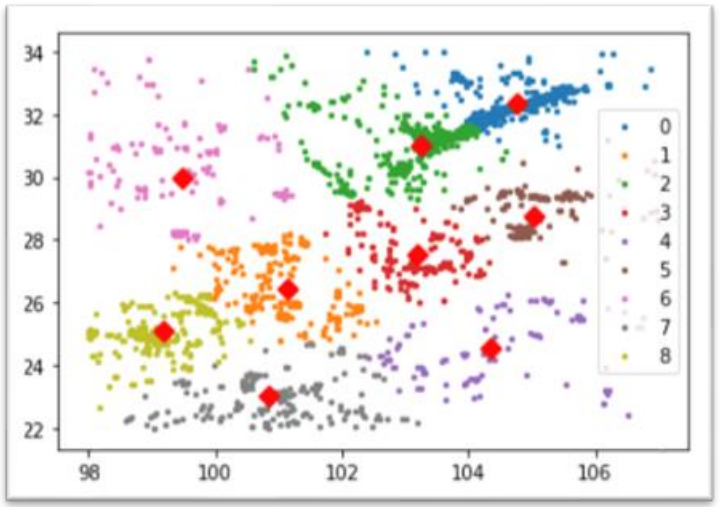
T	优化逐行操作，降低数据处理时间
I	以整型与浮点型数据构成的数据
P	1.改写数据处理函数 2.使用向量化处理方式降低数据处理时间
O	处理后的数据

P	针对功能需求改写数据处理函数
C	数据为行列结构的数据帧格式
D	数据处理函数改写
L	

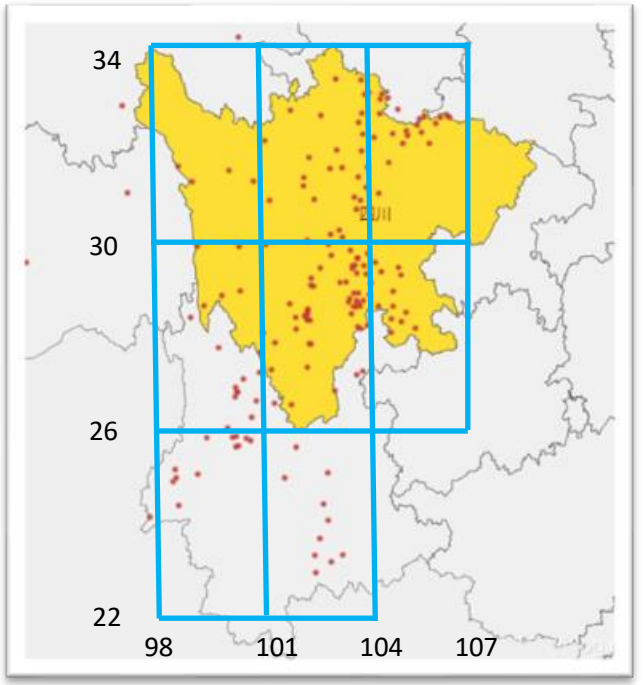
数据处理优化

- AETA数据预处理
  - 根据站台经纬度和区域划分结果对站台的电磁数据进行处理
  - （聚类/经纬度划分）
  - 对站台重新编号?
    - 每次有新数据时又要重新处理
  - 直接逐行遍历判断并进行处理

Title	StationID	Longitude	Latitude
都江堰中学	19	103.65	30.98
通海山洞	24	102.75	24.12
康定姑咱山洞B	26	102.17	30.12
楚雄山洞	29	101.54	25.03
西昌气象局	32	102.27	27.9
石棉挖角乡	33	102.28	29.31
石棉山洞B	35	102.35	29.23
永胜期纳镇	36	100.62	26.33
冕宁防震减灾局	38	102.17	28.55
巧家地震台	39	102.94	26.9



站台信息与区域划分



数据处理优化

- 逐行操作优化（举例）
  - 定义权重处理函数data_process1-4
  - 根据电磁数据中功率谱与所属区域确定权重，并将其写入dataframe中

- 权重函数

```
def new_weight(power, StationID):
    if StationID in area1:
        weight = data_process1(power)
    elif StationID in area2:
        weight = data_process2(power)
    elif StationID in area3:
        weight = data_process3(power)
    elif StationID in area4:
        weight = data_process14(power)
    else:
        weight = default_data(power)
    return weight
```

权重函数
new_weight

for与apply方法均可以使用，改写工作量较小

- 方法1：for函数遍历

```
def data_for(df):
    weight_list = []
    for i in range(len(df)):
        power = df.iloc[i]['magn@power']
        StationID = df.iloc[i]['StationID']
        weight = new_weight(power, StationID)
        weight_list.append(weight)
    df['weight'] = weight_list
```

- 方法2：apply与lambda函数

```
def data_apply(df):
    df['weight'] = df.apply(
        lambda row: new_weight(
            power=row['magn@power'],
            StationID=row['StationID']),
        axis=1)
```

数据处理优化

- 方法3：isin函数

```
def data_isin(df):
    area_1 = df.index.isin(area1)
    area_2 = df.index.isin(area2)
    area_3 = df.index.isin(area3)
    area_4 = df.index.isin(area4)
    area_d = ~df.index.isin([area1, area2, area3, area4])

    df.loc[area_1, 'weight'] = data_process1(df.loc[area_1, 'magn@power'])
    df.loc[area_2, 'weight'] = data_process2(df.loc[area_2, 'magn@power'])
    df.loc[area_3, 'weight'] = data_process3(df.loc[area_3, 'magn@power'])
    df.loc[area_4, 'weight'] = data_process4(df.loc[area_4, 'magn@power'])
    df.loc[area_d, 'weight'] = default_data(df.loc[area_d, 'magn@power'])
```

```
    df.set_index('StationID', inplace=True)
    data_isin(df)
```

isin方法接受一个列表，判断该列中元素是否在列表中

此时权重函数new_weight无法复用

isin局限性更大，需要重新编写总权重函数，不适用于所有场合

灵活性：

for > apply > isin

数据处理优化

- for、apply、isin处理效果对比

方法	处理时间	处理倍率
for	31283.07ms	1
apply	2066.47ms	0.066
isin	28.01ms	0.0009

- for与isin
  - 均获得了很高的性能提升
- 问题：for/apply/isin的时间复杂度均为O(n)，为什么for慢了这么多?

数据处理优化

从三个现实案例引入

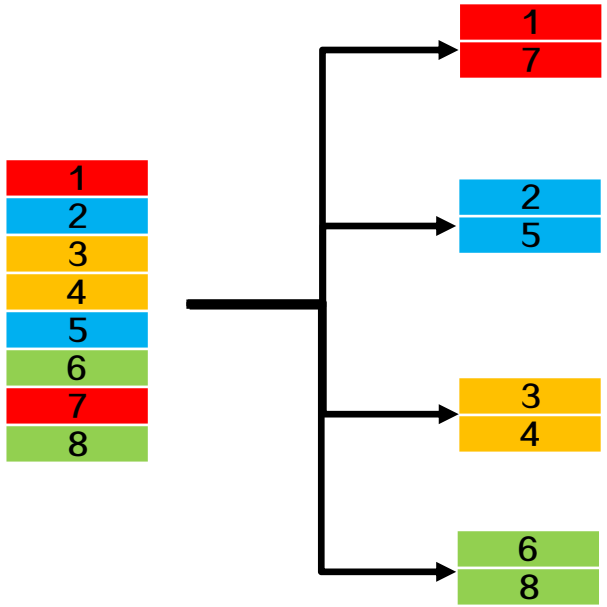
- 两副扑克牌混在一起，2个人分工合作，如何以最快速度将两副牌分开?

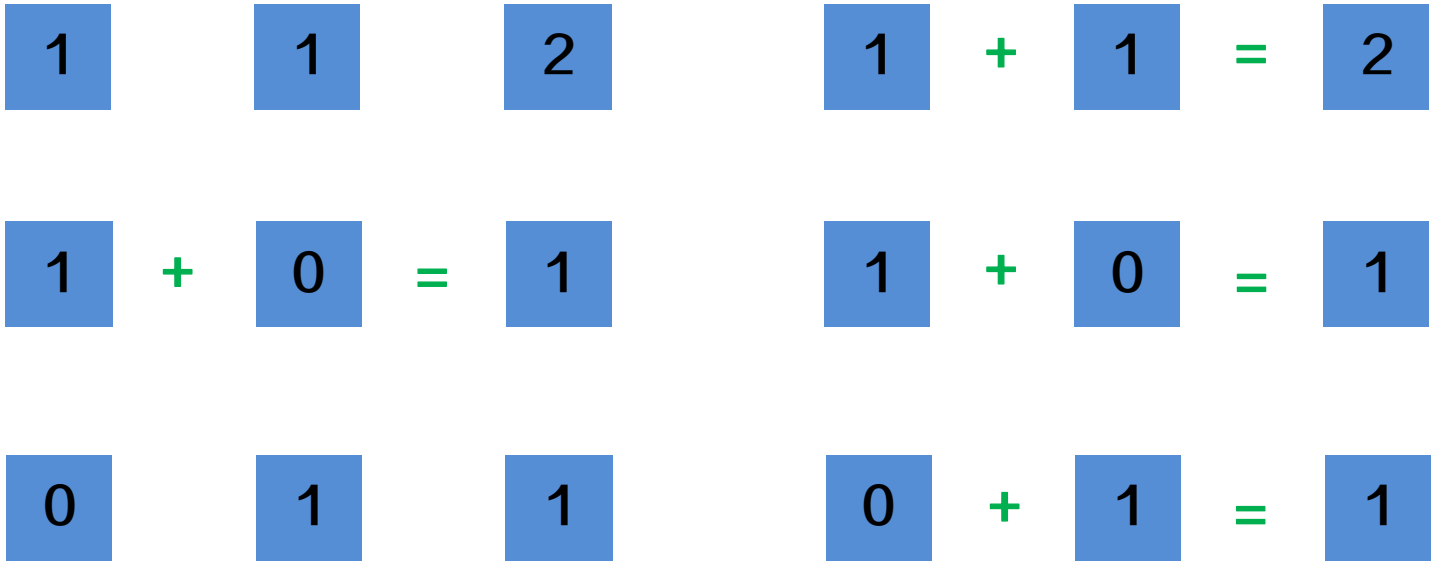
- 老师找你协助毕业材料归档，30个人的毕业论文、材料文件袋、还有其他文件，每一叠材料都以乱序叠放，如何以最快速度分出每一个人的各种材料，并将其装入对应的文件袋中完成归档?

- 海淀区人大代表投票完成后进行计票，2000张选票，如何以最快速度完成计票?

数据处理优化

- 向量化的处理方式
  - 向量化计算是一种特殊的并行计算的方式，它可以在同一时间执行多次操作，通常是对不同的数据执行同样的一个或一批指令，或者说把指令应用于一个数组/向量
  - 向量化计算使用了python的内建函数，调用了CPU/GPU的SIMD指令集进行计算，大大减少了因为python高级语言执行损耗的时间





应用总结

应用总结

- 数据读取优化
  - 对比了各数据格式的优劣
  - pkl：序列化，保存中间变量，读取速度快
  - feather：写入速度快
  - parquet：压缩率高，适用于字符串数据集

- 数据处理优化
  - 向量化的处理方式
  - 使用apply，isin 替代 for

应用总结

前沿发展

- Pytorch 2.0 稳定版
  - torch.compile(model)，返回原来模型的引用，但将forward函数编译成一个更优化的版本
  - 2023.3发布稳定版

- OneFlow（兼容Pytorch）
  - nn.Module（eager）动态图更易搭建网络、调试和验证算法
  - nn.graph 保留静态图高性能优势

未来展望

- Dataloader参数讲解
  - num_workers、pin_memory
  - 多进程数据加载与锁页内存

- Pytorch 建模细节
  - prefetch_generator
    - 创建 worker 线程预读取新的数据
  - torch.backends.cudnn.benchmark = True
    - 寻找最佳算法（计算不同内核大小卷积的cuDNN算法的性能不同）
  - torch.no_grad()
    - 验证期间设置

参考文献

- 易执. 提速百倍的Pandas性能优化方法，让你的Pandas飞起来[EB/OL]. (2020-04-25)[2022-12-25]. https://zhuanlan.zhihu.com/p/97012199.
- 北京大学深圳研究生院. AETA地震预测AI算法大赛[EB/OL]. (2022-06-01)[2022-12-25]. https://tianchi.aliyun.com/competition/entrance/531972/introduction.
- PyTorch. Performance Tuning Guide [EB/OL]. (2022-06-02)[2022-12-25]. https://pytorch.org/tutorials/recipes/recipes/tuning_guide.html#general-optimizations.
- Pandas. pandas.DataFrame.to_feather[EB/OL]. (2022-11-16)[2022-12-25]. https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_feather.html

谢谢！

大成若缺，其用不弊。大盈若冲，其用不穷。大直若屈。大巧若拙。大辩若讷。静胜躁，寒胜热。清静为天下正。