DL深度学习平台

崔剑雄 v20220222

初始化

先用shell文件初始化项目（包含workspace的创建，python环境激活等）

bash下输入:

$source /mnt/lustre/group/aum/init_DL.sh

提示输出dirname: 输入想要本地工作的路径，如当前目录 "./"

可以看到本地创建了工作目录 ./dl_dev_local, 并且激活了python环境aum

** 注意：不要更改目录dl_dev_local和run.py的名称。

目录：dl_dev_local

```
├── models/
|   ├── __init__.py
|   ├── BasicModule.py      ## 模型基类
|   ├── single_input/
|      ├── __init__.py     ## 绑定模型
|      ├── Model1.py       ## 模型文件
|      ├── Model2.py
|      ├── ...
|    ├── multi_input/
|      ├── __init__.py
|      ├── ...
|   └──  combine/
|      ├── __init__.py
|      ├── ...
|
├── trials/             ## 实验任务文件夹
|   ├── demo.yaml               ## 实验任务1
|   ├── demo_mi.yaml            ## 实验任务2
|   ├── ...
|
├── outputs/
|   ├── 20211009/                   ## 实验日期
|      ├── model1_name_time/              ## 每次实验的结果文件夹
|         ├── checkpoints/              ## 模型文件
|           ├── model1_epoch1.pth      ## 各个epoch
|           ├── model1_epoch2.pth
|           ├── ...
|         ├── plots/                ## 性能图
|           ├── plot_epoch1.jpg        ## 各个epoch
|           ├── plot_epoch2.jpg
|           ├── ....
|         ├── results/                  ## 预测值csv
|           ├── res_epoch1.csv         ## 单个epoch
|           ├── res_epoch2.csv
```

```
|           ├── ....
|           ├── res_config.txt        ## 汇总训练参数
|           ├── res_log.csv            ## 汇总训练结果指标
|           ├── trial_name.yaml        ## 实验任务yaml的复制
|   ├── 20211010/
|   ├── ...
|
├── run.py           ## 入口文件
├── exec.so           ## 依赖文件
└── README.md
```

开发：models

按输入分单输入，多输入，信号合成（combine)等开发任务。以单输入为例：

\1. 在models/single_input/下新建模型文件，如FC_DNN.py
\2. 在FC_DNN.py里编写模型class，如下图中的fcDnn_1

models > single_input > FC_DNN.py > ...

```python
from models.BasicModule import BasicModule
import torch.nn as nn
import torch

class fcDnn_1(BasicModule):
    '''
    Fully Connected Deep Neural Network
    '''
    def __init__(self,input_dim=482,drop_rate=0.5):
        super(fcDnn_1,self).__init__()
        self.model_name = 'fcDnn_1'
        self.fc_dnn_layer = nn.Sequential(nn.Linear(input_dim, 300),nn.ReLU(),nn.Dropout(drop_rate),
                                          nn.Linear(300,150),nn.ReLU(),nn.Dropout(drop_rate),
                                          nn.Linear(150,50),nn.ReLU(),nn.Dropout(drop_rate))

        self.end_layer = nn.Linear(50,1,bias=False)

    def forward(self,x):
        x_signal = self.fc_dnn_layer(x[:,-1,:])
        x = self.end_layer(x_signal)
        self.model_out['y_pred'] = x
        self.model_out['signals'] = x_signal
        return self.model_out
```

(编写模型class)

\3. 将模型绑定到__ init __.py里

models > single_input > __init__.py

```python
from .FC_DNN import fcDnn_1,fcDnnCp_2,fcDnnFewSignalsCp_2
```

(绑定模型到模块中)

几个注意点：

- 模型类继承BasicModule，输出self.model_out，即更新基类里的字典

```python
models > BasicModule.py > ...
import torch
import time
import torch.nn as nn

class BasicModule(nn.Module):
    def __init__(self):
        super(BasicModule,self).__init__()
        self.model_name = str(type(self)) # 默认名字
        self.alpha_idx = None
        self.model_out = {'y_pred':None,'signals':None, 'multilayers':None}  ## 基本输出
```

- y_pred（必需）：输出的预测值，形状一般为（batch_size,1)**
- signals（必需）: 倒数第二层的信号，形状一般为（batch_size,n), n为信号个数。**
- multilayers: 用于储存输出中间层信号

```python
    def forward(self,x):
        x_signal = self.fc_dnn_layer(x[:,-1,:])
        x = self.end_layer(x_signal)
        self.model_out['y_pred'] = x
        self.model_out['signals'] = x_signal
        return self.model_out
```

- 命名规则：驼峰_输出标签

- 驼峰命名模型名
- 标签为1，2，3，分别对应单输出、两层输出，多层输出

- self.model_name：模型名称，**为了打印对照方便，应和self.model_name类名一样**
- 绑定模型到当前folder的__ init __.py里，比如single_input下的模型就绑定到single_input下的init里

```python
models > single_input > __init__.py
from .FC_DNN import fcDnn_1,fcDnnCp_2,fcDnnFewSignalsCp_2
```

配置：cfg.yaml

配置任务参数文件cfg.yaml

1. GPU配置:

```yaml
## ------------------------------------------------------------
## GPU
CUDA_VISIBLE_DEVICES: 0,1,2,3,4,5
cuda_device: # 调用的GPU编号
  - 0
```

- 'cuda_device': 调用的显卡编号。（$ watch -n 3 -d nvidia-smi 查看显卡资源，合理分配显存和内存。详细自行百度。）
- 可支持多卡（考虑到服务器负荷，不能超过3张，2张比较保险）：

2. 数据配置:

- ```yaml
  ## ------------------------------------------------------------
  ### 数据
  x_root:
    src1:
      - wp4/
      - wp7/
  y_root:
    y_train: y_train/
    y_eval: y_eval/

  id_train: 1,2,3,4,5,6,7,8,9,10
  id_test: 11,12,13,14

  # id_train: '100'
  # id_test: '100'

  label_train_names:
    - y15
    # - y12
    # - y13
  label_eval_names:
    - eval0
    - bk0
  label_train_loc:
    - 0
  label_eval_loc:
    - 0
  label_test_loc:
    - 1
  label_benchmark_loc: -1
  id_val: '100'
  val_subsample: 'same'
  ## ------------------------------------------------------------
  ```

- id_train, id_val, id_test为数据编号。训练集一般为1-10，测试集为11-14，验证集设置默认'100'
  - 100为默认的dummy小样本，可用来快速debug模型
  - 验证集为训练集的10%抽样数据，和训练集同分布，用来判断模型样本内过拟合程度
- 其它参数由项目负责人确定好，默认不动

3. 模型，损失函数，训练参数:
  ```

```yaml
## 模型
model: fcDnn_1
model_args:
  input_dim: 470
  # head_num: 4

## ------------------------------------------------------------
## 损失函数
loss_name: mseLoss_1
loss_args:
  # y_scale: 1
  # y_mean: 0

## ------------------------------------------------------------
## 训练
trainer: SI
lr: &lr 0.001
batch_size: 16           # minibatch大小
max_epoch: 200            # 最大epoch
ES_patience: 10
ISIC_limit: 0.90
pretrain_model_path:      # 迁移训练
load_model_path:         # 断点训练
```

模型配置

- model: 模型类名称（对应之前__ init __.py的模型import）
- model_args: 可以对模型超参（__ init __参数）进行配置。例如：对于fcDnn_1模型，可以配置yaml字典：

  `model_args:`

  `input_dim: 470`

  `drop_rate: 0.5`
- 从loss_lib里调用适当的损失函数。回归任务常用mse。其它为自定义备选损失函数。
  - loss_args为loss_lib的参数(如有），配置方式同model_args
- 考虑batch_size大小对训练效果、显存容量的影响

4. 优化器、学习器

```yaml
# ----------- 学习率scheduler -----------

use_WarmUp: true
Warmup_step: 4148
Warmup_multiplier: 20
sch_verbose: false

sch_name: 'CyclicLR'
sch_args:
  base_lr: 0.0001
  max_lr:  0.001
  # step_size_up: 3444
  # step_size_up: 3444
  step_size_up: 1036
  step_size_down: 1036
  cycle_momentum: false

# sch_name: 'StepLR'
# sch_args:
#   step_size: 100
#   gamma: 0.9
#   verbose: true

# --------------- 优化器 ----------------
use_SAM: true

# optim_name: SGD
# optim_args:
#   lr: *lr
#   momentum: 0.9
#   weight_decay: 0.0001

optim_name: Adam
optim_args:
  lr: *lr
  weight_decay: 0.0001

```

(学习率和优化器配置)

- 学习率scheduler：配置sch_name和sch_args
  - warmup为模型预热阶段，可调预热学习率增大倍数，预热batch数量等
- 优化器scheduler：配置optim_name和optim_args
  - SAM为抑制陡峭收敛算法，消耗两倍推理时间
- 考虑optimizer特性和lr学习率的搭配效果
- 默认配置为SGD + 固定学习率（如果配置optim和sch参数）

5. 输出：

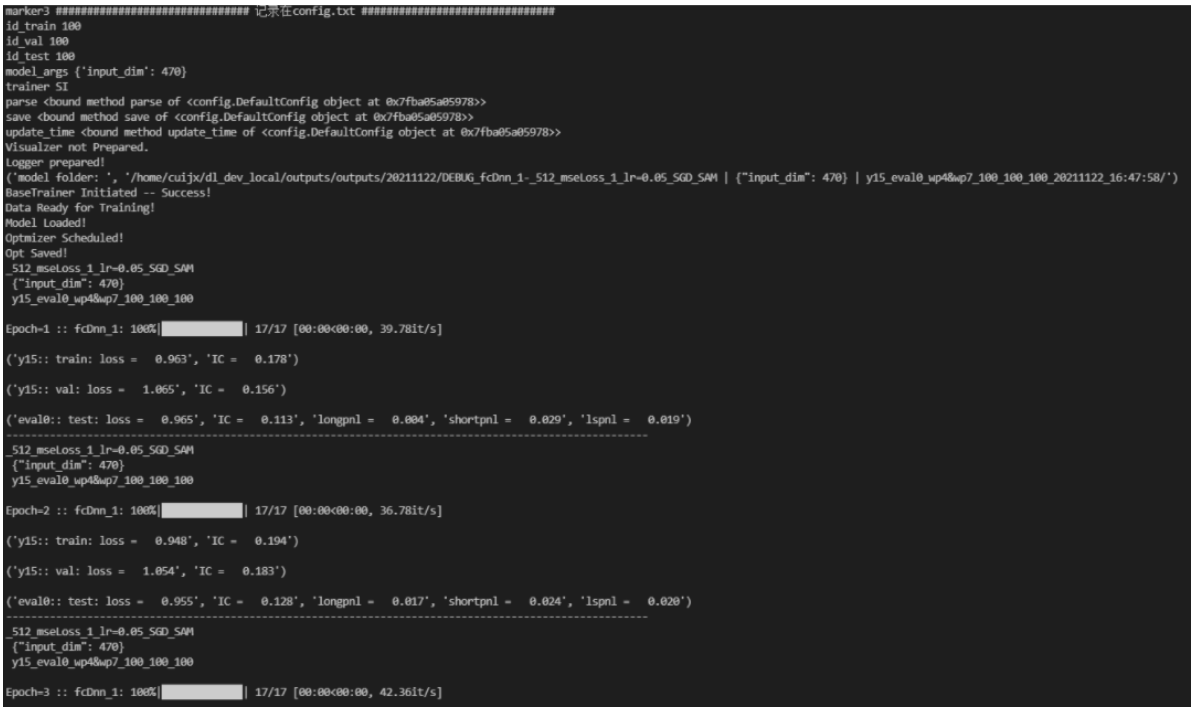
```yaml
## 输出
env: ''                    # 任务标签
debug_mode: false    # 仅用于debug，无输出
vis_plot: false
full_info: false
workspace: '/home/cuijx/dl_dev_local/
```

- env: 实验的额外标签，可以记录任务的目标、特点等等。
- debug_mode: true时没有output输出，用于调试（id_train==100时自动进入debug模式）
- vis_plot：visdom画图模型，暂不使用
- full_info：输出文件名带有配置信息，一般为false，在env里写note信息方便自己辨认
- workspace: output的输出路径，需要手动调整为本地workspace

实验：python run.py

\1. terminal 激活环境
\2. 运行 python run.py {your_yaml_path}

- 进入“读数据、读参数、开始实验”的步骤
- terminal会打印配置文件，打印提示信息



\3. 训练中：

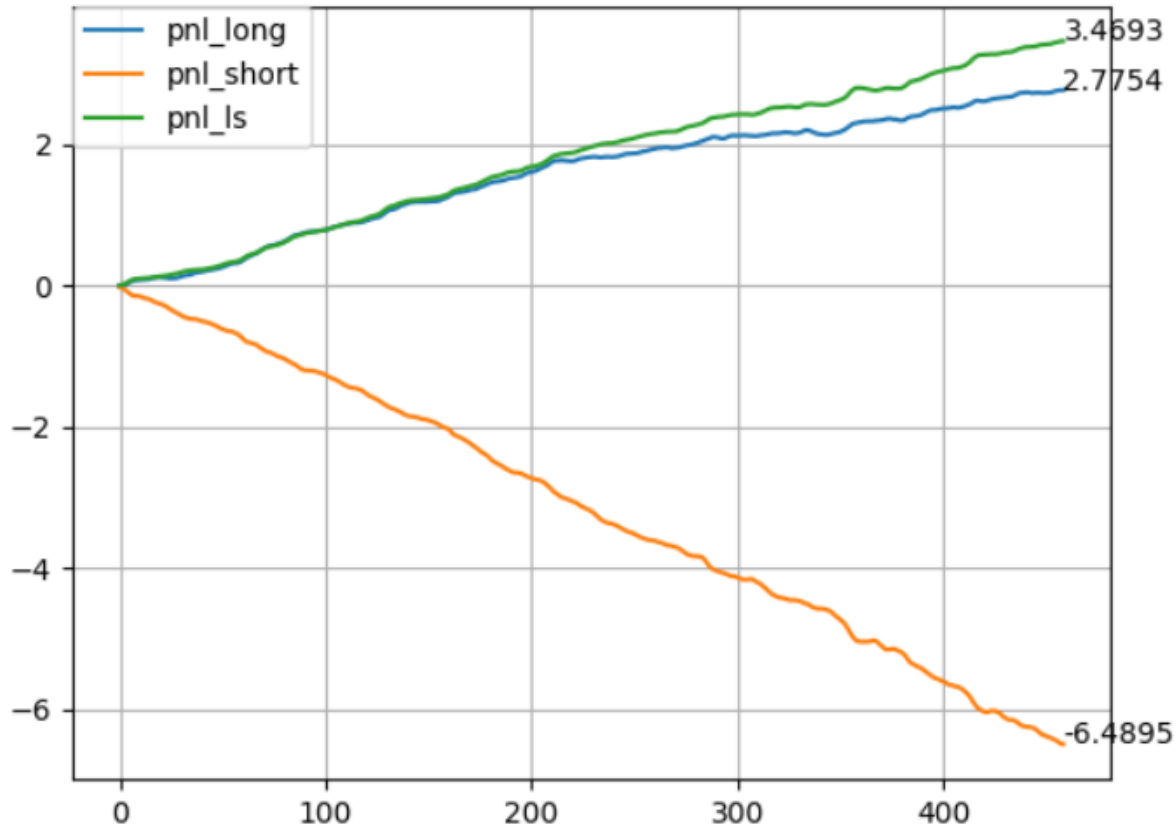
- 每个epoch计算并打印训练集验证集性能：loss, IC
- 每个epoch计算并打印样本外性能：Loss, IC, shortPnL, longPnL, lsPnL
  - IC：预测值和真实值的IC
  - shortPnL: short空头部分的累计收益
  - longPnL: long多头部分的累计收益
  - lsPnL: 多空累计收益
- 每个epoch存储模型文件、性能图、结果文件

\4. Early Stop: 程序会根据样本内外的指标判断要不要停止训练

- 样本内IC，如果太高（如0.5）则有过拟合风险，停止训练
- 样本外的性能增长曲线，如果几个epoch都不变好（具体几个用参数ES_patience控制），则停止训练

\5. 评价模型：

- 单模型最大性能：lsPnL（主要）, longPnL, shortPnL
- 单模型累计收益走势：看图



- 单模型增量信息：测试结果文件和其它模型的相关性
  - 可以通过结果文件.csv的y_pred实现

config额外字段参考

字段名	类型	含义	用法
full_info	bool	是否保存超参至文件名	文件名过长时设为false，在env里设置备注信息
env	str	文件名的备注信息	
clip_grad	bool	是否采用梯度截断	
use_SAM	bool	是否采用SAM逻辑	sharpness aware minimization, 设置为true时必须要显式定义optimizer。增加一倍推理时间
ES_patience	int	early stop的累计值	
finetune	bool	是否finetune	true时不进行early stop
ISIC_limit	float	样本内IC的上限	样本内IC达到这个值时，认为已经过拟合，触发early stop。
user_defined	bool	是否使用自定义组件（loss, optimizer, scheduler)	默认为false，默认调用yaml关键字绑定的组件 true时，调用model.py里定义的loss, optimizer, scheduler

x_slc	dir	字典，用于读取X的切片。传入start,end,step三个整数，为python内置Slice(start,end,step)函数的三个参数。	x_src:
			  src1:
			  - raw3/
			  src2:
			  - raw5/
			  src3:
			  - wp4/
			  - wp7/
			
			x_slc:
			  src1: ### 空值传入None，表示全部读入
			  src2:
			  - 0,490,49 ### 表示raw5每49根bar取一根
			  src3:
			  - 19,20,1
			  - 19,20,1 ###表示wp47各取最后一天