软件全称：迭代嵌入式原子神经网络软件

软件简称：REANN

软件版本：V1.0

用户手册

目录

第一章 软件概况

1.1 软件环境

1. PyTorch 1.9.0 版本及其依赖的的环境包括且不限 python, numpy, mkl, cuda 等等；
2. LibTorch 版本大于等于 1.9.0；
3. opt_einsum 库版本要求大于等于 3.2.0；
4. cmake 版本大于等于 3.1.0。

1.2 编程语言

1. Python；
2. C++14。

1.3 源程序量

代码行数（包括注释）：3044。

第二章 软件功能简介

迭代嵌入式原子神经网络软件（REANN）是用 Python 和 C++语言实现的机器学习软件。可以高效且精确地表示分子或者周期性体系的势能，永久/跃迁偶极矩以及极化率。整个软件是基于脸书公司（Facebook）开发的 Torch 框架，利用 Torch 框架的自动微分技术（autograd），分布式并行处理，REANN 可以利用多个节点 CPU/GPU 进行高效的分布式运行。REANN 已经接入动力学模拟软件 LAMMPS，得益于 LAMMPS 优秀的并行机制，训练好的势函数模型可以方便的进行大规模高效的动力学模拟。

2.1 源程序简介

REANN 设计为拟合动力学模拟过程中所需物理化学性质的软件，加速动力学模拟进程。程序主要分成两个部分，一个是训练模块： run 和 src 文件夹，

1. src 中保存程序的主要组成模块，包括程序以及数据初始化，数据加载，嵌入式密度模块，网络结构模块, 物理化学性质计算模块，训练流程模块；
2. run 文件夹中包含一个 train.py 文件，将各个模块组织起来进行训练。

训练好的模型可以用于预测给定分子结构的各种物理化学性质。这个就是程序的第二部分，将训练好的模型保存为一个序列化的，可优化的，不依赖具体语言环境的序列化文件，因此可以方便的做推理预测以及和各种动力学模拟软件连接，进行相应的动力学模拟。这一部分包含 5 个文件夹：pes, lammps, lammps_REANN, dm, tdm, pol。每个文件夹的主体程序相似，都包含 PES 模块，嵌入式密度和网络结构模块。5 个文件夹分别对应这势能/原子力，势能/原子力（LAMMPS 接口调用），永久偶极矩，跃迁偶极矩，极化率。除了这两个部分外，还有一个 lammps-interface/lammps-REANN-interface 文件夹存放 REANN 接入 LAMMPS 的接口程序。

2.2 输入文件准备

运行 REANN 需要准备两部分文件，一个是包括各种分子或者材料的坐标以及需要表示的各种物理化学性质的结构文件；另一个是用于定义模型以及控制优化过程各种超参数的参数文件，需放置在工作目录下的 para 文件夹中。

1. 数据文件

数据文件夹中包含两个文件夹 train 和 test,分别存放训练集和测试集，测试集用于在训练工程中评估模型精度。两个文件夹中都包含一个具有相同格式的 configuration 文件，下面以甲烷分子为例，文件格式如下：

```
point=     338
100e0   0e0    0e0
0e0   100e0    0e0
0e0   0e0    100e0
pbc  0      0      0
C   12.001     0.00000000      0.00000000      0.00000000      -14.86759064722606      5.82746293920830      18.44949170592880
H   1.008      0.21530327      0.94007371      -2.63454441      0.25580893795267      0.10459715129991      1.17136786972148
H   1.008      -1.75292964      -1.78556053      -0.78100346      0.93859602638230      1.12813629341912      0.29337534301938
H   1.008      0.48285430      -0.16291253      -0.63569389      15.02151392048367      -5.98470653657124      -19.66531191227881
H   1.008      1.40033763      1.82991110      -0.63015415      -1.34869539128798      -1.07538906006715      -0.24900631017049
abprop:    -1082.8707784962112
point=     768
100e0   0e0    0e0
0e0   100e0    0e0
0e0   0e0    100e0
pbc  0      0      0
C   12.001      0.00000000      0.00000000      0.00000000      -0.56097587955648      1.26550114070242      -2.68632166430252
H   1.008      0.60193377      0.47856424      2.48396016      -0.38594846790529      0.05645630842891      -1.11651078817310
H   1.008      -0.11501581      0.04597494      1.07368985      0.61504157519838      -0.93593596186365      3.33340125608123
H   1.008      1.60641275      -0.76736322      -1.42335407      -1.26645604884298      0.93123598318581      1.39934923151112
H   1.008      -1.32167745      1.60357274      0.57388184      1.59837944455446      -1.31720296304213      -0.92990363693260
abprop:    -1086.5145899630843
```

图里只展示了包含两个甲烷分子的数据文件，起始行只是作为一个标识，可以是任意的字符，2~4 行是体系的周期性的晶格参数，用于定义体系周期性边界，对于没有周期性的分子体系或者团簇等等可以是数值精度允许的浮点数。第 5 行的 pbc 为 periodic boundary condition 的缩写，用于确定分子在晶格方向是否有周期性其中 0 代表假，1 代表真。本例子中甲烷分子是一个非周期性体系，因此在三个方向都为 0。接下来的每一行都对应这分子中每一个原子的信息。其中第一列是原子对应的元素符号，第二列是对应元素的相对原子质量，3~5 列是各个原子在 x/y/z 方向上的坐标。接下来三列表示每个原子对应坐标方向上的原子力，这三列只有构建势能函数时引入体系原子力的信息时才是必要的。对于拟合其他物理化学性质，这三列则不需要。最后一行时第一个字符串是“abprop:” 紧接着你要拟合的物理化学性质（能量，偶极矩，极化率）。

2. 参数文件

REANN 软件需要两个必要的参数文件，分别时确定网络结构和控制拟合流程的 input_nn 以及用于决定嵌入密度的超参数文件 input_density。input_density 和 input_nn 采用相同的格式即变量名等于给定值，下面是一个甲烷的的例子。

input_density:

```
neigh_atoms=5
cutoff=6.5e0
nipsin=[0, 1, 2]
atomtype=['C', 'H']
nwave=16
```

(1) Cutoff：代表截断半径，原子间距离大于这个值得得相互作用将不会被忽略；

(2) neigh_atoms: 0 表示每个中心原子截断半径内最多包含的邻近原子数；

(3) atomtype: 表示体系所包含的化学元素列表；

(4) nipsin: 计算嵌入式密度用到的角动量，图中 1, 2, 3 分别对应着 s, p, d … 原子轨道；

(5) nwave: 表示径向的高斯轨道数量，用于确定高斯轨道形状和位置的超参数 α 和 r_s 将会在训练过程中和神经网络参数一起优化。

input_nn:

input_nn 中参数主要分成两部分，第一部分是整体训练过程的超参数以及搭建嵌入密度到原子能量的之间映射关系的神经网络，第二部分则是构建生成迭代嵌入密度的神经网络结构的参数，这一部分参数全部以“oc”开头以区别与其他参数。下面先介绍第一部分参数

```
start_table = 1                 # start_table table for the fit with force(1) or without force(0)
                                # start_table table for DM(2), TDM(3), polarizability(4)
table_coor = 0                  # table_coor   0 for cartesian coordinates and 1 for direct coordinates
nl = [128,128]                  # neural network architecture
nblock =  2
dropout_p=[0.0,0.0,0.0,0.0]
table_init = 1                  # 1 used for load parameters from pth
nkpoint=1                       # number of nkpoint NNs was employed to representation polarizability
Epoch=50000                     # max iterations epoch
patience_epoch = 500            # pre initial learning rate epoch
decay_factor = 0.8              # Factor by which the learning rate will be reduced. new_lr = lr * factor.
start_lr = 0.001                # initial learning rate
end_lr =1e-5                    # final learning rate
re_ceff = 0                     # factor for regularization
ratio = 0.9                     # ratio for training
batchsize = 64                  # batch_size
e_ceff=0.1                      # weight for energy
init_f=0.05                     # init_f
final_f=0.05                    # final_f
activate = 'SiLU'               # default "Softplus", optional "Gelu", "tanh", "SiLU"
queue_size=20                   # queue_size sequence for laod data into gpu
print_epoch=10
table_norm=True
DDP_backend='nccl'
dtype="float64"
floder="/home/home/zyl/pytorch/2021_1_7/data/ch4/data_1e4/"
```

(1) start_table: 可以是 0, 1, 2, 3, 4。0/1/都是表示构建势能函数其中 0 表示只是用能量作为目标训练，而 1 表示引入原子力训练势能函数；2/3/4 表示搭建对称性适配的永久偶极矩/跃迁偶极矩/极化率机器学习模型

(2) table_coor: 取 0 说明数据文件里面的坐标时直角坐标，1 指代分数坐标；

(3) table_init: 取 0 代表使用默认初始化的参数开始新的训练；1 表示从保存在 EANN.pth 中的断点或者预训练模型继续训练；

(4) nblock: nblock>1 时代表残差神经网络模块的个数，nblock=1 表示使用普通的全连接层神经网络模块；

(5) nl: 是一个整数列表表示表示每个神经网络模块的隐藏层神经元的个数；

(6) dropout_p: 是一个实数列表，列表的元素数量与神经网络隐藏层数量相同，且每一个值分别代表这对应的隐藏层 dropout 的几率；

(7) table_norm: 是一个布尔值，确定是否使用层归一化（layer normalization）；

(8) re_ceff: 是 L2 正则化系数；

(9) Epoch: 训练最多进行的完整的迭代 epoch 数量，每次 epoch 需要遍历整个数据集；

(10) start_lr: 初始学习率；

(11) end_lr：终止学习率，如果学习率衰减到 end_lr，训练将会终止；

(12) decay_factor: 学习率的衰减因子；

(13) patience_epoch: 等待的 Epoch 数，如果迭代 patience_epoch 次 epoch 模型的损失函数没有下降，就将学习率乘以 decay_factor，衰减学习率；

(14) print_epoch: 每隔 print_epoch，计算模型的训练以及测试误差并且打印到 nn.err 输出文件里面；

(15) Batchsize: 在梯度下降步（step）中使用的构型数目；

(16) e_ceff: 构建势能函数模型，损失函数中能量的权重；

(17) init_f: 误差函数中原子力权重的初始值；

(18) final_f: 误差函数中原子力权重的终止值；

(19) Activate: 原子神经网络的激活函数; 可选项 Gelu/tanh/Softplus/SiLU

(20) queue_size: 使用 GPU 训练，线程预加载的批（batch）数量;

(21) floder: 数据文件的路径；

(22) dtype: 数据类型 可选项 float32/float64；

(23) DDP_backend: 分布式并行的后端，对于 GPU 并行使用 nccl， CPU 并行只能用 gloo。

下面的参数代表构建迭代嵌入式密度的神经网络参数：

```
oc_nl = [128,128]          # neural network architecture
oc_nblock = 2
oc_dropout_p=[0,0,0,0]
oc_activate = 'SiLU'          # default "Softplus", optional "Gelu", "tanh","SiLU"
oc_table_norm=True
oc_loop=2
```

这里面大部分参数都和前面介绍的参数有着相似的参数名和相同的含义，只是在前面加上 oc_的前缀表示这一部分适用于构建迭代的嵌入式密度。除了 oc_loop 是一个新的参数表示得到最终的嵌入式密度迭代的次数。

2.3 软件运行

准备好数据文件和参数文件，安装好需要的软件环境，我们就可以使用 REANN 软件进行训练，启动前我们需要在工作目录下新建一个 para 文件夹存放

两个参数文件，接着在命令行或者调度系统提交脚本中使用如下的语句启动 REANN:

```
group_zyl@mgt ~]$ python3 -m torch.distributed.run --nproc_per_node=2 --standalone $path
```

其中

--nproc_per_node 表示每个节点启动进程数量；

--standalone 表示是单节点多进程的方式；

--nnodes：指定使用的计算节点的数目；

$path 表示源程序的存放路径；

当我们进行多机多卡训练时还需要使用下面的参数额外指定一些其他信息

--rdzv_id：指定作业管理系统分配的作业 id;

--rdzv_endpoint：指定主节点地址以及通信端口；

以及指定 rdzv_backend 为 c10d。

2.4 输出文件解释

启动程序后，会有多个文件产生，以构建势能函数为例，最后将会产生 nn.err,, EANN.pth, EANN_PES_DOUBLE.pt, EANN_PES_FLOAT.pt, EANN_LAMMPS_DOUBLE.pt 和 EANN_LAMMPS_FLOAT.pt 6 个输出文件。

其中 nn.err 如下图

分别打出 Epoch, learning rate, 能量/原子间力的训练和测试误差。

```
EANN package
2021-01-17-14_21_47
Epoch=    0 learning rate   1.000000e-03 train error:   11.63031    9.00570 test error:   11.32739    8.96940
Epoch=   10 learning rate   1.000000e-03 train error:    1.82564    3.68674 test error:    1.63228    3.71116
Epoch=   20 learning rate   1.000000e-03 train error:    2.49021    2.77032 test error:    2.10424    2.72323
Epoch=   30 learning rate   1.000000e-03 train error:   26.58009    2.58143 test error:   26.50890    2.53251
Epoch=   40 learning rate   1.000000e-03 train error:    8.13302    2.01502 test error:    8.29458    1.98767
Epoch=   50 learning rate   1.000000e-03 train error:    6.84637    1.87493 test error:    7.02975    1.85310
Epoch=   60 learning rate   1.000000e-03 train error:    9.79246    1.76052 test error:    9.94749    1.75381
Epoch=   70 learning rate   1.000000e-03 train error:   10.20592    1.68688 test error:   10.27028    1.67965
Epoch=   80 learning rate   1.000000e-03 train error:    0.72603    1.74120 test error:    0.73186    1.75364
Epoch=   90 learning rate   1.000000e-03 train error:    3.18237    1.58636 test error:    3.29170    1.56910
Epoch=  100 learning rate   1.000000e-03 train error:   10.43401    1.56382 test error:   10.42914    1.54899
Epoch=  110 learning rate   1.000000e-03 train error:   15.46383    1.52625 test error:   15.31574    1.51574
Epoch=  120 learning rate   1.000000e-03 train error:    2.32351    1.38274 test error:    2.15801    1.37255
Epoch=  130 learning rate   1.000000e-03 train error:    1.91815    1.26751 test error:    1.66884    1.26249
Epoch=  140 learning rate   1.000000e-03 train error:    2.51013    1.37732 test error:    2.36825    1.36297
Epoch=  150 learning rate   1.000000e-03 train error:    2.92531    1.21014 test error:    2.74585    1.20664
Epoch=  160 learning rate   1.000000e-03 train error:    1.11917    1.23829 test error:    1.01199    1.23395
```

EANN.pth 则会保存损失函数最小的那一步对应的模型参数，用作断点，在程序不正常退出时，可以通过保存的断点继续拟合，可以节省训练时间。

EANN_PES_DOUBLE.pt/EANN_PES_FLOAT.pt 是一个通用的势能函数模型序列化文件，可以在多种环境中加载。通过传入计算势能所需的结构信息（坐标，元素，周期性边界条件）得到势能和原子间力。可以方便的嵌入到各种动力学模

拟软件，进行动力学模拟。

EANN_LAMMPS_DOUBLE.pt/EANN_LAMMPS_FLOAT.pt 只有在构建势能函数是才会保存的模型文件。是针对动力学模拟软件 LAMMPS，单独进行优化和保存的序列化模型相对于通用的势能函数模型，EANN_LAMMPS_DOUBLE.pt/EANN_LAMMPS_FLOAT.pt 无需进行周期性边界条件的处理，因此更加高效。更多详细的信息在下一章节 LAMMPS 接口。

第三章 LAMMPS 接口

前两章介绍了如何使用 REANN 软件训练势能函数,偶极矩以及极化率。这一章，我们将集中在使用训练好的势能函数模型进行动力学模拟。我们将具体介绍如何将 REANN 和被广泛使用的动力学模拟软件结合，进行高效且精确的动力学模拟。需要说明的是由于和 LAMMPS 并行机理的冲突，这一部分功能对于参数有一些限制，当参数 oc_loop 大于 0，使用 LAMMPS 进行动力学模拟，只能使用单进程进行模拟。

为了能够充分的利用 LAMMPS 优秀的并行设计，我们选择了将 REANN 作为一个新的 pair_style 类，从源码直接接入 LAMMPS。在进行动力学模拟前，我们需要将在软件环境这一届准备的 LibTorch 源码拷贝到 LAMMPS 源码目录下，并且将 REANN 源码目录下 lammps-interface 文件夹中的 pair_eann.cpp 及其头文件 pair_eann.h 拷贝到 LAMMPS 目录下的 src 文件夹中，一切准备就绪后，在 LAMMPS 目录下，build 文件夹下命令行执行

```
group_zyl@mgt /home/home/zyl/lammps/lammps-master/build]$ cmake -D BUILD_MPI=ON -D LAMMPS_MACHINE=mpi ../cmake
```

得到 makefile，然后通过”make”命令将 LibTorch 以一个动态链接库的方式和 LAMMPS 源码一起编译，得到最后可执行文件 lmp_mpi。在具体使用 LAMMPS 时需要在 LAMMPS 输入文件中指定 pair_style 指定为 EANN 即可。需要注意的是在 LAMMPS 的输入文件中，指定元素的顺序要与 input_density 中 atom_type 列表中元素顺序相同（元素的序号要从 1 开始）。