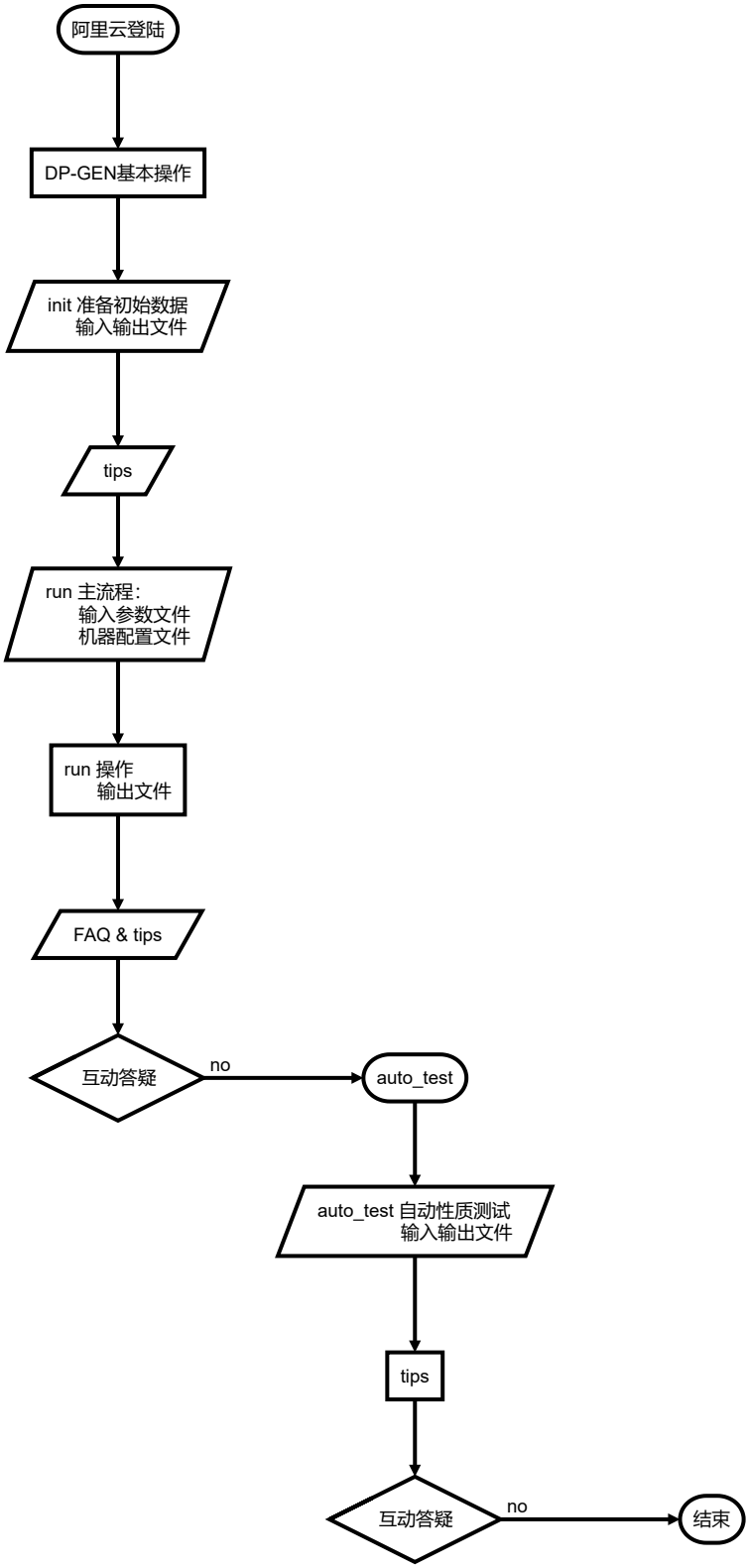
DP-GEN上手

1. 力场构建 (init & run) | 甲烷
2. 性质测试 (auto_test) | 铝



声明:

出于版权考虑，例子文件未提供 VASP 计算所需的 POTCAR 文件，请自行准备相关文件。

上手实例基于阿里云弹性计算服务实现

提要

概述

本上手教程旨在帮助您快速熟悉`DP-GEN`的实际操作。

`DP-GEN`的基本工作流程为`init` → `run` → `auto_test`，
分别对应`调用第一性原理计算软件产生初始数据` → `调用 DeePMD-kit 及动力学模拟软件，实现同步学习策略，循环迭代自动扩充数据集并提升模型质量` → `应用 DP 模型、第一性原理方法及经验立场，调用软件计算参考体系的多种性质`。

本教程以 CH_4 分子为实例演示`init` & `run`，以 AI 为实例演示`auto_test`，调用`VASP`作为第一性原理计算软件，`LAMMPS`作为分子动力学模拟软件。

为简化操作：
`init` 只介绍输入参数文件的与输出文件的结构，
`run` 将基于已经准备好的初始数据(CH_4 AIMD in $50K$)进行操作，
`auto_test` 将使用已训练好的 AI 模型进行操作。
全流程详细说明请参考(https://github.com/deepmodeling/dpgen)及以往教程。

准备

阿里云登陆

本次操作的 DP-GEN 阿里云解决方案：

- root@分配的阿里云节点→本地计算+使用统一的阿里云API授权账户向远程计算节点提交任务。
- 远程计算任务执行流
  程：`DP-GEN 根据所提交任务的计算需求，征用相应数量的 GPU 远程计算节点`，`上传计算任务文件至计算节点并开始计算`，`远程计算完成后，下载结果文件，删除计`

root@分配的阿里云节点

```
ssh root@IP
```

通过 Anaconda 激活节点上 DP-GEN 运行所需的环境

```
conda activate ali-dpgen
```

Windows 操作可使用自带的 PowerShell 或 CMD 命令窗口(win键+R 输入powershell或cmd)，Xshell 等终端模拟软件，Linux 子程序(Win10)等。

本次操作使用的阿里云机器已经部署了`DeePMD-kit`，`DP-GEN`，`dpdata` 等 `deepmodeling` 软件及运行所需的其他软件与环境。`deepmodeling` 相关软件的安装请参考(https://github.com/deepmodeling/)，本教程最后 FAQ & tips 部分也提供了一种 DP-GEN 开发版的简便安装方法。

DP-GEN基本操作

查看 DP-GEN 支持的任务类型(sub-command)：

```
dpgen -h
```

```
Description
------------
usage: dpgen [-h]
             {init_surf,init_bulk,auto_gen_param,init_reaction,run,run/report,simplify,test,db}
             ...

dpgen is a convenient script that uses DeepGenerator to prepare initial data,
drive DeepMDkit and analyze results. This script works based on several sub-
commands with their own options. To see the options for the sub-commands, type
"dpgen sub-command -h".

positional arguments:
  {init_surf,init_bulk,auto_gen_param,init_reaction,run,run/report,simplify,test,db}
    init_surf           Generating initial data for surface systems.
    init_bulk           Generating initial data for bulk systems.
    auto_gen_param      auto gen param.json
    init_reaction       Generating initial data for reactive systems.
    run                 Main process of Deep Potential Generator.
    run/report          Report the systems and the thermodynamic conditions of
                        the labeled frames.
    simplify            Simplify data.
    test                Auto-test for Deep Potential.
    db                  Collecting data from DP-GEN.

optional arguments:
  -h, --help            show this help message and exit
```

基本流程演示中将涉及的任务类型:

- `init_bulk`：产生体相系统的初始数据。
- `run`：Deep Generator主流程。
- `test`：auto_test自动性质测试。

对任一 `sub-command`，可以通过 `dpgen sub-command -h` 查看其用法，如：

```
dpgen init_bulk -h
```

```
Description
------------
usage: dpgen init_bulk [-h] PARAM [MACHINE]

positional arguments:
  PARAM       parameter file, json/yaml format
  MACHINE     machine file, json/yaml format

optional arguments:
  -h, --help  show this help message and exit
```

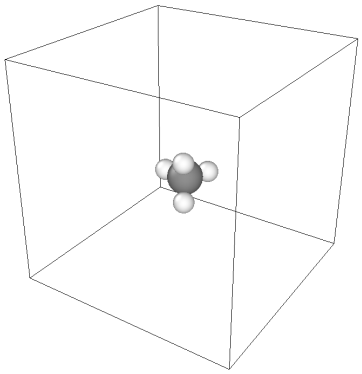
`dpgen`的基本用法即：

```
dpgen sub-command PARAM MACHINE
```

`PARAM` 位置代表输入参数文件，下文使用json格式文件，命名为 `param.json`，
`MACHINE` 位置代表机器配置文件(部分任务类型为可选文件)，下文使用json格式文件，根据 `run` 和 `test` 资源需求的不同，分别命名为 `machine-ali_local.json` 及 `machine_local.json`。

---

1 力场构建 | 甲烷



1.1 init 准备初始数据

```
dpgen init_bulk param.json [MACHINE]
```

`init_bulk` 的基本操作流程分为以下4个**顺序**步骤：

1. **生成初始(平衡)结构**：输出文件存储在 `00.place_ele`。
2. **产生MD起始构象**：放缩扰动 `00.place_ele` 中结构，输出文件存储在 `01.scale_pert`。
3. **执行AIMD**：取 `01.scale_pert` 起始构象调用 `vasp` 执行少步数AIMD，输出文件存储在 `02.md`。
4. **整理初始数据**：调用 `dpdata` 提取 `02.md` 中 AIMD 各 snapshot 数据作为初始数据，输出文件存储在 `02.md`。

`param.json` 中将使用 `stages` 关键词的取值1~4控制以上4步的执行， 操作中，各步骤(2, 3, 4在所需前一步输出文件存在的前提下)可独立执行。

提示 执行至步骤 2 产生的结构也可作为 `dpgen run` 使用 MD 探索训练样本空间时的起始构象，按需可修改相关参数(见下)，调整随机结构数量或超胞尺寸等。后续将看到， `dpgen run` 部分演示的例子，即将各体系起始构象所在的路径设置在 `/somwhere/01.scale_pert/` 下。

1.1.1 输入参数 param.json

CH_4 分子

```
{
      "stages" :          [1,2,3,4],
      "elements":         ["H","C"],
      "cell_type":        "diamond",
      "latt":             10.0,
      "super_cell":       [1, 1, 1],
      "from_poscar":      true,
      "from_poscar_path": "....../CH4.POSCAR",
      "potcars":          ["....../POTCAR_H",
                           "....../POTCAR_C"
                          ],
      "relax_incar":      "....../INCAR_rlx",
      "md_incar" :        "....../INCAR_md",
      "skip_relax":       false,
      "scale":            [1.00],
      "pert_numb":        30,
      "md_nstep" :        10,
      "pert_box":         0.03,
      "pert_atom":        0.01,
      "coll_ndata":       5000,
      "type_map" :        ["H","C"],
      "_comment":         "that's all"
}
```

- 在上面的 `param.json` 中，可以看到出现了两次指定元素的关键词，分别为 `elements` 和 `type_map`。这是因为,当我们需要为训练多元素系统（如 A-B-C）通用模型准备初始数据时，需要分别准备单质 A, B, C; 二元系统 A-B, B-C, A-C; 及三元系统 A-B-C 的数据。对每单次任务，我们通过 `elements` 的变化来指定本次任务使用哪种元素的参数生成数据，而在所有次任务中，都通过同一固定次序的三元素 `type_map` 列表，如：["A", "B", "C"]，将各元素映射到一个固定的 deepmd 格式序号上，如单质时：0，1，2，二元：[0，1]，[1,2] [0,2]; 三元：[0,1,2], 从而使 DeePMD-kit 能够根据这一对应关系，正确地将性质数据地映射到相应的元素上。
- **注意** 在准备多元素体系的初始数据时，请注意正确设置 `type_map`。

对 CH_4 孤立分子，我们通过POSCAR文件指定了 stages 1 的初始结构，此时， `cell_type` `latt` 关键词不生效。

对金属单质等体相体系，可以依据上述关键词及 `super_cell` 自动生成相应的晶体初始结构：

```
      "elements":         ["Mg"],
      "cell_type":        "hcp",
      "latt":             4.479,
      "super_cell":       [2, 2, 2],
      "from_poscar":      false,         <_<_<
```

对于合金等复杂体相体系及特定的应用场景，我们可以跳过 stages 1 的弛豫过程，直接使用自动生成的晶体结构或 POSCAR 中的指定结构进行 stages 2

```
      "skip_relax":       true,
```

下表列出了 `param.json` 中各关键词的详细说明：

其中粗体显示的关键词如 **Elements** 在所有情形下**必须赋值**。

关键词	数据结构	例子	描述
stages	List of Integer	[1,2,3,4]	指定 `init_bulk` 的执行步骤
elements	List of String	["H","C"]	指定元素种类
cell_type	String	"hcp"	指定 `stages 1` 生成结构的晶型. **选项**: fcc, hcp, bcc, sc, diamond.
latt	Float	4.479	指定 `stages 1` 生成结构的元胞晶格常数.
super_cell	List of Integer	[1,1,1]	指定 `stages 1` 生成结构的超胞.
from_poscar	Boolean	true	指定 `stages 1` 是否采用POSCAR文件中结构作为初始结构。 true: `cell_type` `latt` 失效， `from_poscar_path` **必须赋值** false: `cell_type` `latt` **必须赋值**。
from_poscar_path	String	"....../CH4.POSCAR"	指定 `stages 1` POSCAR 文件的路径。

关键词	数据结构	例子	描述
potcars	list of String	["...../POTCAR_H", "...../POTCAR_C"]	指定`stages 1` `stages 3` VASP计算所需 POTCAR 文件的路径.
relax_incar	String	"...../INCAR_rlx"	指定 `stages 1` 中 VASP 弛豫平衡结构所需 INCAR 文件的路径. `stages` 列表中包含 1 时：**必须赋值**(必要性不会受 `skip_relax` 赋值影响)
md_incar	String	"...../INCAR_md"	指定 `stages 3` VASP AIMD 所需 INCAR 文件的路径 `stages` 列表中包含 3 时：**必须赋值**
skip_relax	Boolean	False	指定 `stages 2` 是否使用 `stages 1` 中未弛豫的 POSCAR 结构做放缩扰动。
scale	List of float	[0.980, 1.000, 1.020, 1.040]	指定 `stages 2` 放缩 `stages 1` 结构超胞盒子大小的标量倍率(可执行多个放缩)。
pert_numb	Integer	30	指定 `stages 2` 对每个放缩后结构执行扰动(同时扰动晶格矢量和原子位置)的数量。
md_nstep	Integer	10	指定 `stage 3` AIMD 的步数。 **注意**：当赋值与 `md_incar` 指定文件中 `NSW(VASP控制MD步数的关键词)` 值不同时， `stages 3` 将使用 `NSW` 值.
pert_box	Float	0.03	指定 `stages 2` 扰动盒子矢量的比率，扰动值/原值。
pert_atom	Float	0.01	指定 `stages 2` 扰动原子位置的大小(Å)。
coll_ndata	Integer	5000	指定 `stages 4` 收集 `stages 3` 数据的最大数量。
type_map	List	["H", "C"]	指定 `stages 4` 整理训练数据时，按 deepmd 文件格式记录元素种类的顺序。 **注意** 顺序请与`elements` `potcars` 及 POSCAR 文件等处指定的元素顺序一致。

跳过 操作

```
dpgen init_bulk param.json [MACHINE]
```

实际操作中，当我们不提供 [MACHINE] 处的机器配置文件时, param.json 中的 `stages` 列表应只包含一个步骤对应的数值。此时，DP-GEN 将在相应输出文件夹中准备此步骤的输入脚本，用户需要手动提交脚本完成计算任务。

1.1.2 输出介绍

```
cd /root/workshop/dpgen-example/init
tree CH4.POSCAR.01x01x01/
```

```
CH4.POSCAR.01x01x01/
├── 00.place_ele
│   ├── INCAR
│   ├── jr.json
│   ├── POSCAR
│   ├── POSCAR.copied
│   ├── POTCAR
│   └── sys-0004-0001
│       ├── CONTCAR
│       ├── INCAR
│       ├── OUTCAR
│       ├── POSCAR
│       └── POTCAR
├── 01.scale_pert
│   └── sys-0004-0001
│       └── scale-1.000
│           ├── 000000
│           │   └── POSCAR
│           ├── 000001
│           │   └── POSCAR
│           ├── 000002
│           │   └── POSCAR
│           ├── 000003
│           │   └── POSCAR
│           ├── 000004
│           │   └── POSCAR
│           ├── 000005
│           │   └── POSCAR
│           ├── 000006
│           │   └── POSCAR
│           ├── 000007
│           │   └── POSCAR
│           ├── 000008
│           │   └── POSCAR
│           ├── 000009
│           │   └── POSCAR
│           ├── 000010
│           │   └── POSCAR
│           ├── 000011
│           │   └── POSCAR
│           ├── 000012
│           │   └── POSCAR
│           ├── 000013
│           │   └── POSCAR
│           ├── 000014
│           │   └── POSCAR
│           ├── 000015
│           │   └── POSCAR
│           ├── 000016
│           │   └── POSCAR
│           ├── 000017
│           │   └── POSCAR
│           ├── 000018
│           │   └── POSCAR
│           ├── 000019
│           │   └── POSCAR
│           ├── 000020
│           │   └── POSCAR
│           └── POSCAR
├── 02.md
│   ├── INCAR
│   ├── jr.json
│   ├── POTCAR
│   └── sys-0004-0001
│       ├── deepmd
│       │   ├── box.raw
│       │   ├── coord.raw
│       │   ├── energy.raw
│       │   ├── force.raw
│       │   ├── set.000
│       │   │   ├── box.npy
│       │   │   ├── coord.npy
│       │   │   ├── energy.npy
│       │   │   ├── force.npy
│       │   │   └── virial.npy
│       │   ├── type_map.raw
│       │   ├── type.raw
│       │   └── virial.raw
│       └── scale-1.000
│           ├── 000000
│           │   ├── INCAR
```

```
|           |   ├── OUTCAR
|           |   ├── POSCAR
|           |   └── POTCAR
|           ├── 000001
|           |   ├── INCAR
|           |   ├── OUTCAR
|           |   ├── POSCAR
|           |   └── POTCAR
|           ├── 000002
|           |   ├── INCAR
|           |   ├── OUTCAR
|           |   ├── POSCAR
|           |   └── POTCAR
|           ├── 000003
|           |   ├── INCAR
|           |   ├── OUTCAR
|           |   ├── POSCAR
|           |   └── POTCAR
|           ├── 000004
|           |   ├── INCAR
|           |   ├── OUTCAR
|           |   ├── POSCAR
|           |   └── POTCAR
|           ├── 000005
|           |   ├── INCAR
|           |   ├── OUTCAR
|           |   ├── POSCAR
|           |   └── POTCAR
|           ├── 000006
|           |   ├── INCAR
|           |   ├── OUTCAR
|           |   ├── POSCAR
|           |   └── POTCAR
|           ├── 000007
|           |   ├── INCAR
|           |   ├── OUTCAR
|           |   ├── POSCAR
|           |   └── POTCAR
|           ├── 000008
|           |   ├── INCAR
|           |   ├── OUTCAR
|           |   ├── POSCAR
|           |   └── POTCAR
|           ├── 000009
|           |   ├── INCAR
|           |   ├── OUTCAR
|           |   ├── POSCAR
|           |   └── POTCAR
|           ├── 000010
|           |   ├── INCAR
|           |   ├── OUTCAR
|           |   ├── POSCAR
|           |   └── POTCAR
|           ├── 000011
|           |   ├── INCAR
|           |   ├── OUTCAR
|           |   ├── POSCAR
|           |   └── POTCAR
|           ├── 000012
|           |   ├── INCAR
|           |   ├── OUTCAR
|           |   ├── POSCAR
|           |   └── POTCAR
|           ├── 000013
|           |   ├── INCAR
|           |   ├── OUTCAR
|           |   ├── POSCAR
|           |   └── POTCAR
|           ├── 000014
|           |   ├── INCAR
|           |   ├── OUTCAR
|           |   ├── POSCAR
|           |   └── POTCAR
|           ├── 000015
|           |   ├── INCAR
|           |   ├── OUTCAR
|           |   ├── POSCAR
|           |   └── POTCAR
|           ├── 000016
|           |   ├── INCAR
|           |   ├── OUTCAR
|           |   ├── POSCAR
```

```
│           │   └── POTCAR
│           ├── 000017
│           │   ├── INCAR
│           │   ├── OUTCAR
│           │   ├── POSCAR
│           │   └── POTCAR
│           ├── 000018
│           │   ├── INCAR
│           │   ├── OUTCAR
│           │   ├── POSCAR
│           │   └── POTCAR
│           └── 000019
│               ├── INCAR
│               ├── OUTCAR
│               ├── POSCAR
│               └── POTCAR
└── param.json
```

1.2 run 主流程

`run` 主流程包含一系列迭代，各迭代会根据所选系综在给定的温度、压力或体积等条件下进行 DPMD 采样与筛选，实现同步学习，所有迭代希望能够有效覆盖训练可靠模型所需的势能面上样本空间。

每个迭代依次 `基于 init 准备的初始训练数据与之前迭代积累的训练数据|训练多个新的模型` →
`使用当前的模型进行DPMD采样，根据模型间预测偏差(model-deviation)|挑选候选 snapshot 构象` →
`将本轮候选 snapshot 构象进行第一性原理标定并加入数据集|扩展数据集` 。

以上每个迭代中包含的 3 个步骤分别被命名为 `00.train` → `01.model_devi` → `02.fp`，细节介绍如下：

- `00.train`：使用 `init` 准备的初始训练数据与之前迭代积累的训练数据，调用 `DeepMD-kit` 训练多个 (默认 4 个) 模型。模型间的唯一区别来自于初始化神经网络时使用不同的随机数种子。
- `01.model_devi`：指代 model-deviation。 调用 `LAMMPS` 使用 `00.train` 的 1 个模型进行 MD 模拟。对于任一 MD 中 snapshot，模型间预测偏差越大意味着当前模型系综对该 snapshot 构象的精度越低，通过引入模型偏差作为误差判据并设定上下限, 挑选出有希望有效改进模型对PES整体预测精度的 snapshot 构象，作为准备加入训练数据集的候选构象。
- `02.fp`：调用 `VASP` 对 `01.model_devi` 选取的候选构象进行第一性原理定标(单点计算)，并调用 `dpdata` 收集整理所得数据加入到训练数据集中。

`run` 执行后，相关文件即存储在 `....../iter.*(迭代序号)/步骤同名文件夹` 内。我们可以进入为大家准备的 `run` 样例目录，通过事先执行完成的一个例子文件夹(...../run/previous_example)来浏览一下输出文件的结构：

```
cd /root/workshop/dpgen-example/run
ls previous_example/iter.00000*
```

```
previous_example/iter.000000:
00.train  01.model_devi  02.fp

previous_example/iter.000001:
00.train  01.model_devi  02.fp

previous_example/iter.000002:
00.train
```

以上每个迭代中的每个步骤在实现中被分解为 3 个阶段。以 `00.train` 为例，3 阶段分别对应:
0 `make_train`：为训练任务准备脚本， 1 `run_train`：依据机器配置上传文件并执行训练任务与 2 `post_train`：收集整理分析训练任务的结果。
`01.model_devi` 与 `02.fp` 的实现步骤与之类似，分别使用 3，4，5 及 6，7，8 来标记其阶段。

由此，我们可以将 `run` 主流程的执行过程分解表示如下：

迭代序号	各迭代的阶段序号	进程
0	0	make_train
0	1	run_train
0	2	post_train
0	3	make_model_devi
0	4	run_model_devi
0	5	post_model_devi
0	6	make_fp

迭代序号	各迭代的阶段序号	进程
0	7	run_fp
0	8	post_fp
1	0	make_train

使用该表示法，我们将使用输出文件 `record.dpgen` 来记录进程已经执行过及当前正在执行的阶段。
如进程中断后需要续算， `DP-GEN` 将根据此文件的记录来自动恢复 `run` 主进程。用户也可根据需求手动修改 `record.dpgen` 中的记录，来控制接下来将执行哪一阶段的任务：例如，删除最后一阶段(可能未执行完成即中断)的记录，并从前一个已完成阶段后恢复任务。

接下来，我们将使用已经准备好的 CH_4 分子初始数据来为大家演示，如何通过 `run` 主流程，生成和高效选取训练数据，改进模型预测精度，逼近对 CH_4 分子势能面的一致精确(uniformly accurate)。。

准备好的初始训练数据通过 $50K$ 条件下的 NVT 系综 AIMD 获得，存储在 /root/workshop/dpgen-example/run/CH4.POSCAR.01x01x01。
我们可以通过计数该 `体系` 初始数据文件如 `energy.raw` 的行数，来查看初始数据集所包含的构象个数，也即帧(frame)数：

```
wc -l CH4.POSCAR.01x01x01/02.md/sys-0004-0001/deepmd/energy.raw
```

```
200 CH4.POSCAR.01x01x01/02.md/sys-0004-0001/deepmd/energy.raw
```

共计 200 帧。

1.2.1 输入参数 param.json

`dpgen run PARAM MACHINE`

```
{
  "type_map": ["H", "C"],
  "mass_map": [1.0, 12.0],

  "_comment": "initial data set for Training and the number of frames in each training batch",
  "init_data_prefix": "/root/workshop/dpgen-example/run",
  "init_data_sys":   [
    "CH4.POSCAR.01x01x01/02.md/sys-0004-0001/deepmd"
  ],
  "init_batch_size": [
    8
  ],

  "_comment": "configurations for starting MD in Exploration and batch sizes when traning snapshots derived from these configs (if they were se
  "sys_configs_prefix": "/root/workshop/dpgen-example/run",
  "sys_configs":    [
    [
      "CH4.POSCAR.01x01x01/01.scale_pert/sys-0004-0001/scale-1.000/000000/POSCAR",
      "CH4.POSCAR.01x01x01/01.scale_pert/sys-0004-0001/scale-1.000/000001/POSCAR",
      "CH4.POSCAR.01x01x01/01.scale_pert/sys-0004-0001/scale-1.000/00000[2-9]/POSCAR"
    ],
    [
      "CH4.POSCAR.01x01x01/01.scale_pert/sys-0004-0001/scale-1.000/00001*/POSCAR"
    ]
  ],
  "sys_batch_size": [
    8,
    8
  ],

  "_comment": " 00.train ",
  "numb_models":            4,

  "default_training_param": {
    "model": {
      "type_map":           ["H","C"],
      "descriptor": {
        "type":             "se_a",
        "sel":              [16,4],
        "rcut_smth":        0.5,
        "rcut":             5.0,
        "neuron":           [10,20,40],
        "resnet_dt":        false,
        "axis_neuron":      12,
        "seed":             0
      },
      "fitting_net": {
        "neuron":           [120,120,120],
        "resnet_dt":        true,
        "coord_norm":       true,
        "type_fitting_net": false,
        "seed": 0
      }
    },
      "loss": {
        "start_pref_e":      0.02,
        "limit_pref_e":      2,
        "start_pref_f":      1000,
        "limit_pref_f":      1,
        "start_pref_v":      0,
        "limit_pref_v":      0
    },
      "learning_rate": {
        "type":             "exp",
        "start_lr":         0.001,
        "decay_steps":      180,
        "decay_rate":       0.95
    },
      "training": {
        "systems":          [],
        "set_prefix":       "set",
        "stop_batch":       36000,
        "batch_size":       1,
        "seed": 1,
        "_comment":         "frequencies counted in batch",
        "disp_file":        "lcurve.out",
        "disp_freq":        1000,
        "numb_test":        4,
        "save_freq":        1000,
        "save_ckpt":        "model.ckpt",
        "load_ckpt":        "model.ckpt",
```

```
        "disp_training":     true,
        "time_training":     true,
        "profiling":         false,
        "profiling_file":    "timeline.json"
    }
  },

  "_comment": " 01.model_devi ",
  "model_devi_dt": 0.002,
  "model_devi_skip": 0,
  "model_devi_f_trust_lo":   0.05,
  "model_devi_f_trust_hi":   0.15,
  "model_devi_clean_traj":   false,
  "model_devi_jobs": [
    {
      "sys_idx": [
        0
      ],
      "temps": [
        50
      ],
      "press": [
        1
      ],
      "trj_freq": 10,
      "nsteps":   1000,
      "ensemble": "nvt",
      "_idx":     "00"
    },
    {
      "sys_idx": [
        1
      ],
      "temps": [
        50
      ],
      "press": [
        1
      ],
      "trj_freq": 10,
      "nsteps":   3000,
      "ensemble": "nvt",
      "_idx":     "01"
    }
  ],

  "_comment": " 02.fp ",
  "fp_style":              "vasp",
  "shuffle_poscar":        false,
  "fp_task_max":           30,
  "fp_task_min":           8,
  "fp_pp_path":            "/root/workshop/dpgen-example/run",
  "fp_pp_files":           [ "POTCAR_H","POTCAR_C"],
  "fp_incar":              "INCAR_methane"
}
```

提示

上述参数中，分别指定初始和筛选出的训练数据在 1 个训练 batch 中帧数的 `init_batch_size` 与 `sys_batch_size` 关键词，会覆盖 `default_training_param` 中的 `batch_size` 值。

上例在 `model_devi` 过程中使用了 NVT 系综，所以 `press` 关键词并不会生效，我们可以通过查看 previous_example 中实际执行 MD 的 LAMMPS 脚本确认：

```
cat previous_example/iter.000000/01.model_devi/task.000.000000/input.lammps
```

```
variable        NSTEPS          equal 1000
variable        THERMO_FREQ     equal 10
variable        DUMP_FREQ       equal 10
variable        TEMP            equal 50.000000
variable        PRES            equal -1.000000      <_<_<
variable        TAU_T           equal 0.100000
variable        TAU_P           equal 0.500000

units           metal
boundary        p p p
atom_style      atomic

neighbor        1.0 bin

box          tilt large
read_data       conf.lmp
change_box   all triclinic
mass            1 1.000000
mass            2 12.000000
pair_style      deepmd ../graph.003.pb ../graph.002.pb ../graph.000.pb ../graph.001.pb  out_freq ${THERMO_FREQ} out_file model_devi.out
pair_coeff

thermo_style    custom step temp pe ke etotal press vol lx ly lz xy xz yz
thermo          ${THERMO_FREQ}
dump            1 all custom ${DUMP_FREQ} traj/*.lammpstrj id type x y z

velocity        all create ${TEMP} 727566
fix             1 all nvt temp ${TEMP} ${TEMP} ${TAU_T}

timestep        0.002000
run             ${NSTEPS}
```

下表列出了 `param.json` 中各关键词的详细说明：

其中粗体显示的关键词如 **type_map** 在所有情形下**必须赋值**。

关键词	数据结构	例子	描述
#基本参数			
type_map	List of string	["H", "C"]	元素种类
mass_map	List of float	[1.0, 12.0]	相对原子质量
use_ele_temp	int	0	暂时只支持 `"fp_style": vasp` 情形。 0(默认): 不设置电子温度 1: 以帧（rame）参数形式使用电子温度 2: 以原子参数形式使用电子温度。
#数据			
init_data_prefix	String	"...../dpgen-example/run"	为 `00.train` 指定初始训练数据所在目录的前缀
init_data_sys	List of string	["CH4.POSCAR.01x01x01/.../deepmd"]	为 `00.train` 指定初始数据所在目录。 可以使用绝对路径或相对路径。
sys_format	String	"vasp/poscar"	为 `00.train` 指定初始数据的格式,默认为 `vasp/poscar` 。
init_multi_systems	Boolean	false	为 `00.train` 指定 `init_data_sys` 下是否包含多个系统。 `true`：`init_data_sys` 目录须包含指向多个系统的次级目录，此时，初始数据将包含所有次级目录中系统的数据。

关键词	数据结构	例子	描述
init_batch_size	String of integer	[8]	指定 `00.train` 使用 `init_data_sys` 初始数据训练时，针对每个系统，单个 batch 所包含的帧（frame）数。值与 `init_data_sys` 中指定的系统按列表中元素的顺序一一对应。设定 `init_batch_size` (以及sys_batch_size) 的一种推荐原则是： 使指定的值乘以单帧结构的原子数大于 32。 如果设定为 `auto` (默认), 会自动指定为 32 除以单帧结构的原子数。
sys_configs_prefix	String	".....dpgen-example/run"	指定 `01.model_devi` MD 起始构象所在目录的前缀。
sys_configs	List of list of string	[["CH4.POSCAR.01x01x01/01.scale_pert/sys-0004-0001/scale-1.000/000000/POSCAR", "CH4.POSCAR.01x01x01/01.scale_pert/sys-0004-0001/scale-1.000/00000[1-9]/POSCAR"], ["CH4.POSCAR.01x01x01/01.scale_pert/sys-0004-0001/scale-1.000/000001/POSCAR"]]	指定 `01.model_devi` MD 的起始构象。 可以使用绝对路径或相对路径，支持**通配符**。
sys_batch_size	List of integer	[8, 8]	指定 `00.train` 训练 `01.model_devi` 以 `sys_configs` 构象起始筛选出的样本时，每个 batch 所包含的帧数。列表中的值与 `sys_configs` 列表中的元素一一对应。 可设置为 `auto`, 将指定为 32 除以单个构象的原子数。
#Training			
numb_models	Integer	4 (推荐值)	指定 `00.train` 训练的模型数量。
default_training_param	Dict	{ ... "sel": [16, 4], "rcut_smth": 0.5, "rcut": 5, "neuron": [10, 20, 40], ... }	指定 `00.train` 使用 `DeePMD-kit` 进行训练时的参数. 相关设置可以参考：(https://github.com/deepmodeling/deepmd-kit).. 一般推荐令 `stop_batch` = 200 * `decay_steps` 。 **注意**：其中的 `batch_size` 值会被 `init_batch_size` 和 `sys_batch_size` 值覆盖。
#Exploration			
model_devi_dt	Float	0.002(recommend)	指定 `01.model_devi` MD 的时间步长，单位: ps。
model_devi_skip	Integer	0	指定 `01.model_devi` 筛选构象时跳过每个 MD 轨迹起始帧的数量
model_devi_f_trust_lo	Float	0.05	指定 `01.model_devi` 筛选构象的模型偏差力判据下界
model_devi_f_trust_hi	Float	0.15	指定 `01.model_devi` 筛选构象的模型偏差力判据上界
model_devi_e_trust_lo	Float	1e10	指定 `01.model_devi` 筛选构象的模型偏差能量判据下界。 由于力判据能够提供更加精确的信息，推荐将能量判据下界设置为一个很高的数值 (无效化该判据)。 能量最小化等特定情形可能需要此判据生效。

关键词	数据结构	例子	描述
model_devi_e_trust_hi	Float	1e10	指定 `01.model_devi` 筛选构象的模型偏差能量判据上界。
model_devi_clean_traj	Boolean	true	指定 `01.model_devi` 是否清除输出文件中记载 MD 轨迹的 traj 文件夹(以节省空间)。
model_devi_jobs	[{ "sys_idx": [0], "temps": [100], "press": [1], "trj_freq": 10, "nsteps": 1000, "ensembles": "nvt" }, ...]	List of dict	指定 `01.model_devi` 调用 MD 模拟软件进行 DPMD 取样时的参数。 每个 dict {} 内的参数对应于一次迭代， `model_devi_jobs` 的序号与迭代的序号严格对应。
model_devi_jobs["sys_idx"]	List of integer	[0]	指定 `01.model_devi` 本次迭代选取 `sys_configs` 中的哪些构象作为 MD 的起始构象。序号与 `sys_configs` 列表中元素的序号一一对应。
model_devi_jobs["temps"]	List of integer	[50, 300]	指定 `01.model_devi` 本次迭代 MD 的温度 (**K**)，可在一次迭代中设置多个温度值。
model_devi_jobs["press"]	List of integer	[1,10]	指定 `01.model_devi` 本次迭代 MD 的压力 (**Bar**)，可在一次迭代中设置多个压力值。
model_devi_jobs["trj_freq"]	Integer	10	指定 `01.model_devi` 本次迭代存储 MD 轨迹中 snapshot 的频率（存储的 snapshot 构象可供筛选）。
model_devi_jobs["nsteps"]	Integer	3000	指定 `01.model_devi` 本次迭代 MD 的步数。
model_devi_jobs["ensembles"]	String	"nvt"	指定 `01.model_devi` 本次迭代 MD 选取的系综。 **选项** 包括 “npt” 和 “nvt”。
model_devi_jobs["taut"]	Float	0.1	恒温器的耦合周期 (fs)
model_devi_jobs["taup"]	Float	0.5	恒压器的耦合周期 (fs)
#Labeling			
fp_style	string	"vasp"	指定 `02.fp` 调用的第一性原理计算软件。 **选项**：目前包括 "vasp", "pwscf", "siesta", "gaussian" 和 "cp2k"。
fp_task_max	Integer	20	指定 `02.fp` 针对每次迭代以 `sys_configs` 列表中每个元素为起始筛选出的候选构象，进行第一性原理计算的构象个数上限，当候选构象数量大于其值时，将随机选取指定值个构象进行计算。
fp_task_min	Integer	5	指定 `02.fp` 针对每次迭代以 `sys_configs` 列表中每个元素为起始筛选出的候选构象，进行第一性原理计算的构象个数下限，当候选构象数量小于其值时，将忽略此次筛选的候选构象，不做第一性原理计算。

关键词	数据结构	例子	描述
fp_style == VASP			
fp_pp_path	String	"...../ch4/"	指定 `02.fp` vasp 计算所需赝势文件所在路径
fp_pp_files	List of string	["POTCAR_H", "POTCAR_C"]	指定 `02.fp` vasp 计算所需赝势文件的文件名。元素顺序需与 `type_map` 中记录的元素顺序一致。
fp_incar	String	"...../INCAR"	指定 `02.fp` vasp 计算所需的 INCAR 输入文件，INCAR 中必须指定 KSPACING。
cvasp	Boolean	true	指定 `02.fp` 是否使用 Custodian 辅助控制 VASP 计算。
fp_style == Gaussian			
use_clusters	Boolean	false	指定 `02.fp` 是否对局域团簇而非整个系统进行计算，需要 DeePMD-kit 版本1.x。
cluster_cutoff	Float	3.5	指定 `02.fp` 所计算团簇的截断半径，需要 `use_clusters`： `true`。
fp_params	Dict		指定 `02.fp` Gaussian 计算的参数。
fp_params["keywords"]	String or list	"mn15/6-31g** nosymm scf(maxcyc=512)"	指定 `02.fp` Gaussian 计算的关键词。
fp_params["multiplicity"]	Integer or String	1	指定 `02.fp` Gaussian 计算的自旋多重度。 可以设置为 `auto`： 自旋多重度将被自动指定。 可以设置为 `frag`： "fragment=N" 片段组合波函数方法将会被使用。
fp_params["nproc"]	Integer	4	指定 `02.fp` Gaussian 计算使用的核心数量。
fp_style == siesta			
use_clusters	Boolean	false	定 `02.fp` 是否对局域团簇而非整个系统进行计算，需要 DeePMD-kit 版本1.x。
cluster_cutoff	Float	3.5	指定 `02.fp` 所计算团簇的截断半径，需要 `use_clusters`： `true`。
fp_params	Dict		指定 `02.fp` siesta 计算的参数。
fp_params["ecut"]	Integer	300	指定 `02.fp` siesta 计算中平面波对格点的截断。
fp_params["ediff"]	Float	1e-4	指定 `02.fp` siesta 计算中密度矩阵的容差。
fp_params["kspacing"]	Float	0.4	指定 `02.fp` siesta 计算中对布里渊区的采样因子。
fp_params["mixingweight"]	Float	0.05	指定 `02.fp` siesta 计算中 scf 对前一步 scf 所得密度矩阵的继承比例(线性混合)。
fp_params["NumberPulay"]	Integer	5	指定 `02.fp` siesta 计算对 Pulay convergence accelerator 的设置.
fp_style == cp2k			
fp_params	Dict		指定 `02.fp` cp2k 计算的参数。请参考 manual.cp2k.org。只有 kind section 部分是必须设置。

1.2.2 机器配置文件 machine.json

机器配置文件为 `train`, `model_devi` 和 `fp` 三类任务指定了机器环境与资源需求，涉及使用的 `机器类型与作业管理系统`、`计算资源需求如 CPU/GPU 及内存需求、任务时限，依赖的环境配置、需要加载的模块等` 与 `执行计算的指令及计算机器的任务量的分配`

等。
对每类任务，我们使用一个相对独立的列表来存储相应的关键词设置（list of dict），从而能够为不同类型的计算指定适合的计算环境与资源。

当初次使用新的机器环境时，用户需要根据实际情况调整 `MACHINE` 中的设置。此后， `MACHINE` 就可以被任意类型的 DP-GEN 任务重复使用而不需要再做额外的更改。

本次上手使用阿里云机器，机器配置文件跟据实际情况及 API 做了相应的调整。通过 `run` 例子文件夹中的 `machine-ali_local.json` 文件，我们将实现使用远程 GPU 节点执行 `00.train` 与 `01.model_devi`，以及使用本地 CPU 执行 `02.fp`。

`机器类型与作业管理系统` 的设置由 `machine` dict 来指定，远程与本地计算两种情形下的设置会有区别：

向远程节点提交任务，以例子文件中 `train` 部分为例：

- `machine_type`： 由于阿里云弹性计算服务并不需要传统高性能计算集群上常用的 `slurm` `lsf` 等作业管理系统，所以用于指定作业管理系统的该关键词并不需要出现。
- `regionID`： 指定阿里云(远程)机器所在的区位，本次阿里云解决方案所特有。其他传统 HPC 集群等情形不需要出现该关键词。
- `batch` 指定使用 shell 环境来运行脚本(若 HPC 等情形下， `machine_type` 指定了slurm 等作业系统，不需要该关键词)。
- `hostname` 指定所用(HPC情形多为登陆)节点的IP，本次阿里云环境不需设置。
- `port`：与远程节点通讯的端口号，一般可设置为 22。
- `username` 与 `password` 在远程节点拥有权限的用户名与密码。
- `work_path` 远程节点上，存放计算任务临时文件的路径。
- `ali_auth` 阿里云 API 所需的认证信息，其他情形不需要。

在本地节点执行任务，以例子文件中 `fp` 部分为例，由于不需要登陆操作且本地机器没有 slurm 等作业管理系统，上述关键词只需要保留 `batch` 与 `work_path`。

```
{
  "train": [
    {
      "machine": {
        "regionID":            "cn-beijing",
        "batch":               "shell",
        "hostname":            "",
        "password":            "password",
        "port":                22,
        "username":            "root",
        "work_path":           "/root/dpgen_work",
        "ali_auth": {
            "AccessKey_ID":    "AccessKey_ID",
            "AccessKey_Secret":"AccessKey_Secret",
            "instance_name":   "workshop_CH4",
            "pay_strategy":    "spot",
            "password":        "password"
        }
      },
      "resources": {
        "numb_gpu":            1,
        "numb_node":           1,
        "task_per_node":       12,
        "partition":           "gpu",
        "exclude_list":        [],
        "mem_limit":           32,
        "source_list":         [],
        "module_list":         [],
        "time_limit":          "23:0:0"
      },
      "command":               "/root/deepmd-kit/bin/dp",
      "group_size":            1
    }
  ],

  "model_devi": [
    {
      "machine": {
        "regionID":            "cn-beijing",
        "batch":               "shell",
        "hostname":            "",
        "password":            "password",
        "port":                22,
        "username":            "root",
        "work_path":           "/root/dpgen_work",
        "ali_auth": {
            "AccessKey_ID":    "AccessKey_ID",
            "AccessKey_Secret":"AccessKey_Secret",
            "instance_name":   "workshop_CH4",
            "pay_strategy":    "spot",
            "password":        "password"
        }
      },
      "resources": {
        "numb_gpu":            1,
        "task_per_node":       4,
        "partition":           "gpu",
        "exclude_list":        [],
        "mem_limit":           11,
        "source_list":         [],
        "module_list":         [],
        "time_limit":          "23:0:0"
      },
      "command":               "/root/deepmd-kit/bin/lmp",
      "group_size":            5
    }
  ],

  "fp": [
  {
      "machine": {

        "batch":              "shell",
        "work_path":          "/root/dpgen_work"
      },

      "resources": {
        "numb_gpu":           0,
        "task_per_node":      8,
        "with_mpi":           false,
        "source_list":        ["/opt/intel/parallel_studio_xe_2019/psxevars.sh"],
```

```
        "module_list":        [],
              "partition":          "cpu",
        "envs" :              {"PATH" : "/root/vasp/bin:$PATH"}
      },
      "command":              "ulimit -s unlimited && mpirun -n 8 /root/vasp/bin/vasp_std",
      "group_size":           30
    }
  ]
}
```

提示

- `command` 为用户提供额外的操作自由度：
  注意到上面 `fp` 部分 `resources` 中， 我们令 `"with_mpi": false`，但在实际执行的命令中依旧使用了mpirun `ulimit -s unlimited && mpirun -n 8 /root/vasp/bin/vasp_std`。
  这是由于我们希望在执行任务前附加命令 `ulimit -s unlimited` 以避免 VASP 运行出错。而当 `"with_mpi": true` 时，以上 CPU 任务设置会自动拼接出命令 `mpirun -n 8 command` 从而不能实现这一需求。于是，我们使用了上述的定制 `command` 方式，相应地就需要手动输入 `mpirun`。
  用户可以根据需求灵活使用自动化设置或定制命令。

以下表格指明了各关键词在三种类型任务中是否需要出现，及详细的说明。

关键词	`train`	`model_devi`	`fp`
machine	需要	需要	需要
resources	需要	需要	需要
deepmd_path / python_path	需要		
command	需要	需要	需要
group_size	需要	需要	需要

关键词	**数据结构**	**例**	**描述**
deepmd_path	String	".....tf1120-lowprec"	指定 DeePMD-Kit 0.x 的安装目录, 目录中应包含 `bin lib include`。
python_path	String	".....python3.6/bin/python"	若使用 DeePMD-kit 1.x，则指定Python路径。与 `deepmd_path` 不应一起使用。
machine	Dict		指定机器类型与作业管理环境, 需针对 `train` `model_devi` `fp` 三类任务分别指定。
resources	Dict		指定计算任务的资源需求，需针对 `train` `model_devi` `fp` 三类任务分别指定。
# 以下是 `resources` Dict 中的关键词			
numb_node	Integer	1	指定每组(顺序执行的，见下 `group size`)计算任务所需的节点数
task_per_node	Integer	4	指定每组计算任务所需的CPU核心(cores)数
numb_gpu	Integer	4	指定每组计算任务所需的GPU卡数
manual_cuda_devices	Interger	1	为 `01.model_devi` 指定每组 MD 任务可按照 “manual_cuda” 方式使用的GPU卡数，需与 `manual_cuda_multiplicity` 一同使用。
manual_cuda_multiplicity	Interger	5	指定 `01.model_devi` 步骤能够在 1 块 GPU卡上同时执行的 MD 程序数量，DP-GEN 会根据指定值及 `manual_cuda_devices`，自动将每组 MD 任务分配到不同的 GPU 卡上，这可以提高 V100 等 GPU 的利用效率。
node_cpu	Integer	4	仅适用于 LSF 作业管理系统，指定为每组任务分配的CPU核心数。
source_list	List of string	".....vasp.env"	指定相应步骤任务执行前需要 source 生效的环境文件。如果 `*".env"` 在列表中，'source *.env' 将被写入执行任务的脚本。
module_list	List of string	["Intel/2018", "Anaconda3"]	指定相应步骤任务执行前需要加载的模块，如果 `"Intel/2018"` 在列表中，`module load Intel/2018` 将被写入执行任务的脚本。
time_limit	String (time format)	23:00:00	指定每组任务的最大允许时限

关键词	数据结构	例	描述
partition	String	"AdminGPU"	指定相应步骤任务使用的机器分组或队列。
mem_limit	Interger	16	指定计算任务允许申请的内存上限。
with_mpi	Boolean	true	指定是否使用 mpi 进行计算 `true` 且 `"machine_type": "slurm"`, 则执行任务的脚本中 `command` 前会以添加 "srun"。
qos	"string"	"bigdata"	指定任务的优先级，可选值由使用的超算集群决定。
allow_failure	Boolean	false	允许脚本提交的指令返回非 0 值的退出状态码。
# 放置于 resources Dict 的末尾			
command	String	"lmp_serial"	指定执行任务所需软件的可执行路径, 例如 `lmp_serial` `lmp_mpi` `vasp_gpu` `vasp_std` 等(可使用绝对路径)。
group_size	Integer	5	指定 DP-GEN 将多少个计算任务安排为一个顺序执行的组，使用 1 个脚本文件进行提交。

1.2.3 操作与输出介绍

```
nohup dpgen run param.json machine_local.json > log.out 2>&1 &
```

迭代细节与输出文件

在等待例子任务执行完成的时间里，我们可以通过 previous_example 来熟悉主流程的迭代细节与输出文件的结构。

00.train

```
ls previous_example/iter.000000/00.train/
```

```
000    002    data.init    graph.000.pb graph.002.pb
001    003    data.iters   graph.001.pb graph.003.pb
```

`graph.00x.pb` 是指向 `00x/frozen.pb` 的软链接文件，为 `DeePMD-kit` 训练产生的模型：

```
ls -l previous_example/iter.000000/00.train/
```

```
000
001
002
003
data.init -> /root/workshop/dpgen-example/run
data.iters
graph.000.pb -> 000/frozen_model.pb
graph.001.pb -> 001/frozen_model.pb
graph.002.pb -> 002/frozen_model.pb
graph.003.pb -> 003/frozen_model.pb
```

data.init 指向的即 `param.json` 中的 `init_data_prefix`。

00.train/000

接下来我们查看一下所训练的000模型文件夹：

```
ls previous_example/iter.000000/00.train/000/
```

```
frozen_model.pb  input.json  lcurve.out
```

- `input.json` 是当前任务使用 `DeePMD-kit` 进行训练的输入文件。
- `lcurve.out` 记录了训练过程中的训练误差和测试误差。
- `train.log` 记录了训练过程中的标准输出。

接下来我们进入 000 文件夹查看一下 `lcurve.out` 的记录：

```
cd previous_example/iter.000000/00.train/000/
head -n 2 lcurve.out && tail -n 2  lcurve.out
```

```
# batch      l2_tst    l2_trn    l2_e_tst  l2_e_trn    l2_f_tst  l2_f_trn         lr
      0    8.19e+01  8.83e+01    6.26e+00  6.22e+00    2.59e+00  2.79e+00    1.0e-03
  35000    1.29e-01  1.22e-01    4.16e-04  5.83e-04    1.26e-01  1.19e-01    4.8e-08
  36000    1.29e-01  1.48e-01    4.21e-04  6.27e-04    1.26e-01  1.46e-01    3.5e-08
```

batch 的最终数值即为 `param.json` 中 `stop_batch` 的指定值。

在 `run` 主流程中，我们主要关注力的精度，可以看到 36000 个 batch 后，力的训练和测试误差都已经比初始误差小了一个数量级，所以训练是有效的，精度作为样例可以接受。

01.model_devi

接下来，我们查看 01.model_devi 相关的文件

```
cd ../../01.model_devi/
ls
```

```
confs         graph.000.pb  graph.002.pb  task.000.000000  task.000.000002  task.000.000004  task.000.000006  task.000.000008
cur_job.json  graph.001.pb  graph.003.pb  task.000.000001  task.000.000003  task.000.000005  task.000.000007  task.000.000009
```

这里的 10 个 task.000.00000* 文件夹对应着我们在 `param.json` 中通过 `sys_configs` 为 `01.model_devi` 指定的 10 个 MD 起始构象，分别按顺序存储着以这 10 个起始构象开始的 `01.model_devi` 输出文件。

```
[
  "CH4.POSCAR.01x01x01/01.scale_pert/sys-0004-0001/scale-1.000/000000/POSCAR",
  "CH4.POSCAR.01x01x01/01.scale_pert/sys-0004-0001/scale-1.000/000001/POSCAR",
  "CH4.POSCAR.01x01x01/01.scale_pert/sys-0004-0001/scale-1.000/00000[2-9]/POSCAR"
],
```

POSCAR 格式的起始结构与转换后的 LAMMPS 输入格式起始结构存储在confs目录中。
cur_job.json 中记录的是 `param.json` 中 `model_devi_jobs` 指定的当前迭代任务信息：

```
cat cur_job.json
```

```
{
    "sys_idx": [
        0
    ],
    "temps": [
        50
    ],
    "press": [
        1
    ],
    "trj_freq": 10,
    "nsteps": 1000,
    "ensemble": "nvt",
    "_idx": "00"
}
```

接下来，以 `task.000.000006` 为例查看一个任务的输文件：

```
cd task.000.000006/
ls
```

```
conf.lmp  input.lammps  job.json  model_devi.log  model_devi.out  traj
```

- `conf.lmp` 输入文件：当前任务的 LAMMPS 输入格式起始构象
- `input.lammps` 输入文件：当前任务由 `DP-GEN` 自动生成的 LAMMPS 输入脚本
- `model_devi.out` 输出文件：记录 DPMD 采样构象的能量和力的模型偏差

`model_devi.out` 中记录的 `max_devi_f` 是采样构象应否被选取为候选构象，送至 `02.fp` 进行第一性原理计算的判据。

```
head model_devi.out
```

```
#       step         max_devi_e         min_devi_e         avg_devi_e         max_devi_f         min_devi_f         avg_devi_f
          0       1.702540e+00       4.072232e-01       6.811721e-01       2.398877e-02       9.889648e-03       1.904143e-02
         10       1.724074e+00       4.139336e-01       6.899073e-01       4.082618e-02       1.311508e-02       3.333620e-02
         20       1.749241e+00       4.221212e-01       6.998610e-01       5.046610e-02 <_<_< 8.202700e-03       3.982145e-02
         30       1.762483e+00       4.295333e-01       7.051363e-01       2.386741e-02       1.167313e-02       1.808281e-02
         40       1.753675e+00       4.293247e-01       7.016724e-01       5.860581e-02 <_<_< 6.494130e-03       4.624764e-02
         50       1.731517e+00       4.239708e-01       6.926950e-01       3.087103e-02       7.004353e-03       2.431688e-02
         60       1.707782e+00       4.208287e-01       6.832595e-01       4.169945e-02       1.226093e-02       3.337300e-02
         70       1.692729e+00       4.168854e-01       6.771580e-01       4.525375e-02       8.779865e-03       3.710275e-02
         80       1.691821e+00       4.164997e-01       6.767669e-01       1.825787e-02       9.605197e-03       1.525421e-02
```

根据 `param.json` 中 `"trj_freq": 10`, DPMD 轨迹的采样频率为每 10 帧保存一帧构象。
而后根据 `param.json` 中 `"model_devi_f_trust_lo": 0.05` 及 `"model_devi_f_trust_hi": 0.15`， DP-GEN 会对保存的构象进行判定和筛选。

上例所展示的这些帧中，第 20 和 40 帧将被选为候选构象，将被送至 `02.fp`。

02.fp

```
cd ../../02.fp/
ls
```

```
candidate.shuffled.000.out      task.000.000000  task.000.000005  task.000.000010  task.000.000015  task.000.000020  task.000.000025
data.000                        task.000.000001  task.000.000006  task.000.000011  task.000.000016  task.000.000021  task.000.000026
POTCAR.000                      task.000.000002  task.000.000007  task.000.000012  task.000.000017  task.000.000022  task.000.000027
rest_accurate.shuffled.000.out  task.000.000003  task.000.000008  task.000.000013  task.000.000018  task.000.000023  task.000.000028
rest_failed.shuffled.000.out    task.000.000004  task.000.000009  task.000.000014  task.000.000019  task.000.000024  task.000.000029
```

- `candidate.shuffle.000.out`：输出文件：记录被筛选为候选构象的结构在 `01.model_devi` 中所对应的任务号和帧数。如：`iter.000000/01.model_devi/task.000.000000 70`。
  通常，每个迭代以 `sys_configs` 中一个元素为起始构象，所筛选出的候选构象数量远大于 `fp_task_max` 设定的上限。此时，DP-GEN 会随机选择 `fp_task_max` 指定值数量的结构进行第一性原理计算，并创建目录 `task.*.0000*`，文件夹名字中的第一数字代表迭代序号，第二个数字代表 `02.fp` 第一性原理计算任务的序号。
- `rest_accurate.shuffle.000.out` 和 `rest_failed.shuffle.000.out`：输出文：分别记录 `max_devi_f` 小于 `model_devi_f_trust_lo` 和大于 `model_devi_f_trust_hi` 的构象在 `01.model_devi` 中所对应的任务号和帧数。
- `data.000` 输出文件夹：第一性原理计算完成后， `DP-GEN` 收集整理计算结果并转换为 `DeePMD-kit` 训练所需数据的格式，存储在该文件夹中。在下一个迭代的 `00.train`，这些数据会与初始数据一起被加入到数据集中。
- `POTCAR`：输入文件： 由 `param.json` 中 `fp_pp_path` 和 `fp_pp_files` 指定的 VASP 计算所需两种元素赝势文件合并而成(注意相应位置指定的元素顺序)，所有第一性原理任务都使用该赝势文件。

```
cat candidate.shuffled.000.out | grep task.000.000006
```

```
iter.000000/01.model_devi/task.000.000006 620
iter.000000/01.model_devi/task.000.000006 220
iter.000000/01.model_devi/task.000.000006 40       <_<_<
iter.000000/01.model_devi/task.000.000006 270
iter.000000/01.model_devi/task.000.000006 20       <_<_<
iter.000000/01.model_devi/task.000.000006 670
iter.000000/01.model_devi/task.000.000006 960
iter.000000/01.model_devi/task.000.000006 990
iter.000000/01.model_devi/task.000.000006 610
iter.000000/01.model_devi/task.000.000006 140
iter.000000/01.model_devi/task.000.000006 290
iter.000000/01.model_devi/task.000.000006 320
iter.000000/01.model_devi/task.000.000006 910
iter.000000/01.model_devi/task.000.000006 560
iter.000000/01.model_devi/task.000.000006 240
iter.000000/01.model_devi/task.000.000006 590
iter.000000/01.model_devi/task.000.000006 640
iter.000000/01.model_devi/task.000.000006 90
```

在下一迭代的 `00.train` 中，新的数据将被加入到数据集, 我们可以通过查看 DP-GEN 为 iter.000001 生成的训练输入脚本看到：

```
grep -A 3 "systems" ../../iter.000001/00.train/000/input.json
```

```
"systems": [
    "../data.init/CH4.POSCAR.01x01x01/02.md/sys-0004-0001/deepmd",
    "../data.iters/iter.000000/02.fp/data.000"                      <_<_<
],
```

最后，我们检查一下 iter.000000 新加入数据对模型精度的改进。
我们退回到 precious_example 目录，

```
cd ../..
```

首先，我们统计一下初始模型的表现。

```
wc -l iter.000000/02.fp/*.out
```

```
  82 iter.000000/02.fp/candidate.shuffled.000.out
 928 iter.000000/02.fp/rest_accurate.shuffled.000.out
   0 iter.000000/02.fp/rest_failed.shuffled.000.out
1010 total
```

在 1000 步 MD 中，初始模型对其中928个构象的进行了精确的预测。

注意到在 previous_example 的 `param.json` 中，`fp_task_max` 被指定为 30,，只有30个新数据被加入到了 iter.000001 的数据集中。
然而，我们查看更新后的模型的表现：

```
wc -l iter.000001/02.fp/*.out
```

```
2 iter.000001/02.fp/candidate.shuffled.001.out
3008 iter.000001/02.fp/rest_accurate.shuffled.001.out
0 iter.000001/02.fp/rest_failed.shuffled.001.out
3010 total
```

新模型表现出了显著更高的预测精度。
此外，第二次动力学的时长时第一次动力学的3倍，且我们在 `param.json` 中通过 `sys_configs` 为两次模拟指定了不同的起始构象。

新模型针对更长 MD 模拟中采样构象的具体模型偏差力判据可以在 `iter.000001/01.model_devi/task*/model_devi.out` 查看。

`model_devi_jobs` 中指定的迭代都进行完毕后， `run` 会进行额外的最后一个迭代（本例为 iter.000002），只会使用初始数据及之前迭代积累的所有新增数据进行 `00.train`，随后主进程执行完成。

提示 此处 previous_example 中最后一个迭代 iter.000002 实际并没有进行新的 `00.train`, 而是拷贝了（软连接）iter.000001的 00.train 文件夹，

```
ls -l iter.000002/00.train/
```

```
000 -> ../../iter.000001/00.train/000
001 -> ../../iter.000001/00.train/001
002 -> ../../iter.000001/00.train/002
003 -> ../../iter.000001/00.train/003
copied                                  <_<_<
graph.000.pb -> 000/frozen_model.pb
graph.001.pb -> 001/frozen_model.pb
graph.002.pb -> 002/frozen_model.pb
graph.003.pb -> 003/frozen_model.pb
```

这是由于前一迭代 iter.000001 只筛选出 2 个候选构象，小于 `param.json` 中的 `"fp_task_min": 8`。 iter.000001 的 02.fp 被跳过，并没有向数据集中加入新的数据，所以没有执行新的模型训练任务。

而 `"fp_task_min": 8` 的原因是，根据实现有效的模型训练时，推荐每个 batch 所包含的原子数大于等于 32 的原则，对于甲烷分子，我们在 `param.json` 设置 `"init_batch_size: 8"` 和 `"sys_batch_size": 8`。
在对一般的实际系统进行研究时，当一次迭代中一个体系所提供的所有候选构象不能满足原子总数大于 32 时，推荐将计算资源用于对于模型训练来说可能具有更高价值的其他采样筛选情形, 一般令 `fp_task_min` > batch_size（DP-GEN 自动将每个迭代以 `sys_configs` 中 1 个元素所对应构象为起始，所筛选出的数据，整理为一个独立的体系，数据格式转换为 DeePMD-kit 输入数据的格式，存储在 1 个 独立的set内，若 `fp_task_min` < batch_size 将报错, 类似地， `fp_tack_min` 也应该 > `numb_test`）。

2.性质测试 | 铝

2.1 auto-test 自动性质测试

```
dpgen test PARAM MACHINE
```

使用逻辑：

- `选定参考系统，通过第一性原理计算或传统力场方法计算其性质` → `对参考体系，应用 DP 模型计算其性质` → `比对结果，评估 DP 模型质量`。
- `使用 auto-test 自动预测体系的多种性质`。

支持的测试任务类型：

- `vasp` 第一性原理
- `deepmd` DP模型
  - 当提供多个DP模型(如 `run` 得到的 graph.00*.pb)时,
    基于 `DeePMD-kit 1.x` 进行的计算，将使用第一个模型进行性质预测，并针对计算任务中涉及的构象输出能量与力的模型偏差 `model_devi.out`。
    基于 `DeePMD-kit 0.x` 进行的计算，将给出各模型预测性质的平均值，并给出模型偏差。
- `meam` 经验力场

基本流程支持的性质：

- `00.equi`：平衡态计算
- `01.eos`：状态方程(E-V 曲线)
- `02.elastic`：弹性
- `03.vacancy`：空位形成能
- `04.interstitial`：间隙形成能
- `05.surf`：表面形成能

文件结构：

- 输出文件的目录结构, **依赖于**参考系统构象文件的存储路径。
  如 POSCAR 构象文件存储在 `./confs/"元素"/"构象名"` （例：./confs/Al/std-fcc）文件夹中，则DP-GEN 将在当前工作目录 `./` 下创建以上各性质的同名文件夹，按 `./性质/"元素"/"构象名"/任务类型/分解计算任务(如有)` 的目录结构存储输出文件，计算完成后分析得到的性质结果 `result` 文件 (00.equi 不需) 存储在 `/任务类型` 级目录中。
  - 推荐将参考系统构象文件的存储路径如上指定为 `./confs/"元素"/"构象名"`，其中，如果将:
    `"元素"` 的格式写为： `Al`， `Al-Mg`， `X-Y-Z` 等。
    `"构象名"` 的格式写为 `std-*` (std 代表标准晶型，* 为标准晶型的种类如 fcc，bcc，hcp等）或 `mp-**` (mp 代表 Materials Project, ** 为 Materials Project数据库中该构象的代码，如 2020，→ mp-2020)。
    则DP-GEN会(根据 `param.json` 中 `conf_dir` 指定的上述相对或绝对路径)自动检测该目录中是否存在 POSCAR 文件，如不存在，则（在网络允许的情况下，使用 Materials Project 授权的 API key）自动下载该构象的 POSCAR 文件存储在指定目录中。
- **注意** 如将参考系统的结构文件直接存储在当前工作目录中 `./POSCAR`，将导致不再创建性质同名文件夹，输出文件存储在 `./任务类型/分解计算任务` 中，当使用同一任务类型计算多个性质时，会导致 `./任务类型` 中的同名文件创建失败或覆写从而报错。

下面，我们进入 test 例子文件夹熟悉一下文件结构，并基已经训练好的 AI 模型预测 AI fcc 晶体的以上各性质， 相应的第一性原理计算结果已经事先准备好。

```
cd /root/workshop/dpgen-example/test
ls
```

```
00.equi  01.eos  02.elastic  03.vacancy  04.interstitial  05.surf  Al_model  confs  machine_local.json  param.json  record.auto_test  stables
```

模型所在目录：

```
ls Al_model/
```

```
graph.000.pb  graph.001.pb  graph.002.pb  graph.003.pb
```

构象文件目录：

```
tree confs
```

```
confs/
└── Al
    └── std-fcc
        └── POSCAR

2 directories, 1 file
```

输出文件目录

```
tree 03.vacancy/
```

```
03.vacancy/
└── Al
    └── std-fcc
        └── vasp-relax_incar
            ├── INCAR
            ├── jr.json
            ├── POSCAR
            ├── POTCAR
            ├── result
            └── struct-3x3x3-000
                ├── autotest.out
                ├── INCAR
                ├── OSZICAR
                ├── OUTCAR
                ├── POSCAR
                ├── POTCAR
                └── supercell.out

4 directories, 12 files
```

`result` 文件：

```
head 01.eos/Al/std-fcc/vasp-relax_incar/result
```

```
conf_dir:./confs/Al/std-fcc
 VpA(A^3)  EpA(eV)
 12.000   -3.2186
 12.500   -3.3582
 13.000   -3.4692
 13.500   -3.5562
 14.000   -3.6228
 14.500   -3.6724
 15.000   -3.7047
 15.500   -3.7275
```

2.1.1 输入参数 param.json

```
dpgen test param.json machine_local.json
```

输入文件的第一部分：

```
"_comment":           "reference geometry, which method & property",
"conf_dir":           "./confs/Al/std-fcc",
"key_id":             "",
"task_type":          "deepmd",
"task":               "all",
```

- `conf_dir` : 指定参考系统 POSCAR 构象文件所在目录
- `key_id` : 如果使前述推荐的 `conf_dir` 写法而指定的目录下不存在 POSCAR 结构文件, DP-GEN 将使用此处指定的API key 由 Materials Project 数据库自动下载该构象的 POSCAR 文件至 `conf_dir` 中。
- `task_type` : 测试任务的类型, `vasp` `deepmd` `meam` 三选一。
- `task` : 测试的性质 `equi` , `eos` , `elastic` , `vacancy` , `interstitial` , `surf` 及 `all` , `all` 会依次计算上述所有性质。
- **注意** 对于较大系统的性质测试， `"task_type": "vasp"` 情况下不推荐使用 `"task": "all"` , 由于第一性原理计算资源需求较高，推荐用户根据研究系统与计算任务等实际情况，合理设置相关参数(如针对较大系统的计算，分别使用低，中，高精度接续计算以提高计算效率等)，指定 VASP 计算输入文件的关键词见下。

输入文件的第二部分：

```
"_comment":           "files and parameters for VASP and LAMMPS"
"relax_incar":        "./relax_incar",
"scf_incar":          "./scf_incar_sp",
"potcar_map":         {
                      "Al": "./POTCAR_Al"
                      },
"vasp_params":{
  "ecut":              650,
  "ediff":             1e-6,
  "kspacing":          0.1,
  "kgamma":            false,
  "npar":              1,
  "kpar":              1,
  "LREAL":             false,
  "_comment":          "that's all"
},
"lammps_params": {
  "model_dir":         "./Al_model",
  "type_map":          ["Al"],
  "deepmd_version":    "1.1.0",
  "model_name":        false,
  "model_param_type":  false
},
```

第二部分是对 VASP 和 LAMMPS 计算所需文件和参数的设置，大部分关键词的说明前面已经介绍过，新的关键词及需要注意的部分如下：

- `relax_incar`：与 `scf_incar` 处指定的 VASP 输入文件中参数的优先级，高于 `vasp_params` 的设置及 DP-GEN 生成 VASP 脚本时使用其的他默认参数。如指定 `relax_incar`，需注意相关参数的设置需要依据测试性质的不同而做调整。
- `model_dir`：为 `deepmd` 及 `meam` 任务类型指定 DP 或 meam 经验力场模型所在路径
- `type_map`：为 `deepmd` 及 `meam` 任务类型指定元素种类
- `deepmd_version`：指定训练当前 DP 模型时所使用的 DeePMD-kit 的版本，如不做指认，则**默认**只支持 **0.x** 版本生成的模型。
- `model_name`：为 `meam` 任务类型指定模型名称，指定时数据结构为列表。其余任务类型可设置为 false (如将 DP 模型文件名修改为非 *.pb, 可在此处指定新的模型文件名)。
- `model_param_type`：为 `meam` 任务类型指定模型参数类型，指定时数据结构为列表。其余任务类型可设置为 false。
  提示 `meam` 任务类型相关关键词的设置可参考 DP-GEN 程序包例子文件夹中的: “dpgen/examples/test/meam_param.json”。

输入文件的第三部分：
以下部分指定测试的性质，可按需求在 `param.json` 指定特定或多个性质的参数。

- **注意** 由于平衡态计算是后续其他续测试任务的基础(如在平衡态结构基础上做调整，或以平衡态能量为参考值等)，请在开始各任务类型的其他性质测试之前，完成 VASP 的 `00.equi` 测试，否则部分性质测试会报错。 `deepmd` 的 `00.equi` 如在其他 `deepmd` 类型测试前未执行，将被自动附带完成。

00.equi

```
"_comment": "00.equi",
"store_stable":      true,
"alloy_shift":       false,
```

- `store_stable`
  - 对单质：指定是否将平衡态计算所得的原子平均势能与原子平均体积存储在工作目录下的 `./stables` 文件夹中(作为该元素原子平均势能的参考值，用于后续合金/多元素体系的形成能计算)。
  - 对多元素系统：不生效。
- `alloy_shift`：
  - 对单质：不生效。
  - 对多元素体系，指定是否计算系统(合金)以各元素单质为势能参考起点的形成能。
    需要 `./stables` 文件夹存在，且存储有各组分元素的原子平均势能文件 `元素.任务类型(计算参数).e`。

  提示 计算合金形成能之前，需根据情况选取合适的单质体系作为势能参考点。

01.eos

```
"_comment": "01.eos",
"vol_start":         12,
"vol_end":           22,
"vol_step":          0.5,
```

- `vol_start` `vol_end` 和 `vol_step` 分别指定状态方程 E-V 关系计算中的原子初始原子平均体积、最终原子平均体积和原子平均体积变化步长，单位：$Å^3$。
  下图所为使用原子平均体积分别为 12.0 $Å^3$, 16.5 $Å^3$ 和 21.5 $Å^3$ 时 Al fcc 3x3x3 超胞的构象

02.elastic

```
"_comment": "02.elastic",
"norm_deform":       2e-2,
"shear_deform":      5e-2,
```

- `norm_deform` 和 `shear_deform` 分别指定构象的晶格发生单方向上拉伸压缩和剪切形变时，改变量的比例范围(对应形变矩阵中对角元的改变比例和非对角元的大小)，实现中会取指定值的 [-1.0, -0.5, 0.5, 1.0] 倍生成形变构象，计算完成后根据应力-形变关系分析系统的弹性性质。
  具体实现细节请参见 DP-GEN 主页 "The content of the auto_test" 部分的链接与我们以往的教程。

03.vacancy

```
"_comment": "03.vacancy",
"supercell":         [3, 3, 3],
```

- `supercell`：指定计算 `03.vacancy` 空位形成能**与** `04.interstitial` **间隙形成能** 时的超胞大小。
- **注意** 当构象结构文件的晶胞中只有一个原子时，不做扩胞将产生“真空”导致计算出错。

04.interstitial

```
"_comment": "04.interstitial",
"insert_ele":        ["Al"],
"reprod-opt":        true,
```

- `insert_ele`：指定间隙原子的元素种类。
- `repord-opt`：指定 `deepmd` 及 `meam` 任务类型的计算是否不做优化，而是根据之前 `vasp` 优化轨迹中的构象进行单点计算，复现 `vasp` 的优化轨迹。不可用于 `vasp` 任务类型。

05.surface

```
"_comment": "05.surface",
"min_slab_size":     10,
"min_vacuum_size":   11,
"_comment": "pert xz to work around vasp bug...",
"pert_xz":           0.01,
"max_miller":        2,
"static-opt":        true,
"relax_box":         false
```

- `min_slab_size` 和 `min_vacuum_size`：分别指定样品层和真空层的最小厚度。
- `pert_xz`：指定在 `vasp` 类型计算时，改变 POSCAR 文件晶格 c 矢量 x 坐标的大小(为了避免 vasp 计算因 bug 出错)，单位：Å。
- `max_miller`：指定产生表面的最大 miller 指数。
- `static-opt`：指定是否只对表面进行静态计算而不优化原子位置。
- `relax_box`：指定是否优化构象的盒子(晶格)。

产生的表面结构图示：

2.1.2 输出介绍

00.equi

test results

```
conf_dir:        EpA(eV)  VpA(A^3)
confs/Al/std-fcc  -3.7456   16.477
```

字段	数据结构	例	描述
EpA(eV)	实数	-3.7456	原子平均势能
VpA(A^3)	实数	16.477	原子平均体积

01.eos

```
conf_dir:./confs/Al/std-fcc
VpA(A^3)  EpA(eV
12.000   -3.1389
12.500   -3.3207
13.000   -3.4315
13.500   -3.5151
14.000   -3.5799
14.500   -3.6303
15.000   -3.6678
15.500   -3.7293
16.000   -3.7418
16.500   -3.7456
17.000   -3.7416
17.500   -3.7314
18.000   -3.7162
18.500   -3.6965
19.000   -3.6734
19.500   -3.6474
20.000   -3.6190
20.500   -3.5888
21.000   -3.5574
21.500   -3.5250
```

字段	数据类型	例	描述
EpA(eV)	实数列表	[15.5,16.0,16.5,17.0]	原子平均势能
VpA(A^3)	实数列表	[-3.7293, -3.7418, -3.7456, -3.7416]	原子平均体积

02.elastic

```
conf_dir:./confs/Al/std-fcc
  134.46   55.97   52.69    4.64    0.00    0.00
   56.16  134.18   52.67   -4.13    0.00    0.00
   52.71   52.71  136.53    0.00    0.00    0.00
    4.38   -3.63   -0.98   34.98    0.00   -0.00
   -0.00    0.00    0.00   -0.00   35.06    3.82
    0.00    0.00    0.00   -0.00    4.29   37.86
 #Bulk   Modulus BV = 80.90 GPa
 #Shear  Modulus GV = 37.84 GPa
 #Youngs Modulus EV = 98.20 GPa
 #Poission Ratio uV = 0.30
```

字段	数据类型	例子	描述
elastic module(GPa)	6*6 实数矩阵	[[134.46 55.97 52.69 4.64 0.00 0.00] [56.16 134.18 52.67 -4.13 0.00 0.00] [52.71 52.71 136.53 0.00 0.00 0.00] [4.38 3.63 -0.98 34.98 0.00 -0.00] [-0.00 0.00 0.00 0.00 35.06 3.82] [0.00 0.00 0.00 -0.00 4.29 37.86]]	Voigt- 记法表示的弹性模量; 行矩阵元的顺序为 (xx, yy, zz, yz, xz, xy)
Bulk Modulus(GPa)	实数	80.90	体模量
Shear Modulus(GPa)	实数	37.84	剪切模量
Youngs Modulus(GPa)	实数	98.20	杨氏模量
Poission Ratio	实数	0.30	泊松比

03.vacancy

```
conf_dir:./confs/Al/std-fcc
Structure:      Vac_E(eV)  E(eV) equi_E(eV)
struct-3x3x3-000:   0.717  -96.687 -97.404
```

字段	数据结构	例	描述
Structure	字符串列表	['struct-3x3x3-000']	构象名称
Vac_E(eV)	实数	0.717	空位形成能 = E - equi_E
E(eV)	实数	-96.687	空位缺陷构象的势能

字段	数据结构	例	描述
equi_E(eV)	实数	-97.404	平衡态原子平均势能 * 空位构象原子数

04.interstitial

```
/root/workshop/dpgen-example/test_0/04.interstitial/Al/std-fcc/deepmd-reprod-relax_incar/struct-Al-3x3x3-000
18 18
/root/workshop/dpgen-example/test_0/04.interstitial/Al/std-fcc/deepmd-reprod-relax_incar/struct-Al-3x3x3-001
21 21
struct-Al-3x3x3-000 EpA_std_err= 0.0009547408849371239
struct-Al-3x3x3-001 EpA_std_err= 0.0009156983853728134
```

上例输出为 `"reprod-opt": true` 时的输出，前两行为 LAMMPS 任务的输出路径，分别对应从 `vasp` 任务类型的优化路径中提取了 18 和 21 帧构象。后两行为每条路径上， `deepmd` 结果相比于 `vasp` 结果的构象平均原子平均势能标准差 = $\sum_i^n (E_{ideepmd} - E_{i,vasp}) / (n*m)$，其中，n 为每条路径上的构象数目，m为一个构象的原子数。

`"reprod-opt": false` 时类似 `03.vacancy` 的输出。

05.surface

```
conf_dir:./confs/Al/std-fcc
Miller_Indices:         Surf_E(J/m^2) EpA(eV) equi_EpA(eV)
struct-000-m1.1.1m:        0.817     -3.601   -3.746
struct-001-m2.2.1m:        0.990     -3.577   -3.746
struct-002-m1.1.0m:        0.945     -3.552   -3.746
struct-003-m2.2.-1m:       0.992     -3.557   -3.746
struct-004-m2.1.1m:        1.023     -3.560   -3.746
struct-005-m2.1.-1m:       1.060     -3.543   -3.746
struct-006-m2.1.-2m:       1.035     -3.550   -3.746
struct-007-m2.0.-1m:       0.958     -3.567   -3.746
struct-008-m2.-1.-1m:      0.949     -3.576   -3.746
```

字段	数据结构	例	描述
Miller_Indices	字符串	struct-000-m1.1.1m	生成表面的miller指数
Surf_E(J/m^2)	实数	0.817	表面形成能 = (EpA* - equi_EpA)*#表面态构象的总原子数/表面面积
EpA(eV)	实数	-3.628	表面态的原子平均势能
equi_EpA	实数	-3.747	平衡态的原子平均势能

FAQ & tips

0. 快速安装 DP-GEN 与 例子文件概览：
   在linux系统下(Win10 可使用 Windows 应用商店下载的 linux 子程序)，推荐使用 conda 为 DP-GEN 创建环境：

   ```
   conda create -n dpgenDev python=3.7
   ```

   激活创建的环境:

   ```
   conda activate dpgenDev
   ```

   使用 git 在当前目录下载 DP-GEN:

   ```
   git clone https://github.com/deepmodeling/dpgen.git
   ```

   进入 DP-GEN 文件夹：

   ```
   cd dpgen
   ```

   使用pip在当前目录进行安装(默认为稳定的master分支版本)

   ```
   pip install .
   ```

   查看是否安装成功：

   ```
   dpgen -h
   ```

   DP-GEN 软件在开发过程中有两个分支，分别为 master 与 devel。

- devel为实时更新的开发板，所有的功能特性更新，bug修复都会首先出现在这一分支上，是世界各地开发者共同维护的最新版本。
- 每隔一段时间，devel分支上经过测试可以稳定运行的近期更新就会合并进入master分支。 master分支为世界各地用户提供可以稳定使用的定期更新版本。

如果用户根据需求希望切换到 devel 分支的最新版，可以使用 git 切换本地代码分支并更新代码，随后重新安装，完成本地 DP-GEN 的更新：

```
git branch
```

可以看到本地代码有两个分支：

```
devel
* master
```

切换至 devel 分支：

```
git checkout devel
```

同步该分支的本地代码为 github 上的最新版本

```
git pull
```

重新安装，跟新软件版本为 devel 分支最新版：

```
git install .
```

完成。
如用户希望退回至 master 的本地稳定版，即可将分支切换回 master 后重新安装即可：

```
git checkout master
git install .
```

master分支的本地版本也可类似地使用 git pull 后重新安装来更新至最近的版本。

查看例子文件：

```
cd ....../somewhere/dpgen
cd examples
ls
```

```
database  init  machine  run  simplify  test
```

各文件夹内为 DP-GEN 所支持各 sub-conmmand 的例子文件。
例子文件目录中的次级目录名字及各文件的名字对应所适用的软件、软件版本号与研究系统等：

```
tree */
```

```
database
└── param_Ti.json
init
├── al.json
├── al.POSCAR
├── al.yaml
├── ch4.json
├── CH4.POSCAR
├── ch4.yaml
├── reaction.json
├── surf.json
└── surf.yaml
machine
├── bk
│   ├── machine-hnu.json
│   ├── machine-tiger.json
│   ├── machine-tiger-pwscf-della.json
│   ├── machine-tiger-vasp-della.json
│   └── machine-ucloud.json
├── DeePMD-kit-0.12
│   ├── machine-aws.json
│   ├── machine-local.json
│   ├── machine-lsf.json
│   ├── machine-slurm-vasp-multi.json
│   ├── machine-slurm-vasp-multi.yaml
│   ├── machine-slurm-vasp-single.json
│   └── machine-slurm-vasp-single.yaml
└── DeePMD-kit-1.0
    ├── machine-local.json
    └── machine-pbs-gaussian.json
run
├── bk
│   ├── param-h2oscan-vasp.json
│   ├── param-mg-vasp.json
│   ├── param-mg-vasp-ucloud.json
│   └── param-pyridine-pwscf.json
├── dp-lammps-cp2k
│   └── CH4
│       ├── param_CH4.json
│       └── param_CH4.yaml
├── dp_lammps_gaussian
│   └── dodecane
│       └── dodecane.json
├── dp-lammps-pwmat
│   ├── machine-slurm-pwmat-single.json
│   └── param_CH4.json
├── dp-lammps-siesta
│   └── dp-lammps-siesta
│       └── CH4
│           ├── param_CH4.json
│           └── param_CH4.yaml
├── dp-lammps-vasp
│   ├── Al
│   │   ├── param_al_all_gpu-deepmd-kit-1.1.0.json
│   │   ├── param_al_all_gpu.json
│   │   └── param_al_all_gpu.yaml
│   └── CH4
│       ├── INCAR_methane
│       ├── param_CH4_deepmd-kit-1.1.0.json
│       ├── param_CH4.json
│       └── param_CH4.yaml
└── dp-lammps-vasp-et
    └── param_elet.json
simplify
└── qm7.json
test
├── deepmd_param.json
├── meam_param.json
├── vasp_param_from_incar.json
└── vasp_param.json
```

1. 若在init步骤使用多个单元素 POTCAR 文件连结成的单一 POTCAR文件，请注意连结元素顺序须与 `param.json` 中的 `elements` 列表中的元素顺序一致。
   如甲烷例 `param.json`： `“elements”: ["H", "C"]`
   POTCAR须按序连结： `cat /somewhere/POTCAR_H /somewhere/POTCAR_C > POTCAR`

   由于上述方法并不方便且容易在操作过程中产生问题, 因此**并不推荐**如此准备 POTCAR 文件。
   推荐 在各步骤均在相应 `param.json` 的关键词（列表）处，按顺序声明多个单元素 POTCAR_element 文件，且注意与 `type_map` `mass_map` 关键词

中的元素顺序保持一致:

init:

```
"potcars": ["somewhere/POTCAR_H",
            "somewhere/POTCAR_C"],
```

run:

```
"fp_pp_path": "/somewhere",
"fp_pp_files": ["POTCAR_H","POTCAR_C"],
```

test:

```
"potcar_map": {"H": "/somewhere/POTCAR_H",
               "C": "/somewhere/POTCAR_C"},
```

2. 请注意 `param.json` 中相关联的关键词应保持一致，如：
   - `init_data_sys` 与 `init_batch_size` 列表中元素的个数。
   - `sys_configs` 与 `sys_batch_size` 列表中元素的个数。
   - `sel` 中元素对应的原子种类顺序与体系中实际的原子种类顺序。
   - `sys_configs` 与 `sys_idx` 中的序数等。
3. 请仔细检查 `sys_configs` 指定的路径，如果 `01.model_devi` 中缺失了部分或全部 POSCAR 文件，有可能是这里出现了问题.
4. `sys_batch_size` 与 `init_batch_size` 设置的推荐原则是 `batch_size` 值乘以构象的原子数应该大于 32。然而，过大的 `batch_size` 可能会导致 GPU 显存不足。
5. 对训练参数，一般可令 `stop_batch` = 200* `decay_steps` 。
6. 使用 VASP 进行 `02.fp` 第一性原理计算时，请求的总CPU核数 `task_per_node` 应该能够被 `INCAR` 文件中 `npar` 与 `kpar` 之积整除。使用单个节点进行计算时，令 `task_per_node` / `npar` = 单颗 CPU 中的核心数可能在 `task_per_node` 取值较大时提供较好的计算性能。