TensorAlloy Manual

- Author: Xin Chen
- Email: Bismarrck@me.com

- Date: Dec 23, 2018
- **TensorAlloy** version: 1.0

1. Program

The `tensoralloy` program has two subcommands:

- `build`: build a *sqlite3* database from a *xyz* or *extxyz* file.
- `run`: run an experiment from a TOML input file.

1.1 Subcommand: `build`

The `build` comamnd is used to build a ase.db.

Usage

```
tensoralloy build filename [-h]
                           [--num-examples NUM_EXAMPLES]
                           [--energy-unit ENERGY_UNIT]
                           [--forces-unit FORCES_UNIT]
                           [--stress-unit STRESS_UNIT]
```

Args

- `filename`: a *xyz* file or an *extxyz* file. Please refer to ase.io.write for more information.
- `--num-examples`: the total number of examples to read. If not set, all structures in the file will be read.
- `--energy-unit`: the unit of the total energies, possible options: *eV* (default), *Hartree* or *kcal/mol*.
- `--forces-unit`: the unit of the atomic forces, possible options: *eV/Angstrom* (default), *eV/Bohr*, *Hartree/Angstrom*, *Hartree/Bohr*, *kcal/mol/Angstrom* or *kcal/mol/Bohr*.
- `--stress-unit`: the unit of the stress tensors, possible options: *GPa* (default) or *kbar*.

Example

```
tensoralloy build datasets/qm7.xyz --energy-unit=Hartree
```

1.2 Subcommand: `run`

The subcommand `run` is used to run an experiment given a TOML input file. Section 2 will introduce the input file.

Usage

```
tensoralloy run input_file
```

Args

- `input_file`: a TOML format input file to read.

Example

```
export PYTHONPATH=`pwd`:$PYTHONPATH
cd test_files/inputs
tensoralloy run qm7.behler.k2.toml
```

2. Input

This section defines settings of the input TOML file.

Below is the basic unit of TOML where *key* represents an option of *section* and *val* is value of *key*.

```
[section]
key = val
```

There are two types of keys: *required* and *optional*. If a key is marked *required*, its value must be provided by user in the input file. Optional keys can be ignored.

2.1 Section: `[dataset]`

- `sqlite3`: the ase.db to use. This database should be generated by command `tensoralloy build`.
  - type: string
  - required
- `name`: the name of this experiment.
  - type: string
  - required
  - **note: the character '-' is not allowed.**
- `descriptor`: the descriptor to describe this dataset. Available options are *eam* and *behler*.
  - type: string
  - required
- `rc`: the cutoff radius, in Angstroms.
  - type: float
  - default: 6.5
- `tfrecords_dir`: the directory to save/load generated tfrecords files.
  - type: string
  - default: "."

- `test_size`: the size of the evalutaion dataset. This can be an `int` (absolute size, e.g. 1000) or a `float` (fraction size, e.g. 0.2).
    - type: int or float
    - default: 0.2

2.2 Section: `[nn]`

- `activation`: the activation function to use, *leaky_relu*, *relu*, *elu*, *tanh* or *sigmoid*.
    - type: string
    - default: 'leaky_relu'
- `l2_weight`: the weight of the L2 regularization term. If zero, L2 regularization will be disabled.
    - type: float
    - default: 0.0
- `minimize`: specify the structural properties to minimize. Should be a list of string. Availabel properties are: *energy*, *forces*, *stress* or *total_pressure*
    - type: List[str]
    - default: ['energy', 'forces']
- `predict`: specify the structural properties to predict. Should be a list of string. Availabel properties are the same with `minimize`.
    - type: List[str]
    - default: ['energy', 'forces']

2.3 Section: `[nn.atomic]`

Note: This section should be configured if **dataset.descriptor** is **behler**.

This section defines the settings of the atomic neural network.

- `arch`: the architecture of the atomic neural network. Current implemented archs are: *AtomicNN* and *AtomicResNN*.
    - type: string
    - default: AtomicNN
- `input_normalizer`: specify the algorithm to normalize the symmetry function descriptors. If *false*, normalization will be disabled. Availabel methods are *linear* and *arctan*.
    - type: string or false
    - default: linear

2.4 Section: `[nn.atomic.layers]`

This section defines the layers of the atomistic neural network for each type of element. The keys are elements of the datasets and values should be lists of integers.

Here is an example for Ni-Mo alloy:

```
[nn.atomic.layers]
Ni = [128, 64, 32]
Mo = [64, 32]
```

In this example, `Ni` will be described by an NN with **three** hidden layers and the correponding hidden layer sizes are 128, 64 and 32 while `Mo` will be described by an NN with **two** hidden layers of sizes 64 and 32.

`[64, 32]` is the default setting for all elements.

Note: This section should be configured if **dataset.descriptor** is **behler**.

2.5 Section `[behler]`

This section defines settings of Behler's symmetry function descriptor.

There are two types of symmetry functions:

1. Radial function G2:

$$G_i^2 = \sum_j{}' \exp\left(-\frac{\eta}{R_c^2} r_{ij}^2\right) f_c(r_{ij})$$

2. Angular function G4:

$$G_i^4 = 2^{1-\zeta} \sum_j{}' (1 + \gamma \cos\theta_{ijk}) \exp\left(-\frac{\beta}{R_c^2}(r_{ij}^2 + r_{ik}^2 + r_{jk}^2)\right) f_c(r_{ij}) f_c(r_{ik}) f_c(r_{jk})$$

and $f_c(r)$ is the cosine cutoff function:

$$f_c(r) = \begin{cases} \frac{1}{2}\left(\cos\left(\frac{r}{R_c}\pi\right) + 1\right) & r \le R_c \\ 0.0 & r > R_c \end{cases}$$

- `eta`: the parameters η for the radial function G2.
  - type: List[float]
  - default: [0.01, 0.4, 2.0, 10.0]
- `beta`: the parameters β for the angular function G4.
  - type: List[float]
  - default:
- `gamma`: the parameters γ for the angular function G4.
  - type: List[float]
  - default:
- `zeta`: the parameters ζ for the angular function G4.
  - type: List[float]
  - default:

Note: This section should be configured if **dataset.descriptor** is **behler**.

2.6 Section: `[nn.eam]`

This section defines the settings of nn-EAM.

- `arch`: the architecture of the nn-EAM. Availabel archs are: *EamAlloyNN* and *EamFsNN*. *EamAlloyNN* represents the LAMMPS/eam/alloy potential and *EamFsNN* represents the LAMMPS/eam/fs (Finnis-Sinclair) potential.

- type: string
- default: `EamAlloyNN`

Note: This section should be configured if **dataset.descriptor** is **eam**.

2.7 Section: `[nn.eam.rho]`

This section defines the settings of the electron density functions of nn-EAM.

2.7.1 `EamAlloyNN`

Style `eam/alloy` computes pairwise interactions for metals and metal alloys using embedded-atom method (EAM) potentials (Baskes). The total energy E_i of an atom i is given by:

$$E_i = F_\alpha\left(\sum_j{}' \rho_\beta(r_{ij})\right) + \frac{1}{2}\sum_j{}' \phi_{\alpha\beta}(r_{ij})$$

where F_α is the embedding energy which is a function of the atomic electron density ρ, ϕ is a pair potential interaction, and α and β are the element types of atoms i and j. The multi-body nature of the EAM potential is a result of the embedding energy term. Both summations in the formula are over all neighbors j of atom i within the cutoff distance.

Here is an example for AlCu alloy:

```
[nn.eam.rho]
Al = "zjw04"
Cu = [128, 64]
```

The electron density function of Al is zjw04 while an NN with **two** layers is used to describe the electron density of Cu.

Available eam/alloy empirical potentials

- zjw04: Al, Cu
- sutton90: Ag

2.7.2 `EamFsNN`

Style `eam/fs` computes pairwise interactions for metals and metal alloys using a generalized form of EAM potentials due to Finnis and Sinclair (Finnis). The total energy E_i of an atom i is given by

$$E_i = F_\alpha\left(\sum_j{}' \rho_{\alpha\beta}(r_{ij})\right) + \frac{1}{2}\sum_j{}' \phi_{\alpha\beta}(r_{ij})$$

where $\sum_j{}'$ means the summation goes over all possible j except $j = i$.

This has the same form as the EAM formula above, except that $\rho_{\alpha\beta}$ is now a functional specific to the atomic types of both atoms i and j, so that different elements can contribute differently to the total electron density at an atomic site depending on the identity of the element at that atomic site.

Here is an example:

```
[nn.eam.rho]
AlAl = "msah11"
AlFe = [128, 64]
FeFe = "msah11"
FeAl = [128, 64]
```

The electron density of Al-Al and Fe-Fe are described by msah11 while Al-Fe and Fe-Al are described by NNs.

Available eam/alloy empirical potentials

- msah11: Al, Fe

Note: This section should be configured if **dataset.descriptor** is **eam**.

2.8 Section: `[nn.eam.phi]`

This section defines the settings of the pairwise potential functions of nn-EAM.

Here is an example:

```
[nn.eam.phi]
AlAl = "zjw04"
CuCu = "zjw04"
AlCu = [128, 64]
```

The pairwise potentials of Al-Al and Cu-Cu are zjw04 while Al-Cu is described by an NN.

Note: This section should be configured if **dataset.descriptor** is **eam**.

2.9 Section: `[nn.eam.embed]`

This section defines the settings of the embedding energy functions of nn-EAM.

```
[nn.eam.phi]
Al = "zjw04"
Cu = [128, 64]
```

The embedding function of Al is zjw04 while Cu is described by an NN.

Note: This section should be configured if **dataset.descriptor** is **eam**.

2.10 Section: `[nn.eam.export]`

This section defines the settings for exporting a *setfl* potential.

- `nr`: the number of `r` used to describe density and pair potentials.

    - type: int
    - default: 10000
- `dr`: the delta `r` used for tabulating density and pair potentials.
    - type: float
    - default: 0.00065
- `nrho`: the number of `rho` used to describe embedding functions.
    - type: int
    - default: 10000
- `drho`: the delta `rho` used for tabulating embedding functions.
    - type: float
    - default: 0.001

Note: This section should be configured if **dataset.descriptor** is **eam**.

2.11 Section: `[opt]`

This section defines the setting for SGD optimization.

- `method`: the optimization method to use. Supported algorithms are: *adam*, *rmsprop*, *adadelta*. See [tf.train] module for more information.
    - type: string
    - default: adam
- `learning_rate`: the initial learning rate.
    - type: float
    - default: 0.01
- `decay_function`: the learning rate decay function to use. If false, learning rate decay will be disabled. Supported decay functions are: *exponential*, *inverse_time* and *natural_exp*. See [tf.train] module for more information.
    - type: string or false
    - default: false
- `decay_rate`: the decay rate. See the computation detail of the decay function. `0.0 < decay_rate < 1.0`.
    - type: float
    - default: 0.95
- `decay_steps`: the decay steps. See the computation detail of the decay function. Must be positive.
    - type: int
    - default: 1000
- `staircase`: if true, decay the learning rate at discrete intervals.
    - type: string or false
    - default: false

2.12 Section: `[train]`

This section defines settings for training a model.

- `restart`: a boolean. If true, restart the previous training if possible. Otherwise, files in `model_dir` will be deleted.
    - type: boolean

    - default: false
- `batch_size`: the size of one batch.
    - type: int
    - default: 50
- `shuffle`: a boolean indicating whether the dataset should be shuffled or not. In most cases this should be set to true.
    - type: boolean
    - default: false
- `model_dir`: directory where model parameters, graph, etc are saved.
    - type: string
    - required
- `max_checkpoints_to_keep`: the maximum number of recent checkpoint files to keep. As new files are created, older files are deleted. If zero, all checkpoint files are kept.
    - type: int
    - default: 20
- `train_steps`: positive number of total steps for which to train model.
    - type: int
    - default: 10000
- `eval_steps`: positive number of steps for which to evaluate model.
    - type: int
    - default: 1000
- `summary_steps`: save summaries every this many steps.
    - type: int
    - default: 100
- `log_steps`: log key tensors and training progress every this many steps.
    - type: int
    - default: 100
- `profile_steps`: save profile traces every N steps.
    - type: int
    - default: 2000