TensorAlloy

TensorAlloy is a TensorFlow based machine learning framework for metal alloys. **TensorAlloy** builds direct computation graph from atomic positions to total energy. Thus, atomic forces and the virial stress tensor can be derived by the **AutoGrad** module of TensorFlow directly.

1. Requirements

- Python>=3.7.0
- TensorFlow==1.13.1
- scikit-learn
- scipy
- numpy
- ase>=3.18.0
- matplotlib>=3.0.0
- toml==0.10.0
- joblib
- wheel

Anaconda3 can install above packages without pain. However, the performance of conda-provided tensorflow is not that good.

Natively compiled TensorFlow, with all CPU features (SSE, AVX, etc.) enabled, is strongly recommended.

Note: `[prefix]` indicates the top-level directory where the program is unzipped.

2. Training

There are two training examples:

1. QM7
    - This example starts from the raw `extxyz` file. The first step is building the SQLITE3 database.
    - Angular symmetry functions are enabled.
2. SNAP/Ni-Mo
    - The binary alloy dataset.
    - Only radial symmetry functions are used by default.

2.1 Usage

TOML is the configuration file format used by TensorAlloy. All the necessary keys are included in the two examples. Default keys and values can be found in default.toml.

A command-line program tensoralloy is provided. Add the directory tools to `PATH` to use this program:

```
export PATH=[prefix]/tensoralloy/tools:$PATH
chmod +x [prefix]/tensoralloy/tools/tensoralloy
```

Here are some key commands:

- `tensoralloy build database [extxyz]`: build a database from an `extxyz` file.
- `tensoralloy run [input.toml]`: run an experiment from a `toml` input file.
- `tensoralloy print [logfile]`: print the evaluation results from a logfile.
- `tensoralloy --help`: print the help messages.

The `run.sh` in qm7 and snap give more details about these commands.

2.2 Output

After the training, a binary `pb` file (the trained model) will be exported to the `model_dir` specified in the input toml file. This exported `pb` file can be used by the ASE-style `TensorAlloyCalculator`.

We also provide three pre-trained models:

- Ni: energy, force
- Mo: energy, force, stress
- Ni-Mo: energy, force, stress

3. Prediction

To use the `TensorAlloyCalculator` calculator, `PYTHONPATH` should be configured first:

```
export PYTHONPATH=[prefix]:${PYTHONPATH}
```

To calculate properties of an arbitrary Ni-Mo structure, simply do:

```
#!coding=utf-8
""" Simple usage. """
from tensoralloy.calculator import TensorAlloyCalculator
from ase.build import bulk
from ase.units import GPa

calc = TensorAlloyCalculator("NiMo.pb")
atoms = bulk("Ni", cubic=True)
atoms.calc = calc
print(atoms.get_total_energy())
print(atoms.get_forces())
print(atoms.get_stress() / GPa)
```

4. Input

In this section we will introduce the options and values of the input toml file.

4.1 Root

- `precision`: `medium` (float32) or `high` (float64). Default is `medium`.
- `seed`: the global seed. Default is 611.

4.2 Dataset

- `dataset.sqlite3`: a string, the ASE Sqlite3 Database to use.
- `dataset.name`: a string, the name of this experiment.
- `dataset.rc`: a float, the cutoff radius.
- `dataset.tfrecords_dir`: a string, the directory to save/load tfrecords files.
- `dataset.test_size`: an integer, the number of examples for evaluation.
- `dataset.serial`: a boolean. Default is `false`. Typically this should be false so that structures in the sqlite3 database can be written to tfrecords in parallel. For some old Linux systems or debugging, this can be set to `true`.

4.3 NN

4.3.1 General options

- `nn.activation`: a string, the activation function to use. Options are: `elu`, `leaky_relu`, `softplus`, `softsign`, `tanh`, `sigmoid`.
- `nn.minimize`: a list of string, the properties to minimize. Some examples: `["energy", "forces"]`, `["energy", "stress"]`
- `nn.export`: a list of string, the properties that the exported model should predict. This option has the same format with `nn.minimize`.

4.3.2 Loss options

- `nn.loss.energy.weight`: a float, the weight of the energy loss. Default is 1.

- `nn.loss.energy.per_atom_loss`: a boolean. If true, per-atom energy loss will be used. Default is false.

- `nn.loss.energy.method`: a string, `"rmse"` or `"logcosh"`.

- `nn.loss.forces.weight`: a float, the weight of the force loss. Default is 1.

- `nn.loss.forces.method`: a string, `"rmse"` or `"logcosh"`.

- `nn.loss.stress.weight`: a float, the weight of the force loss. Default is 1.

- `nn.loss.stress.method`: a string, `"rmse"` or `"logcosh"`.

- `nn.loss.l2.weight`: a float, the weight of the L2 regularization. Default is

0. **Note: adaptive optimizers (adam, nadam, etc) are not compatible with L2. If you use any adaptive optimizer, please set this to 0.**

- `nn.loss.l2.decayed`: a boolean. If True, the L2 weight will decay exponentially.
- `nn.loss.l2.decay_rate`: a float controls the decay rate.
- `nn.loss.l2.decay_steps`: an integer.

4.3.3 Neural network options

- `nn.atomic.arch`: a string, the architecture of the atomistic neural networks. There two options in this package: `"AtomicNN"` or `"AtomicResNN"`. The former is the traditional type and later is the modified version proposed by the TensorAlloy paper.
- `nn.atomic.kernel_initializer`: a string, the initialization method.
- `nn.atomic.minmax_scale`: a boolean. If True, the min-max normalization of the input features will be enabled. Default is true.
- `nn.atomic.resnet.fixed_static_energy`: a boolean. If `nn.atomic.arch` is `AtomicResNet` and this is true, the static energy parameters will be fixed.
- `nn.atomic.behler.eta`: a list of float, the eta values for radial symmetry functions.
- `nn.atomic.behler.omega`: a list of float, the omega values for radial symmetry functions.
- `nn.atomic.behler.beta`: a list of float, the beta values for angular symmetry functions.
- `nn.atomic.behler.gamma`: a list of float, the gamma values for angular symmetry functions.
- `nn.atomic.behler.zeta`: a list of float, the zeta values for angular symmetry functions.
- `nn.atomic.behler.angular`: a boolean. If true, angular symmetry functions will be used. Otherwise `gamma`, `zeta` and `beta` will be ignored.

4.3.4 Layers

The following section demonstrates how to setup the hidden layers of the atomistic neural network for element **C**. This block is optional. The default setting is `[64, 32]`.

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

4.4 Opt

- `opt.method`: a string, the optimizer to used. Options are: `adam`, `nadam`, `sgd`, `adadelta`, `rmsprop`. Default is `adam`.
- `opt.learning_rate`: a float, the initial learning rate. Default is 0.01.
- `opt.decay_function`: a string, the decay function to use. Options are: `exponential`, `inverse_time`, `natural_exp`. Default is `false` which means learning rate decay is disabled.
- `opt.decay_rate`: a float, the decay rate.
- `opt.decay_steps`: an integer.
- `opt.staircase`: a boolean.

4.5 Train

- `train.reset_global_step`: a boolean. If True, the global step will be set to 0 by force. If you want to continue a previous training, this should be changed to `false`.
- `train.batch_size`: an integer, the batch size. Default is 50.
- `train.shuffle`: a boolean. If True, the input dataset will be shuffled. In most cases this should be true.
- `train.model_dir`: a string, the working directory for this experiment.
- `train.train_steps`: an integer, the maximum training steps.
- `train.eval_steps`: an integer, the intervals between two evaluations.
- `train.summary_steps`: an integer, the intervals between two writing summary operations.
- `train.log_steps`: an integer, the intervals between two logging operations.
- `train.profile_steps`: an integer, the intervals between two performance profiling operations. Set this to 0 to disable profiling.
- `train.ckpt.checkpoint_filename`: a string, the previous checkpoint file to read. Default is `false`.
- `train.ckpt.use_ema_variables`: a boolean. If `train.ckpt.checkpoint_filename` is provided, the moving average variables will be loaded if this is true.
- `train.ckpt.restore_all_variables`: a boolean. It only works when `train.ckpt.checkpoint_filename` is provided. If this is true, all variables (including the optimizer-specific variables) will be loaded. If false, only model variables will be used. Typically, if you want to continue a previous training with exactly the same settings but more training steps, set `train.reset_global_step` to false and this option to true. If you want to use the checkpoint values as initial guesses, this option shall be false.

5. License

This TensorAlloy program is licensed under GNU Lesser General Public License v3.0. For more information please read LICENSE.