AlphaCryo4D v0.1.1 Documentation

AlphaCryo4D is an open-source free software released under GNU General Public LICENSE that implements 3D classification of single-particle cryo-EM data using deep manifold learning and novel energy-based particle voting methods.

AlphaCryo4D v0.1.1 is currently a prototype, which is still under development. Users are free to use and modify the source code, providing their compliance with the GPL and that any publication making use of this software shall cite the following reference or its formally published form:

Zhaolong Wu, Enbo Chen, Shuwen Zhang, Yinping Ma, Congcong Liu, Chang-Cheng Yin, Youdong Mao. Visualizing conformational space of functional biomolecular complexes by deep manifold learning. bioRxiv preprint doi: https://doi.org/10.1101/2021.08.09.455739.

Installation

1. Installation of EMAN2 and RELION are required before using AlphaCryo4D according to the websites https://github.com/cryoem/eman2 and https://github.com/3dem/relion, respectively.

2. Download the source code:
git clone https://github.com/AlphaCryo4D/AlphaCryo4D.git
cd AlphaCryo4D/

3. Create the conda environment:
conda create -n AlphaCryo4D python=3.7.1

4. Activate the environment:
source activate AlphaCryo4D

5. Install the dependencies:
conda install --yes --file EnvConda.txt
pip install -r EnvPip.txt

Programs and Scripts

Descriptions of integrated scripts

Quick start with AlphaCryo4D

1. Resample/Star.py

Purpose: Prepare the star files for 3D volume resampling in an integrated way. This script integrates the scripts of ‘Resample/randsf.py’ and ‘Resample/resample.py’.

Usage:
python Resample/Star.py --star $star_fn --number $nr_prtl --relion $relion_v --fold $m_fold

Control parameters:
--star, -s <particles.star>
    Star file of all particles
--number, -n <100000>
    Particle number of each sub-dataset
--relion, -r <2>
    Relion version
--fold, -f <3>
    Fold of data augmentation

Input: The star file of all particles $star_fn

Output: Folders of the dataset in batches started with ‘batch’, with a star file started with ‘batch’ and $m_fold + 1 resampled star files started with ‘union’ under each folder

2. Auto4D.py

Purpose: Perform the deep manifold learning calculation in an integrated way. This script integrates the scripts of ‘Resample/bigdata.py’, ‘DeepFeature/run_prepare.py’, ‘DeepFeature/run_resnet.py’, ‘DeepFeature/run_predict.py’, ‘ManifoldLandscape/tsne_prepare.py’, ‘ManifoldLandscape/tsne_rd.py’, ‘ManifoldLandscape/enumerate.sh’, ‘ManifoldLandscape/string_method.py’ and ‘ParticleVoting/clustering.py’.

Usage:
python Auto4D.py --folder $maps_path --stars $stars_path --std $maps_std --epoch $nr_epoch --batchsizetrain $batch_size_train --validationsize $validation_size --regularization $coef_reg --gpu $gpu --batchsizepredict $batch_size_predict --seed $random_seed --perplexity $perplexity --niter $niter --outputtsne $output_fn --landscape $tsne_fn --number $num_fn --range $range --start $start_point --end $end_point --stepmax $max_step --stepsize $step_size --tolerance $tolerance --interpolate $interpolate --kcenters $nr_centers --radius $radius

Control parameters:

--folder, -f <./maps_aligned/>
    Folder of maps
--stars, -a <./star/>
    Path of starfiles
--std, -s <True>
    If standardize the maps
--epoch, -e <50>
    Training epochs
--batchsizetrain, -b <8>
    Training batch size
--validationsize, -v <800>
    Size of validation data when training
--regularization, -r <0>
    L2 regularization coefficient when training
--gpu, -g <0,1,2,3>
    Gpu utilization
--batchsizepredict, -B <8>
    Predicting batch size
--seed, -S <0>
    Random seed of t-SNE
--perplexity, -p <30.0>
    Perplexity of t-SNE
--niter, -N <1000>
    Maximum number of iterations in t-SNE
--outputtsne, -o <output.npy>
    Output of t-SNE
--landscape, -l <output.npy>
    Free energy landscape points mapping by t-sne
--number, -n <num.txt>
    Particle number file
--range, -G <100.0>
    Maximum value of the reaction coordinate of free energy landscape
--start, -A <-80.0 -80.0>
    Position around start point of transition path
--end, -E <80.0 80.0>
    Position around end point of transition path
--stepmax, -m <100000>
    Maximum steps in String method algorithm
--stepsize, -i <1e-1>
    Step size in String method algorithm
--tolerance, -t <1e-7>
    Convergence condition in String method algorithm
--interpolate, -I <linear>
    Interplotion method used in free energy landscape plotting, linear or cubic
--kcenters, -k <20>
    K clustering centers along transition path
--radius, -u <30.0>
    Clustering radius along transition path

Input: The folder of resampled maps $maps_path and path of corresponding star files $stars_path

Output: Merged 3D volume data ‘rdata.npy’ and ‘rdata_3d.npy’ with its number file $num_fn and list file ‘data.log’, free energy landscape ‘el.pdf’, clustering centers ‘centers_k.npy’, and three folders ‘Deep’, ‘Manifold’ and ‘Vote’ (‘Deep’ containing preprocessed input data for 3D residual autoencoder ‘data_dl.npy’, loss function change in training process ‘train_process.npz’, folder of the final network model ‘model’, folder of the best network model ‘checkpoint’ and folder of 3D feature maps ‘result’, ‘Manifold’ containing preprocessed input data for manifold learning ‘input.npy’, low-dimension mapping $tsne_fn and corresponding plot ‘tsne.png’, ‘Vote’ containing folders of each classes started with ‘c’)

3. ParticleVoting/Vote.py

Purpose: Conduct particle voting in an integrated way. This script integrates the scripts of ‘ParticleVoting/vote_prepare.sh’, ‘ParticleVoting/gethead.py’ and ‘ParticleVoting/post_and_f.sh’.

Usage:
python ParticleVoting/Vote.py --classes $classes_fn --head $head_fn --relion $relion_v --threshold $voting_threshold

Control parameters:
--classes, -c <c1_small>
    Classes to vote
--head, -he <head.star>
    Head of star file
--relion, -r <2>
    Relion version
--threshold, -t <2>
    Threshold for voting

Input: The classes of star files to be voted $classes_fn

Output: Log and star files before particle voting (‘pre_vote.log’ and ‘pre_vote.star’) and after particle voting (‘post_vote.log’ and ‘post_and.star’)

Details of each script step by step

Resample Directory: 3D resample with M-fold data augmentation and particle shuffling

1. randsf.py

Purpose: Split the star file of all particles into several sub-datasets.

Usage:
python Resample/randsf.py --star $star_fn --number $nr_prtl

Control parameters:
--star, -s <particles.star>
    Star file of all particles
--number, -n <100000>
    Particle number of each sub-dataset

Input: The star file of all particles $star_fn

Output: Star files in batches started with ‘batch’

2. resample.py

Purpose: Shuffle each sub-dataset by M times to generate resampled 3D volumes.

Usage:
python Resample/resample.py --star $batch_fn --fold $m_fold

Control parameters:
--star, -s <batch1.star>
    Star file of a batch of dataset
--fold, -f <3>
    Fold of data augmentation

Input: The star file of a batch $batch_fn

Output: Resampled star files started with ‘union’

3. link.sh

Purpose: Link the star and mrc files of 3D resampled volumes into two folders.

Usage:
bash Resample/link.sh

Control parameters:
Set the fold identifier ur (the 7th line of the script), batch number br (the 8th line of the script), class number cr (the 9th line of the script) and your path of star (the 18th line of the script) and mrc files (the 14th line of the script)

Input: The original star files and mrc files

Output: Linked files of the star files and mrc files

4. fit.sh

Purpose: Align all 3D density maps to a common reference map.

Usage:
bash Resample/fit.sh

Control parameters:
Set your path of all 3D volumes (the 4th line of the script) and the reference map (the 10th line of the script)

Input: All 3D volumes

Output: Aligned 3D volumes with the reference map

5. bigdata.py

Purpose: Prepare the 3D volume dataset after resampling.

Usage:
python Resample/bigdata.py --folder $maps_path --std $maps_std

Control parameters:
--folder, -f <./maps_aligned/>
    Folder of maps
--std, -s <True>
    If standardize the maps

Input: All aligned 3D volumes

Output: 3D volume data ‘rdata.npy’ and ‘rdata_3d.npy’, with their order recorded in ‘data.log’

DeepFeature Directory: extract 3D feature by 3D deep residual autoencoder

6. run_prepare.py

Purpose: Preprocess the input data for 3D deep neural network.

Usage:
python DeepFeature/run_prepare.py --data $data_fn

Control parameters:
--data, -d <rdata_3d.npy>
    3D data stack

Input: The 3D volume data $data_fn

Output: Preprocessed input data for 3D residual autoencoder ‘data_dl.npy’

7. run_resnet.py

Purpose: Train the 3D residual autoencoder network.

Usage:
python DeepFeature/run_resnet.py --epoch $nr_epoch --batchsize $batch_size --validationsize $validation_size --regularization $coef_reg --data $input_fn --checkpoint $check_fn --finalmodel $finial_fn --gpu $gpu

Control parameters:
--epoch, -e <50>
    Training epochs
--batchsize, -b <8>
    Batch size of training
--validationsize, -v <800>
    Size of validation data
--regularization, -r <0>
    L2 regularization coefficient
--data, -d <data_dl.npy>
    Input 3D data
--checkpoint, -c <checkpoint/check.h5>
    Path of best model
--finalmodel, -f <model/final_model.h5>
    Path of final model
--gpu, -g <0,1,2,3>
    Gpu utilization

Input: The input data $input_fn

Output: Loss function change in training process ‘train_process.npz’, folder of the final network model $finial_fn and folder of the best network model $check_fn

8. run_predict.py

Purpose: Calculate the 3D feature maps by the autoencoder.

Usage:
python DeepFeature/run_predict.py --data $input_fn --model $model_fn --batchsize $batch_size --feature $feature_fn

Control parameters:
--data, -d <data_dl.npy>
    Input 3D data
--model, -m <checkpoint/check.h5>
    Model of network for feature extraction
--batchsize, -b <8>
    Batch size of prediction
--feature, -f <result/feature.npy>
    Output of feature engineering

Input: The input data $input_fn and the best network model $model_fn

Output: 3D feature maps $feature_fn

ManifoldLandscape Directory: manifold embedding of 3D volume data for free-energy landscape reconstitution

9. tsne_prepare.py

Purpose: Preprocess the 3D density maps and their feature maps for manifold learning.

Usage:
python ManifoldLandscape/tsne_prepare.py --data $data_fn --feature $feature_fn --std $feature_std

Control parameters:
--data, -d <rdata.npy>
    Raw data
--feature, -f <result/feature.npy>
    Deep feature
--std, -s <True>
    If true, standardize the feature maps

Input: The 3D volume data $data_fn and the 3D feature maps $feature_fn

Output: Preprocessed input data for manifold learning ‘input.npy’

10. tsne_rd.py

Purpose: Map the two-dimensional manifold embedding of input volume data by t-SNE.

Usage:
python ManifoldLandscape/tsne_rd.py --input $input_fn --output $output_fn --seed $random_seed --perplexity $perplexity --niter $niter

Control parameters:
--input, -i <input.npy>
    Input of t-SNE
--output, -o <output.npy>
    Output of t-SNE
--seed, -s <0>
    Random seed
--perplexity, -p <30.0>
    Perplexity of t-SNE
--niter, -n <1000>
    Maximum number of iterations in t-SNE

Input: The input data $input_fn

Output: Low-dimension mapping $output_fn and corresponding plot ‘tsne.png’

11. enumerate.sh

Purpose: Count the particle number of each data point.

Usage:
bash ManifoldLandscape/enumerate.sh

Control parameters:
Set your path of the star files (the 11th line of the script)

Input: The path of the star files

Output: Number file ‘num.txt’

12. string_method.py

Purpose: Compute the free energy landscape and find the minimum energy path by the string method.

Usage:
python ManifoldLandscape/string_method.py --landscape $tsne_fn --number $num_fn --range $range --start $start_point --end $end_point --stepmax $max_step --stepsize $step_size --tolerance $tolerance --interpolate $interpolate --kcenters $nr_centers

Control parameters:
--landscape, -l <output.npy>
    Free energy landscape points mapping by t-sne
--number, -n <num.txt>
    Particle number file
--range, -r <100.0>
    Maximum value of the reaction coordinate of free energy landscape
--start, -s <-80.0 -80.0>
    Position around start point of transition path
--end, -e <80.0 80.0>
    Position around end point of transition path
--stepmax, -m <100000>
    Maximum steps in String method algorithm
--stepsize, -i <1e-1>
    Step size in String method algorithm
--tolerance, -t <1e-7>
    Convergence condition in String method algorithm
--interpolate, -I <linear>
    Interplotion method used in free energy landscape plotting, linear or cubic
--kcenters, -k <20>
    K clustering centers along transition path

Input: The low-dimension mapping of t-SNE $tsne_fn and the number file $num_fn

Output: Free energy landscape ‘el.pdf’ and clustering centers ‘centers_k.npy’

ParticleVoting Directory: 3D classification with particle voting on the free energy landscape

13. clustering.py

Purpose: Find the conformer ID and distance to the nearest clustering center of the data points within one cluster.

Usage:
python ParticleVoting/clustering.py --points $tsne_fn --centers $centers_fn --radius $radius

Control parameters:
--points, -p <output.npy>
    Files of data points
--centers, -c <centers_k.npy>
    Files of clustering centers
--radius, -r <30.0>
    Clustering radius

Input: The low-dimension mapping of t-SNE $tsne_fn and clustering centers $centers_fn

Output: Folders of each classes started with ‘c’

14. vote_prepare.sh

Purpose: Prepare the star files of the points of one cluster.

Usage:
bash ParticleVoting/vote_prepare.sh $c

Control parameters:
Set the ID and distance file $c and your path of star files (the 14th line of the script)

Input: The classes of star files to be voted $c and the path of star files

Output: Linked files of the star files to be voted

15. gethead.py

Purpose: Get the header lines of these star files.

Usage:
python ParticleVoting/gethead.py --star $star_fn --head $head_fn

Control parameters:
--star, -s <u123_p1c1.star>
    Input star file
--head, -he <head.star>
    Head of the star file

Input: The input star file

Output: Head of the star file

16. post_vote_and.sh

Purpose: Vote on particles by the energy-based algorithm.

Usage:
bash ParticleVoting/post_vote_and.sh

Control parameters:
Set the threshold of voting th (the 2nd line of the script)

Input: The linked files of the star files to be voted

Output: Star files after particle voting ended with ‘post_and.star’

17. post_and_f.sh

Purpose: Classify particles quickly by the energy-based particle-voting algorithm.

Usage:
bash ParticleVoting/post_and_f.sh

Control parameters:
Set the threshold of voting th (the 2nd line of the script)

Input: The linked files of the star files to be voted

Output: Log and star files before particle voting (‘pre_vote.log’ and ‘pre_vote.star’) and after particle voting (‘post_vote.log’ and ‘post_and.star’)

18. post_or_parallel.sh

Purpose: Classify particles by the distance-based algorithm in parallel.

Usage:
bash ParticleVoting/post_or_parallel.sh

Control parameters:
Set the threshold of voting th (the 3rd line of the script) and core number nproc (the 9th line of the script)

Input: The linked files of the star files to be voted

Output: Star files after distance-based classification ‘post_or.star’

19. dedup.sh

Purpose: Remove the duplication of particles when using the distance-based algorithm.

Usage:
bash ParticleVoting/dedup.sh

Control parameters:
Set the core number nproc (the 5th line of the script)

Input: The star files containing duplicated particles

Output: Star files after removing the duplication of particles ended with ‘dedup.star’

Simulation data generation

Synthetic particles projected from NLRP3 inflammasome

The process of simulation data generation from the NLRP3 inflammasome protein is demostrated in the folder ‘Examples/Simulation/’. To generate simulation data of distinct SNRs, the atomic models of 20 hypothetical conformers in the *pdb* format are transformed into cryo-EM density maps of the *mrc* format respectively at the first stage. Each density map is added a synthetic B-factor of 100.

. pdb2mrc.sh

Then the simulated particle datasets of SNR=0.01, 0.05 and 0.005 are projected from the prepared density maps respectively. For each conformer, 100,000 projected particles are generated with noise in the simulation.

cd Particles
. projection.sh