symmys Documentation

Release 0.1.0

Matthew Spellings

May 29, 2020

Contents

CHAPTER 1

Introduction

`symmys` is an in-development library for performing symmetry detection and related tasks using tensorflow. Currently it attempts to identify rotation transformations that leave a given point cloud unchanged and distills these rotations into a set of n-fold symmetric axes.

1.1 Documentation

Browse more detailed documentation online or build the sphinx documentation from source:

```
git clone https://github.com/klarh/symmys
cd symmys/doc
pip install -r requirements.txt
make html
```

1.1.1 Optimization

class `symmys.optimization.`**`PointRotations`**(*num_rotations*, *quaternion_dim=8*, *include_inversions=True*, *loss=<function mean_exp_rsq>*)

Finds rotations that leave a point cloud unchanged up to a permutation.

This method optimizes a set of unit quaternions to match the distribution of transformed points to the set of unrotated points. Quaternions are then clustered by their axis of rotation and merged into N-fold rotation symmetries.

Parameters

- **`num_rotations`** – Number of plain rotations (and rotoinversions, if enabled) to consider
- **`quaternion_dim`** – Optimizer dimension for quaternions (higher may make optimization easier at the cost of more expensive optimization steps)
- **`include_inversions`** – If True, include rotoinversions as well as rotations

- **loss** – Loss function to use; see *symmys.losses*

build_model()

Create the tensorflow model.

This method can be replaced by child classes to experiment with different network architectures. The returned result should be a dictionary containing at least:

- *model*: a *tensorflow.keras.models.Model* instance that replicates a given set of input points
- *rotation_layer*: a layer with a *quaternions* attribute to be read
- *rotoinversion_layer* (if inversions are enabled): a layer with a *quaternions* attribute to be read

fit(*points*, *epochs=1024*, *early_stopping_steps=16*, *validation_split=0.3*, *hash_sample_N=128*, *reference_fraction=0.1*, *optimizer='adam'*, *batch_size=256*, *valid_symmetries=12*, *extra_callbacks=[]*)

Fit rotation quaternions and analyze the collective symmetries of a set of input points.

This method builds a rotation model, fits it to the given data, and groups the found quaternions by their axis and rotation angle.

After fitting, a map of symmetries will be returned: a dictionary of {N-fold: [axes]} containing all the axes about which each observed symmetry were found.

Parameters

- **points** – Input points to analyze:: (N, 3) numpy array-like sequence
- **epochs** – Maximum number of epochs to train
- **early_stopping_steps** – Patience (in epochs) for early stopping criterion; training halts when the validation set loss does not improve for this many epochs
- **validation_split** – Fraction of training data to use for calculating validation loss
- **hash_sample_N** – Minimum number of points to use as reference data for the loss function (see `hash_sample()`)
- **reference_fraction** – Fraction of given input data to be hashed to form the reference data
- **optimizer** – Tensorflow/keras optimizer name or instance
- **batch_size** – Batch size for optimization
- **valid_symmetries** – Maximum degree of symmetry (N) that will be considered when identifying N-fold rotations
- **extra_callbacks** – Additional tensorflow callbacks to use during optimization

model

Return the tensorflow model that will perform rotations.

rotation_layer

Return the tensorflow.keras layer for rotations.

rotoinversion_layer

Return the tensorflow.keras layer for rotoinversions.

1.1.2 Losses

symmys.losses.**mean_exp_rsq**(*pred*, *reference*, *r_scale=1.0*)

Returns mean(1 - exp(-R^2/r_scale^2)) for a set of reference points.

`symmys.losses.`**`mean_sqrt_rsq`**(*pred*, *reference*)

Returns mean(sqrt(R^2)) for a set of reference points.

1.1.3 Layers

class `symmys.layers.`**`QuaternionRotation`**(*num_rotations*, *quaternion_dim=6*, *include_reverse=True*, **args*, ***kwargs*)

Perform rotations of a set of input points, parameterized by unit quaternions.

This layer takes a point cloud as input and produces rotated images of all the points in the point cloud. The rotations that are applied are parameterized by unit quaternions, which are treated as layer weights to be optimized.

Quaternions are optimized in a higher dimension and then projected down through a *sum* operation to improve the speed of the optimization process.

Parameters

- **`num_rotations`** – Number of rotation quaternions to use
- **`quaternion_dim`** – Pre-projection dimension of quaternion parameters
- **`include_reverse`** – If True, also output points rotated by the conjugate quaternion for each learned quaternion

class `symmys.layers.`**`QuaternionRotoinversion`**(*num_rotations*, *quaternion_dim=6*, *include_reverse=True*, **args*, ***kwargs*)

Learn rotoinversions, rather than rotations. Otherwise identical to `QuaternionRotation`.

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

B

F

M

P

Q

R

S