sagenet

Elyas Heidari

Dec 13, 2021

MAIN

SageNet is a robust and generalizable graph neural network approach that probabilistically maps dissociated single cells from an scRNAseq dataset to their hypothetical tissue of origin using one or more reference datasets aquired by spatially resolved transcriptomics techniques. It is compatible with both high-plex imaging (e.g., seqFISH, MERFISH, etc.) and spatial barcoding (e.g., 10X visium, Slide-seq, etc.) datasets as the spatial reference.

SageNet is implemented with pytorch and pytorch-geometric to be modular, fast, and scalable. Also, it uses `anndata` to be compatible with scanpy and squidpy for pre- and post-processing steps.

CHAPTER

ONE

INSTALLATION

You can get the latest development version of our toolkit from Github using the following steps:

First, clone the repository using `git`:

```
git clone https://github.com/MarioniLab/sagenet
```

Then, `cd` to the sagenet folder and run the install command:

```
cd sagenet
python setup.py install #or pip install `
```

The dependency `torch-geometric` should be installed separately, corresponding the system specefities, look at this link for instructions.

CHAPTER

TWO

NOTEBOOKS

To see some examples of our pipeline's capability, look at the notebooks directory. The notebooks are also avaialble on google colab:

1. Intro to SageNet
2. Using multiple references

CHAPTER

THREE

INTERACTIVE EXAMPLES

See this

CHAPTER

FOUR

SUPPORT AND CONTRIBUTE

If you have a question or new architecture or a model that could be integrated into our pipeline, you can post an issue or reach us by email.

CHAPTER

FIVE

CONTRIBUTIONS

This work is led by Elyas Heidari and Shila Ghazanfar as a joint effort between MarioniLab@CRUK@EMBL-EBI and RobinsonLab@UZH.

5.1 SageNet: Single-cell Spatial Locator

SageNet is a robust and generalizable graph neural network approach that probabilistically maps dissociated single cells from an scRNAseq dataset to their hypothetical tissue of origin using one or more reference datasets aquired by spatially resolved transcriptomics techniques. It is compatible with both high-plex imaging (e.g., seqFISH, MERFISH, etc.) and spatial barcoding (e.g., 10X visium, Slide-seq, etc.) datasets as the spatial reference.

SageNet is implemented with pytorch and pytorch-geometric to be modular, fast, and scalable. Also, it uses `anndata` to be compatible with scanpy and squidpy for pre- and post-processing steps.

5.2 Installation

Note: 0.1.0

The dependency `torch-geometric` should be installed separately, corresponding the system specefities, look at this link for instructions. We recommend to use Miniconda.

5.2.1 PyPI

The easiest way to get SageNet is through pip using the following command:

```
pip install sagenet
```

5.2.2 Development

First, clone the repository using `git`:

```
git clone https://github.com/MarioniLab/sagenet
```

Then, `cd` to the sagenet folder and run the install command:

```
cd sagenet
python setup.py install #or pip install .
```

5.3 Usage

```
import sagenet as sg
import scanpy as sc
import squidpy as sq
import anndata as ad
import random
random.seed(10)
```

5.3.1 Training phase:

Input:

- Expression matrix associated with the (spatial) reference dataset (an `anndata` object)

```
adata_r = sg.datasets.seqFISH1()
```

- gene-gene interaction network

```
glasso(adata_r, [0.5, 0.75, 1])
```

- one or more partitionings of the spatial reference into distinct connected neighborhoods of cells or spots

```
adata_r.obsm['spatial'] = np.array(adata_r.obs[['x','y']])
sq.gr.spatial_neighbors(adata_r, coord_type="generic")
sc.tl.leiden(adata_r, resolution=.01, random_state=0, key_added='leiden_0.01',␣
↪adjacency=adata_r.obsp["spatial_connectivities"])
sc.tl.leiden(adata_r, resolution=.05, random_state=0, key_added='leiden_0.05',␣
↪adjacency=adata_r.obsp["spatial_connectivities"])
sc.tl.leiden(adata_r, resolution=.1, random_state=0, key_added='leiden_0.1',␣
↪adjacency=adata_r.obsp["spatial_connectivities"])
sc.tl.leiden(adata_r, resolution=.5, random_state=0, key_added='leiden_0.5',␣
↪adjacency=adata_r.obsp["spatial_connectivities"])
sc.tl.leiden(adata_r, resolution=1, random_state=0, key_added='leiden_1',␣
↪adjacency=adata_r.obsp["spatial_connectivities"])
```

Training:

```
sg_obj = sg.sage.sage(device=device)
sg_obj.add_ref(adata_r, comm_columns=['leiden_0.01', 'leiden_0.05', 'leiden_0.1',
↪'leiden_0.5', 'leiden_1'], tag='seqFISH_ref', epochs=20, verbose = False)
```

Output:

- A set of pre-trained models (one for each partitioning)

```
!mkdir models
!mkdir models/seqFISH_ref
sg_obj.save_model_as_folder('models/seqFISH_ref')
```

- A concensus scoring of spatially informativity of each gene

```
ind = np.argsort(-adata_r.var['seqFISH_ref_entropy'])[0:12]
with rc_context({'figure.figsize': (4, 4)}):
        sc.pl.spatial(adata_r, color=list(adata_r.var_names[ind]), ncols=4, spot_size=0.
↪03, legend_loc=None)
```

5.3.2 Mapping phase

Input:

- Expression matrix associated with the (dissociated) query dataset (an `anndata` object)

```
adata_q = sg.datasets.MGA()
```

Mapping:

```
sg_obj.map_query(adata_q)
```

Output:

- The reconstructed cell-cell spatial distance matrix

```
adata_q.obsm['dist_map']
```

- A concensus scoring of mapability (uncertainity of mapping) of each cell to the references

```
adata_q.obs
```

```
import anndata
dist_adata = anndata.AnnData(adata_q.obsm['dist_map'], obs = adata_q.obs)
knn_indices, knn_dists, forest = sc.neighbors.compute_neighbors_umap(dist_adata.X, n_
↪neighbors=50, metric='precomputed')
dist_adata.obsp['distances'], dist_adata.obsp['connectivities'] = sc.neighbors._compute_
↪connectivities_umap(
    knn_indices,
    knn_dists,
    dist_adata.shape[0],
    50, # change to neighbors you plan to use
)
sc.pp.neighbors(dist_adata, metric='precomputed', use_rep='X')
sc.tl.umap(dist_adata)
sc.pl.umap(dist_adata, color='cell_type', palette=celltype_colours)
```

5.4 Notebooks

To see some examples of our pipeline's capability, look at the notebooks directory. The notebooks are also avaialble on google colab:

1. Intro to SageNet
2. Using multiple references

5.5 Interactive examples

- Spatial mapping of the mouse gastrulation atlas

5.6 Support and contribute

If you have a question or new architecture or a model that could be integrated into our pipeline, you can post an issue or reach us by email.

5.7 Contributions

This work is led by Elyas Heidari and Shila Ghazanfar as a joint effort between MarioniLab@CRUK@EMBL-EBI and RobinsonLab@UZH.

5.8 API

The API reference contains detailed descriptions of the different end-user classes, functions, methods, etc.

Note: This API reference only contains end-user documentation. If you are looking to hack away at sagenet' internals, you will find more detailed comments in the source code.

- *sage*
- *classifier*
- *utils*

5.8.1 sage

class sagenet.sage.**sage**(*device='cpu'*)

Bases: `object`

A *sagenet* object.

Parameters **device** (*str, default = 'cpu'*) – the processing unit to be used in the classifiers (gpu or cpu).

Methods

add_ref(adata[, tag, comm_columns, ...])	Trains new classifiers on a reference dataset.
load_model(tag[, dir])	Loads a single pre-trained model.
load_model_as_folder([dir])	Loads pre-trained models from a directory.
map_query(adata_q)	Maps a query dataset to space using the trained models on the spatial reference(s).
save_model(tag[, dir])	Saves a single trained model.
save_model_as_folder([dir])	Saves all trained models stored in the *sagenet* object as a folder.

add_ref(*adata*, *tag=None*, *comm_columns='class_'*, *classifier='TransformerConv'*, *num_workers=0*, *batch_size=32*, *epochs=10*, *n_genes=10*, *verbose=False*)

Trains new classifiers on a reference dataset.

Parameters

- **adata** (*AnnData*) – The annotated data matrix of shape *n_obs* × *n_vars* to be used as the spatial reference. Rows correspond to cells (or spots) and columns to genes.
- **tag** (str, default = *None*) – The tag to be used for storing the trained models and the outputs in the *sagenet* object.
- **classifier** (str, default = *'TransformerConv'*) – The type of classifier to be passed to *sagenet.Classifier()*
- **comm_columns** (list of str, *'class_'*) – The columns in *adata.obs* to be used as spatial partitions.
- **num_workers** (*int*) – Non-negative. Number of workers to be passed to *torch_geometric.data.DataLoader*.
- **epochs** (*int*) – number of epochs.
- **verbose** (*boolean, default=False*) – whether to print out loss during training.

  Return

- **------** –
- **nothing.** (*Returns*) –

Notes

Trains the models and adds them to *.models* dictionery of the *sagenet* object. Also adds a new key *{tag}_entropy* to *.var* from *adata* which contains the entropy values as the importance score corresponding to each gene.

load_model(*tag*, *dir='.'*)

Loads a single pre-trained model.

Parameters

- **tag** (*str*) – Name of the trained model to be stored in the *sagenet* object.
- **dir** (*dir, defult=`'.'`*) – The input directory.

load_model_as_folder(*dir='.'*)

Loads pre-trained models from a directory.

Parameters **dir** (*dir, defult=`'.'`*) – The input directory.

map_query(*adata_q*)

Maps a query dataset to space using the trained models on the spatial reference(s).

Parameters **adata** (*AnnData*) – The annotated data matrix of shape *n_obs* × *n_vars* to be used as the query. Rows correspond to cells (or spots) and columns to genes.

Returns

Return type Returns nothing.

Notes

- Adds new key(s) *pred_{tag}_{partitioning_name}* to *.obs* from *adata* which contains the predicted partition for partitioning *{partitioning_name}*, trained by model *{tag}*.
- Adds new key(s) *ent_{tag}_{partitioning_name}* to *.obs* from *adata* which contains the uncertainity in prediction for partitioning *{partitioning_name}*, trained by model *{tag}*.
- Adds a new key *distmap* to *.obsm* from *adata* which is a sparse matrix of size *n_obs* × *n_obs* containing the reconstructed cell-to-cell spatial distance.

save_model(*tag*, *dir='.'*)

Saves a single trained model.

Parameters

- **tag** (*str*) – Name of the trained model to be saved.
- **dir** (*dir, defult=`'.'`*) – The saving directory.

save_model_as_folder(*dir='.'*)

Saves all trained models stored in the *sagenet* object as a folder.

Parameters **dir** (*dir, defult=`'.'`*) – The saving directory.

5.8.2 classifier

class sagenet.classifier.**Classifier**(*n_features*, *n_classes*, *n_hidden_GNN=[]*, *n_hidden_FC=[]*, *K=4*, *pool_K=4*, *dropout_GNN=0*, *dropout_FC=0*, *classifier='MLP'*, *lr=0.001*, *momentum=0.9*, *log_dir=None*, *device='cpu'*)

Bases: `object`

A Neural Network Classifier. A number of Graph Neural Networks (GNN) and an MLP are implemented.

Parameters

- **n_features** (*int*) – number of input features.
- **n_classes** (*int*) – number of classes.
- **n_hidden_GNN** (*list, default=[]*) – list of integers indicating sizes of GNN hidden layers.
- **n_hidden_FC** (*list, default=[]*) – list of integers indicating sizes of FC hidden layers. If a GNN is used, this indicates FC hidden layers after the GNN layers.
- **K** (*integer, default=4*) – Convolution layer filter size. Used only when *classifier* == *'Chebnet'*.
- **dropout_GNN** (*float, default=0*) – dropout rate for GNN hidden layers.

- **dropout_FC** (*float, default=0*) – dropout rate for FC hidden layers.
- **classifier** (*str, default='MLP'*) –
  - 'MLP' –> multilayer perceptron
  - 'GraphSAGE'–> GraphSAGE Network
  - 'Chebnet'–> Chebyshev spectral Graph Convolutional Network
  - 'GATConv'–> Graph Attentional Neural Network
  - 'GENConv'–> GENeralized Graph Convolution Network
  - 'GINConv'–> Graph Isoform Network
  - 'GraphConv'–> Graph Convolutional Neural Network
  - 'MFConv'–> Convolutional Networks on Graphs for Learning Molecular Fingerprints
  - 'TransformerConv'–> Graph Transformer Neural Network
- **lr** (*float, default=0.001*) – base learning rate for the SGD optimization algorithm.
- **momentum** (*float, default=0.9*) – base momentum for the SGD optimization algorithm.
- **log_dir** (*str, default=None*) – path to the log directory. Specifically, used for tensorboard logs.
- **device** (*str, default='cpu'*) – the processing unit.

See also:

Classifier.fit fits the classifier to data

Classifier.eval evaluates the classifier predictions

Methods

eval(data_loader[, verbose])	evaluates the model based on predictions
fit(data_loader, epochs[, test_dataloader, ...])	fits the classifier to the input data.
interpret(data_loader, n_features, n_classes)	interprets a trained model, by giving importance scores assigned to each feature regarding each class it uses the *IntegratedGradients* method from the package *captum* to computed class-wise feature importances and then computes entropy values to get a global importance measure.

eval(*data_loader*, *verbose=False*)

evaluates the model based on predictions

Parameters

- **test_dataloader** (*torch-geometric dataloader, default=None*) – the dataset on which the model is evaluated.
- **verbose** (*boolean, default=False*) – whether to print out loss during training.

Returns

- **accuracy** (*float*) – accuracy
- **conf_mat** (*ndarray*) – confusion matrix

- **precision** (*fload*) – weighted precision score
- **recall** (*float*) – weighted recall score
- **f1_score** (*float*) – weighted f1 score

fit(*data_loader*, *epochs*, *test_dataloader=None*, *verbose=False*)
fits the classifier to the input data.

Parameters

- **data_loader** (*torch-geometric dataloader*) – the training dataset.
- **epochs** (*int*) – number of epochs.
- **test_dataloader** (*torch-geometric dataloader, default=None*) – the test dataset on which the model is evaluated in each epoch.
- **verbose** (*boolean, default=False*) – whether to print out loss during training.

interpret(*data_loader*, *n_features*, *n_classes*)
interprets a trained model, by giving importance scores assigned to each feature regarding each class it uses the *IntegratedGradients* method from the package *captum* to computed class-wise feature importances and then computes entropy values to get a global importance measure.

Parameters

- **data_loder** (*torch-geometric dataloader, default=None*) – the dataset on which the model is evaluated.
- **n_features** (*int*) – number of features.
- **n_classes** (*int*) – number of classes.

Returns **ent**

Return type numpy ndarray, shape (n_features)

5.8.3 utils

sagenet.utils.**compute_metrics**(*y_true*, *y_pred*)
Computes prediction quality metrics.

Parameters

- **y_true** (*1d array-like, or label indicator array / sparse matrix*) – Ground truth (correct) labels.
- **y_pred** (*1d array-like, or label indicator array / sparse matrix*) – Predicted labels, as returned by a classifier.

Returns

- **accuracy** (*accuracy*)
- **conf_mat** (*confusion matrix*)
- **precision** (*weighted precision score*)
- **recall** (*weighted recall score*)
- **f1** (*weighted f1 score*)

sagenet.utils.**get_dataloader**(*graph*, *X*, *y*, *batch_size=1*, *undirected=True*, *shuffle=True*, *num_workers=0*)
Converts a graph and a dataset to a dataloader.

Parameters

- **graph** (*igraph object*) – The underlying graph to be fed to the graph neural networks.
- **X** (*numpy ndarray*) – Input dataset with columns as features and rows as observations.
- **y** (*numpy ndarray*) – Class labels.
- **batch_size** (*int, default=1*) – The batch size.
- **undirected** (*boolean*) – if the input graph is undirected (symmetric adjacency matrix).
- **shuffle** (boolean, default = *True*) – Wheather to shuffle the dataset to be passed to *torch_geometric.data.DataLoader*.
- **num_workers** (*int, default = 0*) – Non-negative. Number of workers to be passed to *torch_geometric.data.DataLoader*.

Returns

- **dataloader** (*a pytorch-geometric dataloader. All of the graphs will have the same connectivity (given by the input graph),*)
- *but the node features will be the features from X.*

sagenet.utils.**glasso**(*adata*, *alphas=5*, *n_jobs=None*, *mode='cd'*)

Recustructs the gene-gene interaction network based on gene expressions in *.X* using a guassian graphical model estimated by *glasso*.

Parameters

- **adata** (*AnnData*) – The annotated data matrix of shape *n_obs* × *n_vars*. Rows correspond to cells and columns to genes.
- **alphas** (*int or array-like of shape (n_alphas,), dtype=`float`, default=`5`*) – Non-negative. If an integer is given, it fixes the number of points on the grids of alpha to be used. If a list is given, it gives the grid to be used.
- **n_jobs** (int, default *None*) – Non-negative. number of jobs.

Returns

Return type adds an *csr_matrix* matrix under key *adj* to *.varm*.

References

Friedman, J., Hastie, T., & Tibshirani, R. (2008). Sparse inverse covariance estimation with the graphical lasso. Biostatistics, 9(3), 432-441.

sagenet.utils.**kullback_leibler_divergence**(*X*)

Finds the pairwise Kullback-Leibler divergence matrix between all rows in X.

Parameters **X** (*array_like, shape (n_samples, n_features)*) – Array of probability data. Each row must sum to 1.

Returns **D** – The Kullback-Leibler divergence matrix. A pairwise matrix D such that D_{i, j} is the divergence between the ith and jth vectors of the given matrix X.

Return type ndarray, shape (n_samples, n_samples)

Notes

Based on code from Gordon J. Berman et al. (https://github.com/gordonberman/MotionMapper)

References

Berman, G. J., Choi, D. M., Bialek, W., & Shaevitz, J. W. (2014). Mapping the stereotyped behaviour of freely moving fruit flies. Journal of The Royal Society Interface, 11(99), 20140672.

`sagenet.utils.`**`multinomial_rvs`**(*n*, *p*)

Sample from the multinomial distribution with multiple p vectors.

Parameters

- **n** (*int*) – must be a scalar >=1
- **p** (*numpy ndarray*) – must an n-dimensional he last axis of p holds the sequence of probabilities for a multinomial distribution.

Returns **D** – same shape as p

Return type ndarray

`sagenet.utils.`**`save_adata`**(*adata*, *attr*, *key*, *data*)

updates an attribute of an *AnnData* object

Parameters

- **adata** (*AnnData*) – The annotated data matrix of shape *n_obs* × *n_vars*. Rows correspond to cells and columns to genes.
- **attr** (*str*) – must be an attribute of *adata*, e.g., *obs*, *var*, etc.
- **key** (*str*) – must be a key in the attr
- **data** (*non-specific*) – the data to be updated/placed

5.9 Multiple references

In this notebook we show installation and basic usage of **SageNet**.

[ ]:
```
!pip install -q torch-scatter -f https://data.pyg.org/whl/torch-1.10.0+cu111.html
!pip install -q torch-sparse -f https://data.pyg.org/whl/torch-1.10.0+cu111.html
!pip install -q git+https://github.com/pyg-team/pytorch_geometric.git
```

[ ]:
```
!pwd
```

[ ]:
```
!git clone https://github.com/MarioniLab/sagenet
%cd sagenet
!pip install .
```

```
fatal: destination path 'sagenet' already exists and is not an empty directory.
/content/sagenet/sagenet
ERROR: Directory '.' is not installable. Neither 'setup.py' nor 'pyproject.toml' found.
```

[ ]:
```
import sagenet as sg
import scanpy as sc
import squidpy as sq
import anndata as ad
import random
random.seed(10)
```

[ ]:
```
celltype_colours = {
  "Epiblast" : "#635547",
  "Primitive Streak" : "#DABE99",
  "Caudal epiblast" : "#9e6762",
  "PGC" : "#FACB12",
  "Anterior Primitive Streak" : "#c19f70",
  "Notochord" : "#0F4A9C",
  "Def. endoderm" : "#F397C0",
  "Definitive endoderm" : "#F397C0",
  "Gut" : "#EF5A9D",
  "Gut tube" : "#EF5A9D",
  "Nascent mesoderm" : "#C594BF",
  "Mixed mesoderm" : "#DFCDE4",
  "Intermediate mesoderm" : "#139992",
  "Caudal Mesoderm" : "#3F84AA",
  "Paraxial mesoderm" : "#8DB5CE",
  "Somitic mesoderm" : "#005579",
  "Pharyngeal mesoderm" : "#C9EBFB",
  "Splanchnic mesoderm" : "#C9EBFB",
  "Cardiomyocytes" : "#B51D8D",
  "Allantois" : "#532C8A",
  "ExE mesoderm" : "#8870ad",
  "Lateral plate mesoderm" : "#8870ad",
  "Mesenchyme" : "#cc7818",
  "Mixed mesenchymal mesoderm" : "#cc7818",
  "Haematoendothelial progenitors" : "#FBBE92",
  "Endothelium" : "#ff891c",
  "Blood progenitors 1" : "#f9decf",
  "Blood progenitors 2" : "#c9a997",
  "Erythroid1" : "#C72228",
  "Erythroid2" : "#f79083",
  "Erythroid3" : "#EF4E22",
  "Erythroid" : "#f79083",
  "Blood progenitors" : "#f9decf",
  "NMP" : "#8EC792",
  "Rostral neurectoderm" : "#65A83E",
  "Caudal neurectoderm" : "#354E23",
  "Neural crest" : "#C3C388",
  "Forebrain/Midbrain/Hindbrain" : "#647a4f",
  "Spinal cord" : "#CDE088",
  "Surface ectoderm" : "#f7f79e",
  "Visceral endoderm" : "#F6BFCB",
  "ExE endoderm" : "#7F6874",
  "ExE ectoderm" : "#989898",
  "Parietal endoderm" : "#1A1A1A",
```

(continues on next page)

(continued from previous page)

```
  "Unknown" : "#FFFFFF",
  "Low quality" : "#e6e6e6",
  # somitic and paraxial types
  # colour from T chimera paper Guibentif et al Developmental Cell 2021
  "Cranial mesoderm" : "#77441B",
  "Anterior somitic tissues" : "#F90026",
  "Sclerotome" : "#A10037",
  "Dermomyotome" : "#DA5921",
  "Posterior somitic tissues" : "#E1C239",
  "Presomitic mesoderm" : "#9DD84A"
}
```

[ ]:
```
from copy import copy
adata_r1 = sg.datasets.seqFISH1()
adata_r2 = sg.datasets.seqFISH2()
adata_r3 = sg.datasets.seqFISH3()
adata_q1 = copy(adata_r1)
adata_q2 = copy(adata_r2)
adata_q3 = copy(adata_r3)
adata_q4 = sg.datasets.MGA()
sc.pp.subsample(adata_q1, fraction=0.25)
sc.pp.subsample(adata_q2, fraction=0.25)
sc.pp.subsample(adata_q3, fraction=0.25)
sc.pp.subsample(adata_q4, fraction=0.25)
adata_q = ad.concat([adata_q1, adata_q2, adata_q3, adata_q4], join="inner")
del adata_q1
del adata_q2
del adata_q3
del adata_q4
```

[ ]:
```
from sagenet.utils import glasso
import numpy as np
glasso(adata_r1, [0.5, 0.75, 1])
adata_r1.obsm['spatial'] = np.array(adata_r1.obs[['x','y']])
sq.gr.spatial_neighbors(adata_r1, coord_type="generic")
sc.tl.leiden(adata_r1, resolution=.01, random_state=0, key_added='leiden_0.01',␣
↪adjacency=adata_r1.obsp["spatial_connectivities"])
sc.tl.leiden(adata_r1, resolution=.05, random_state=0, key_added='leiden_0.05',␣
↪adjacency=adata_r1.obsp["spatial_connectivities"])
sc.tl.leiden(adata_r1, resolution=.1, random_state=0, key_added='leiden_0.1',␣
↪adjacency=adata_r1.obsp["spatial_connectivities"])
sc.tl.leiden(adata_r1, resolution=.5, random_state=0, key_added='leiden_0.5',␣
↪adjacency=adata_r1.obsp["spatial_connectivities"])
sc.tl.leiden(adata_r1, resolution=1, random_state=0, key_added='leiden_1',␣
↪adjacency=adata_r1.obsp["spatial_connectivities"])
glasso(adata_r2, [0.5, 0.75, 1])
adata_r2.obsm['spatial'] = np.array(adata_r2.obs[['x','y']])
sq.gr.spatial_neighbors(adata_r2, coord_type="generic")
sc.tl.leiden(adata_r2, resolution=.01, random_state=0, key_added='leiden_0.01',␣
↪adjacency=adata_r2.obsp["spatial_connectivities"])
sc.tl.leiden(adata_r2, resolution=.05, random_state=0, key_added='leiden_0.05',␣
↪adjacency=adata_r2.obsp["spatial_connectivities"])
```

(continues on next page)

(continued from previous page)

```
sc.tl.leiden(adata_r2, resolution=.1, random_state=0, key_added='leiden_0.1',␣
↪adjacency=adata_r2.obsp["spatial_connectivities"])
sc.tl.leiden(adata_r2, resolution=.5, random_state=0, key_added='leiden_0.5',␣
↪adjacency=adata_r2.obsp["spatial_connectivities"])
sc.tl.leiden(adata_r2, resolution=1, random_state=0, key_added='leiden_1',␣
↪adjacency=adata_r2.obsp["spatial_connectivities"])
glasso(adata_r3, [0.5, 0.75, 1])
adata_r3.obsm['spatial'] = np.array(adata_r3.obs[['x','y']])
sq.gr.spatial_neighbors(adata_r3, coord_type="generic")
sc.tl.leiden(adata_r3, resolution=.01, random_state=0, key_added='leiden_0.01',␣
↪adjacency=adata_r3.obsp["spatial_connectivities"])
sc.tl.leiden(adata_r3, resolution=.05, random_state=0, key_added='leiden_0.05',␣
↪adjacency=adata_r3.obsp["spatial_connectivities"])
sc.tl.leiden(adata_r3, resolution=.1, random_state=0, key_added='leiden_0.1',␣
↪adjacency=adata_r3.obsp["spatial_connectivities"])
sc.tl.leiden(adata_r3, resolution=.5, random_state=0, key_added='leiden_0.5',␣
↪adjacency=adata_r3.obsp["spatial_connectivities"])
sc.tl.leiden(adata_r3, resolution=1, random_state=0, key_added='leiden_1',␣
↪adjacency=adata_r3.obsp["spatial_connectivities"])
```

```
[ ]: import torch
     if torch.cuda.is_available():
       dev = "cuda:0"
     else:
       dev = "cpu"
     device = torch.device(dev)
     print(device)
```

```
cpu
```

```
[ ]: sg_obj = sg.sage.sage(device=device)
```

```
[ ]: sg_obj.add_ref(adata_r1, comm_columns=['leiden_0.01', 'leiden_0.05', 'leiden_0.1',
     ↪'leiden_0.5', 'leiden_1'], tag='seqFISH_ref1', epochs=15, verbose = False)
```

```
/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:13: UserWarning:
↪'data.DataLoader' is deprecated, use 'loader.DataLoader' instead
  warnings.warn(out)
/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:13: UserWarning:
↪'data.DataLoader' is deprecated, use 'loader.DataLoader' instead
  warnings.warn(out)
/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:13: UserWarning:
↪'data.DataLoader' is deprecated, use 'loader.DataLoader' instead
  warnings.warn(out)
/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:13: UserWarning:
↪'data.DataLoader' is deprecated, use 'loader.DataLoader' instead
  warnings.warn(out)
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:␣
↪UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with␣
↪no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:␣
↪UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with␣
↪no predicted samples. Use `zero_division` parameter to control this behavior.
```

(continues on next page)

(continued from previous page)

```
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:13: UserWarning:
↪'data.DataLoader' is deprecated, use 'loader.DataLoader' instead
  warnings.warn(out)
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:␣
↪UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with␣
↪no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:␣
↪UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with␣
↪no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:␣
↪UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with␣
↪no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

```
[ ]: sg_obj.add_ref(adata_r2, comm_columns=['leiden_0.01', 'leiden_0.05', 'leiden_0.1',
     ↪'leiden_0.5', 'leiden_1'], tag='seqFISH_ref2', epochs=15, verbose = False)
```

```
/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:13: UserWarning:
↪'data.DataLoader' is deprecated, use 'loader.DataLoader' instead
  warnings.warn(out)
/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:13: UserWarning:
↪'data.DataLoader' is deprecated, use 'loader.DataLoader' instead
  warnings.warn(out)
/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:13: UserWarning:
↪'data.DataLoader' is deprecated, use 'loader.DataLoader' instead
  warnings.warn(out)
/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:13: UserWarning:
↪'data.DataLoader' is deprecated, use 'loader.DataLoader' instead
  warnings.warn(out)
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:␣
↪UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with␣
↪no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:13: UserWarning:
↪'data.DataLoader' is deprecated, use 'loader.DataLoader' instead
  warnings.warn(out)
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:␣
↪UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with␣
↪no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:␣
↪UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with␣
↪no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1308:␣
↪UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with␣
↪no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[ ]:
```
sg_obj.add_ref(adata_r3, comm_columns=['leiden_0.01', 'leiden_0.05', 'leiden_0.1',
↪'leiden_0.5', 'leiden_1'], tag='seqFISH_ref3', epochs=15, verbose = False)
```

[ ]:
```
ind = np.argsort(-(adata_r.var['seqFISH_ref_entropy']+ adata_r.var['seqFISH_ref2_entropy
↪'] + adata_r.var['seqFISH_ref3_entropy']))[0:12]
with rc_context({'figure.figsize': (4, 4)}):
  sc.pl.spatial(adata_r, color=list(adata_r.var_names[ind]), ncols=4, spot_size=0.03,␣
↪legend_loc=None)
```

[ ]:
```
!mkdir models
!mkdir models/seqFISH_ref
sg_obj.save_model_as_folder('models/seqFISH_multiple_ref')
```

[ ]:
```
sg_obj.map_query(adata_q)
```

[ ]:
```
import anndata
dist_adata = anndata.AnnData(adata_q.obsm['dist_map'], obs = adata_q.obs)
knn_indices, knn_dists, forest = sc.neighbors.compute_neighbors_umap(dist_adata.X, n_
↪neighbors=50, metric='precomputed')
dist_adata.obsp['distances'], dist_adata.obsp['connectivities'] = sc.neighbors._compute_
↪connectivities_umap(
    knn_indices,
    knn_dists,
    dist_adata.shape[0],
    50, # change to neighbors you plan to use
)
sc.pp.neighbors(dist_adata, metric='precomputed', use_rep='X')
sc.tl.umap(dist_adata)
sc.pl.umap(dist_adata, color='cell_type', palette=celltype_colours, save='eli.pdf')
```

5.10 Hello World!

In this notebook we show installation and basic usage of **SageNet**.

5.10.1 pyG dependencies

pytorch geometric has specefic dependencies which we highly recommend the user to install them following this.

[ ]:
```
!pip install -q torch-scatter -f https://data.pyg.org/whl/torch-1.10.0+cu111.html
!pip install -q torch-sparse -f https://data.pyg.org/whl/torch-1.10.0+cu111.html
!pip install -q git+https://github.com/pyg-team/pytorch_geometric.git
```

5.10.2 Install SageNet

The use can clone SageNet from GitHub and install it as follows.

[ ]:
```
!git clone https://github.com/MarioniLab/sagenet
%cd sagenet
!pip install .
```

5.10.3 Import packages

We use `anndata <https://anndata.readthedocs.io/en/latest/>`__ to be compatible with `scanpy <https://scanpy.readthedocs.io>`__ and `squidpy <https://squidpy.readthedocs.io/en/stable/>`__ for pre- and post-processing steps.

[ ]:
```
import sagenet as sg
import scanpy as sc
import squidpy as sq
import anndata as ad
import random
random.seed(10)
```

5.10.4 Load datasets

[ ]:
```
from copy import copy
adata_r = sg.datasets.seqFISH()
adata_q1 = copy(adata_r)
adata_q2 = sg.datasets.MGA()
sc.pp.subsample(adata_q1, fraction=0.25)
sc.pp.subsample(adata_q2, fraction=0.25)
adata_q = ad.concat([adata_q1, adata_q2], join="inner")
del adata_q1
del adata_q2
```

[ ]:
```
celltype_colours = {
  "Epiblast" : "#635547",
  "Primitive Streak" : "#DABE99",
  "Caudal epiblast" : "#9e6762",
  "PGC" : "#FACB12",
  "Anterior Primitive Streak" : "#c19f70",
  "Notochord" : "#0F4A9C",
  "Def. endoderm" : "#F397C0",
  "Definitive endoderm" : "#F397C0",
  "Gut" : "#EF5A9D",
  "Gut tube" : "#EF5A9D",
  "Nascent mesoderm" : "#C594BF",
  "Mixed mesoderm" : "#DFCDE4",
  "Intermediate mesoderm" : "#139992",
  "Caudal Mesoderm" : "#3F84AA",
  "Paraxial mesoderm" : "#8DB5CE",
  "Somitic mesoderm" : "#005579",
  "Pharyngeal mesoderm" : "#C9EBFB",
```

(continues on next page)

(continued from previous page)

```
  "Splanchnic mesoderm" : "#C9EBFB",
  "Cardiomyocytes" : "#B51D8D",
  "Allantois" : "#532C8A",
  "ExE mesoderm" : "#8870ad",
  "Lateral plate mesoderm" : "#8870ad",
  "Mesenchyme" : "#cc7818",
  "Mixed mesenchymal mesoderm" : "#cc7818",
  "Haematoendothelial progenitors" : "#FBBE92",
  "Endothelium" : "#ff891c",
  "Blood progenitors 1" : "#f9decf",
  "Blood progenitors 2" : "#c9a997",
  "Erythroid1" : "#C72228",
  "Erythroid2" : "#f79083",
  "Erythroid3" : "#EF4E22",
  "Erythroid" : "#f79083",
  "Blood progenitors" : "#f9decf",
  "NMP" : "#8EC792",
  "Rostral neurectoderm" : "#65A83E",
  "Caudal neurectoderm" : "#354E23",
  "Neural crest" : "#C3C388",
  "Forebrain/Midbrain/Hindbrain" : "#647a4f",
  "Spinal cord" : "#CDE088",
  "Surface ectoderm" : "#f7f79e",
  "Visceral endoderm" : "#F6BFCB",
  "ExE endoderm" : "#7F6874",
  "ExE ectoderm" : "#989898",
  "Parietal endoderm" : "#1A1A1A",
  "Unknown" : "#FFFFFF",
  "Low quality" : "#e6e6e6",
  # somitic and paraxial types
  # colour from T chimera paper Guibentif et al Developmental Cell 2021
  "Cranial mesoderm" : "#77441B",
  "Anterior somitic tissues" : "#F90026",
  "Sclerotome" : "#A10037",
  "Dermomyotome" : "#DA5921",
  "Posterior somitic tissues" : "#E1C239",
  "Presomitic mesoderm" : "#9DD84A"
}
```

5.10.5 Preprocess the reference dataset

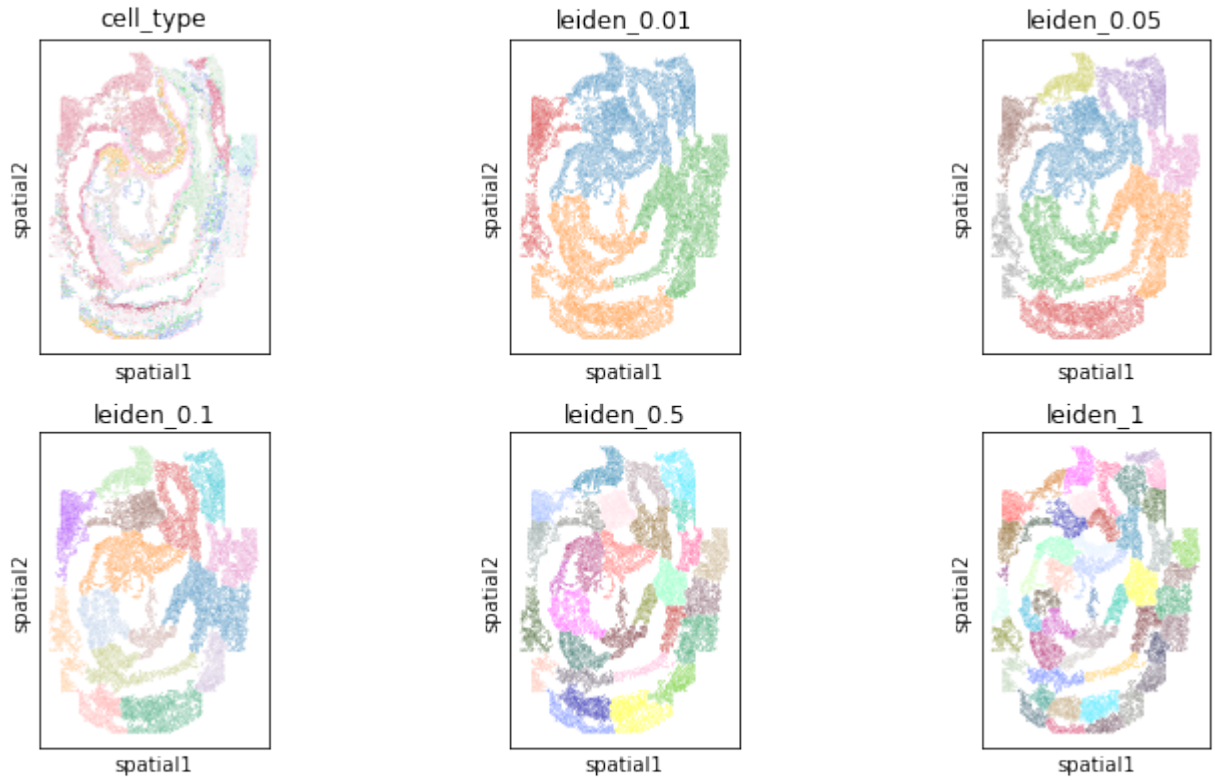
[ ]:
```
from sagenet.utils import glasso
import numpy as np
glasso(adata_r, [0.5, 0.75, 1])
adata_r.obsm['spatial'] = np.array(adata_r.obs[['x','y']])
sq.gr.spatial_neighbors(adata_r, coord_type="generic")
sc.tl.leiden(adata_r, resolution=.01, random_state=0, key_added='leiden_0.01',␣
↪adjacency=adata_r.obsp["spatial_connectivities"])
sc.tl.leiden(adata_r, resolution=.05, random_state=0, key_added='leiden_0.05',␣
↪adjacency=adata_r.obsp["spatial_connectivities"])
```

(continues on next page)

(continued from previous page)

```
sc.tl.leiden(adata_r, resolution=.1, random_state=0, key_added='leiden_0.1',␣
↪adjacency=adata_r.obsp["spatial_connectivities"])
sc.tl.leiden(adata_r, resolution=.5, random_state=0, key_added='leiden_0.5',␣
↪adjacency=adata_r.obsp["spatial_connectivities"])
sc.tl.leiden(adata_r, resolution=1, random_state=0, key_added='leiden_1',␣
↪adjacency=adata_r.obsp["spatial_connectivities"])
```

[ ]:
```
from matplotlib.pyplot import rc_context
with rc_context({'figure.figsize': (3, 3)}):
  sc.pl.spatial(adata_r, color=['cell_type', 'leiden_0.01', 'leiden_0.05', 'leiden_0.1',
↪'leiden_0.5', 'leiden_1'], ncols=3, spot_size=0.03, legend_loc=None)
```



5.10.6 Check for GPU availibility

[ ]:
```
import torch
if torch.cuda.is_available():
  dev = "cuda:0"
else:
  dev = "cpu"
device = torch.device(dev)
print(device)
```

```
cuda:0
```

5.10.7 Define the sagenet object

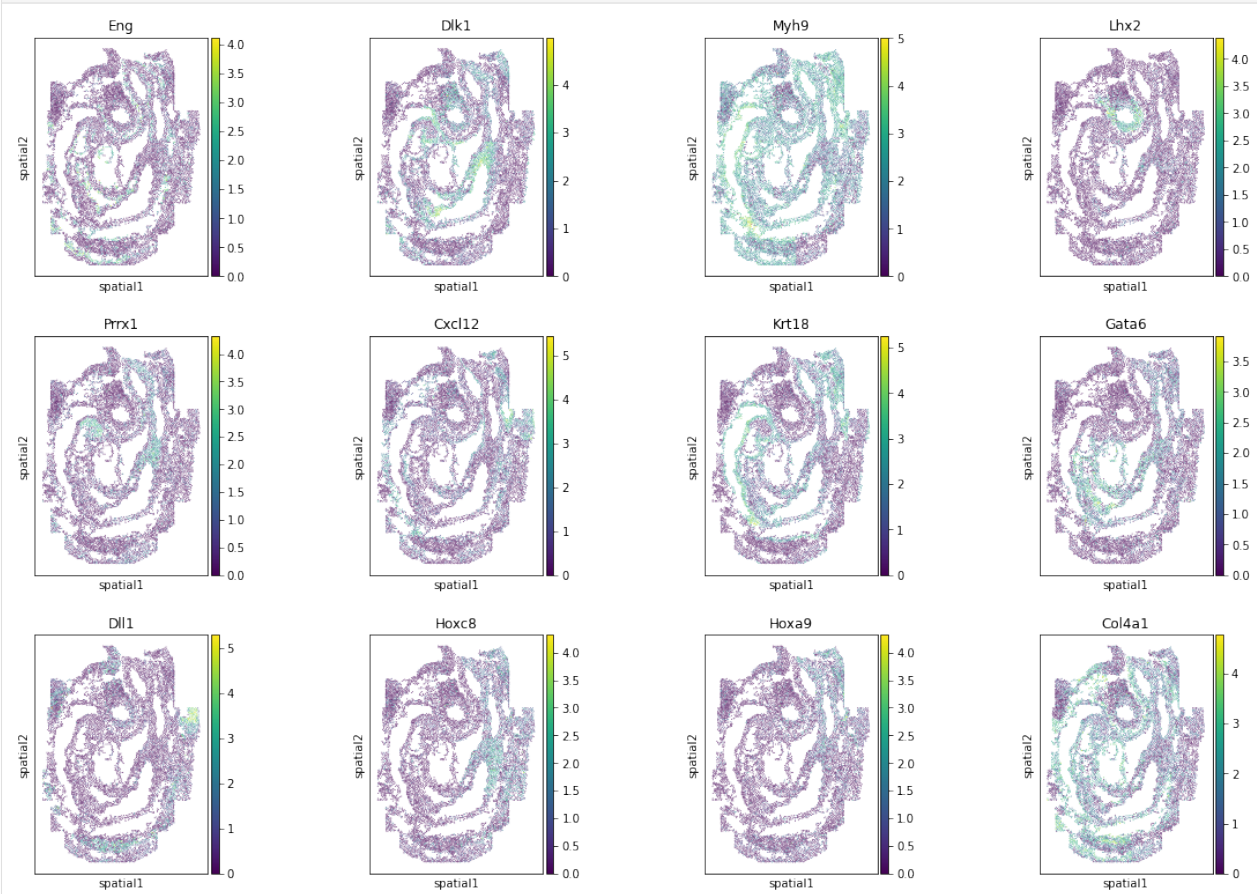
```
[ ]: sg_obj = sg.sage.sage(device=device)
```

5.10.8 Train on the reference dataset

```
[ ]: sg_obj.add_ref(adata_r, comm_columns=['leiden_0.01', 'leiden_0.05', 'leiden_0.1',
     ↪'leiden_0.5', 'leiden_1'], tag='seqFISH_ref', epochs=20, verbose = False)
```

5.10.9 Spatially informative genes

```
[ ]: ind = np.argsort(-adata_r.var['seqFISH_ref_entropy'])[0:12]
     with rc_context({'figure.figsize': (4, 4)}):
       sc.pl.spatial(adata_r, color=list(adata_r.var_names[ind]), ncols=4, spot_size=0.03,␣
     ↪legend_loc=None)
```



5.10.10 Save the trained model

[ ]:
```
!mkdir models
!mkdir models/seqFISH_ref
sg_obj.save_model_as_folder('models/seqFISH_ref')
```

```
mkdir: cannot create directory ‘models’: File exists
mkdir: cannot create directory ‘models/seqFISH_ref’: File exists
```

5.10.11 Map the query dataset to space

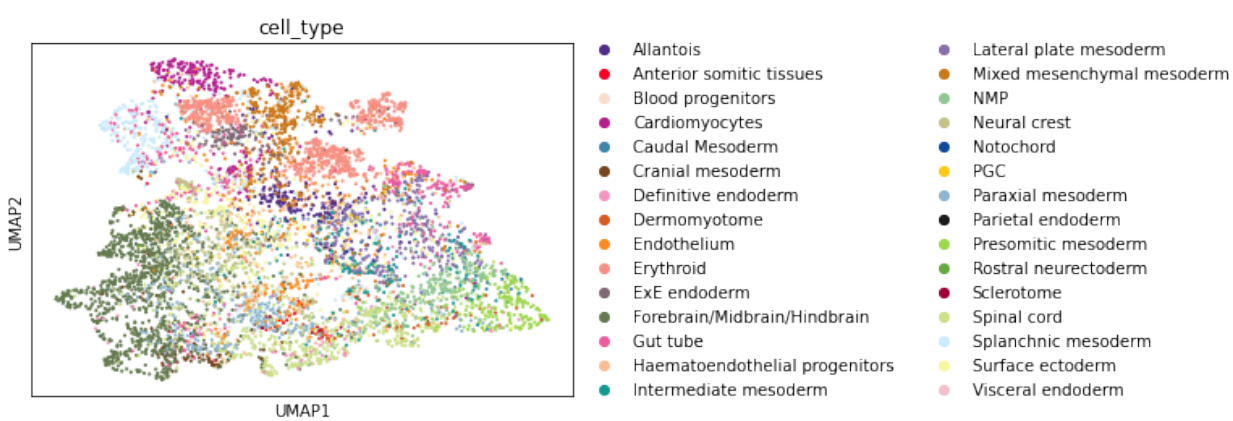
[ ]:
```
sg_obj.map_query(adata_q)
```

```
/usr/local/lib/python3.7/dist-packages/torch_geometric/deprecation.py:13: UserWarning:
↪'data.DataLoader' is deprecated, use 'loader.DataLoader' instead
  warnings.warn(out)
```

5.10.12 Compute UMAP of the embedded cells

[ ]:
```
import anndata
dist_adata = anndata.AnnData(adata_q.obsm['dist_map'], obs = adata_q.obs)
knn_indices, knn_dists, forest = sc.neighbors.compute_neighbors_umap(dist_adata.X, n_
↪neighbors=50, metric='precomputed')
dist_adata.obsp['distances'], dist_adata.obsp['connectivities'] = sc.neighbors._compute_
↪connectivities_umap(
    knn_indices,
    knn_dists,
    dist_adata.shape[0],
    50, # change to neighbors you plan to use
)
sc.pp.neighbors(dist_adata, metric='precomputed', use_rep='X')
sc.tl.umap(dist_adata)
sc.pl.umap(dist_adata, color='cell_type', palette=celltype_colours, save='eli.pdf')
```

```
WARNING: saving figure to file figures/umapeli.pdf
```



PYTHON MODULE INDEX

S

INDEX