CodingLab10

July 24, 2022

Neural Data Science

Lecturer: Prof. Dr. Philipp Berens

Tutors: Jonas Beck, Ziwei Huang, Rita González Márquez

Summer term 2022

Names: FILL IN YOUR NAMES HERE

1 Coding Lab 10

Introduction into the analysis of neural morphologies (10 pts + bonus 3 pts)

1.1 Introduction

The anatomical shape of a neuron — its morphology — has fascinated scientists ever since the pioneering work of Cajal (Ramon y Cajal, 1911). A neuron's dendritic and axonal processes naturally decide what other neurons it can connect to, hence, its shape plays an important role for its function in the circuit. In particular, different functional types of neurons have fundamentally different morphologies.

This notebook will introduce you to the analysis of neural morphologies using the dendrites of over 500 retinal ganglion cells. The aim is to teach you two different ways of representing morphologies and give you an impression of their repsective strengths and weaknesses.

1.1.1 1. Data

The data set contains morphological reconstructions of 599 retinal ganglion cell dendrites with cell type label and projection target to either the parabigeminal (Pbg) or the pulvinar nucleus (LP)(Reinhard et al. (2019)). Here we only keep cells that map to clusters with more than six cells per cluster which leads to 550 remaining reconstructions.

Download the data file `nda_ex_10_data.zip` from ILIAS and unzip it in a subfolder `../data/`

1.1.2 Software

We will use MorphoPy (Laturnus, et al., 2020; https://github.com/berenslab/MorphoPy) for this exercise. We recommend to use the Github version, as it is more up-to-date:

```
git clone https://github.com/berenslab/MorphoPy
pip install -e MorphoPy
```

Most of the computations and even some plottings will be handled by MorphoPy. You can learn more about MorphoPy's APIs in this tutorial.

[1]:
```
import pandas as pd
import numpy as np
import os
import sys
sys.path.append("../")

from morphopy.computation import file_manager
from morphopy.neurontree.plotting import show_threeview
from morphopy.neurontree import NeuronTree as nt

import warnings
warnings.filterwarnings("ignore")

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

sns.set_context('notebook')
```

2 Inspect the raw data

File format Morphological reconstructions are typically stored in the SWC file format, a simple text file that holds node information in each row and connects nodes through the `parent` node id. A parent id of -1 indicates no parent, so the starting point of the tree graph, also called the root. The `type` label indicates the node type (1: somatic , 2: axonal, 3: dendritic (basal), 4: dendritic (apical), 5+: custom). The code snippet below loads in one swc file and prints its head.

You can find a more detailed specification of SWC and SWC+ here and here.

[2]:
```
def load_swc(filepath):
    """
    Loads in the swc located at filepath as a pandas dataframe.
    """
    swc = pd.read_csv(filepath, delim_whitespace=True, comment='#',
                      names=['n', 'type', 'x', 'y', 'z', 'radius', 'parent'],␣
 ↪index_col=False)
    return swc

colors = sns.color_palette('rainbow_r', n_colors=14)
```

[3]:
```
PATH = 'data/nda_ex_10_data/'
data_path = PATH + 'neurons/soma-centered/'
filename = '0006_00535_4L_C02_01.swc'
filepath = data_path + filename
```

```
swc = load_swc(filepath)
swc.head()
```

[3]:

	n	type	x	y	z	radius	parent
0	1	1	0.0	0.0	47.0	1.0	-1
1	2	3	-3.0	0.0	47.0	1.0	1
2	3	3	17.0	-8.0	51.0	1.0	1
3	4	3	24.0	-31.0	38.0	1.0	3
4	5	3	2.0	14.0	42.0	1.0	1

The labels `x`, `y`, and `z` hold a node's 3D coordinate in tracing space (here in microns). For reasons of simplicity we will work with reconstructions that are soma centered in XY.

The assigned cell type labels are stored in the file `rgc_labels.csv` and indexed by their `Cell_nr`. In this file you find three different cluster assignments: `clusterA` is the assignment of the authors (clus1 – clus14), `clusterB` is the respective cluster identifier of the Eyewire museum (also see Bae et al. 2018), and `clusterC` are molecular or functional label names when available. We have formatted the cluster assignments of the authors (`clusterA`) into integer values and stored them in the column `cluster`, which we will use in the following.

[4]:
```
labels = pd.read_csv(PATH + "rgc_labels.csv", index_col=0)

cluster_label, cluster_counts = np.unique(labels['cluster'], return_counts=True)
labels.head()
```

[4]:

	Cell_nr	projection_site	clusterA	clusterB	clusterC	cluster
1	2	LP	clus6	4ow	tOFF	6
2	3	LP	clus2	2an	F-mini-OFF	2
3	4	LP	clus1	1wt	sOFF	1
4	6	LP	clus7	5to	NaN	7
5	7	LP	clus10	6sn	NaN	10

2.1 Task 1: Plotting individual morphologies

Load data using `file_manager` and plot individual morphologie using `show_threeview` of from `MorphoPy`. It plots all three planar views on the reconstruction.

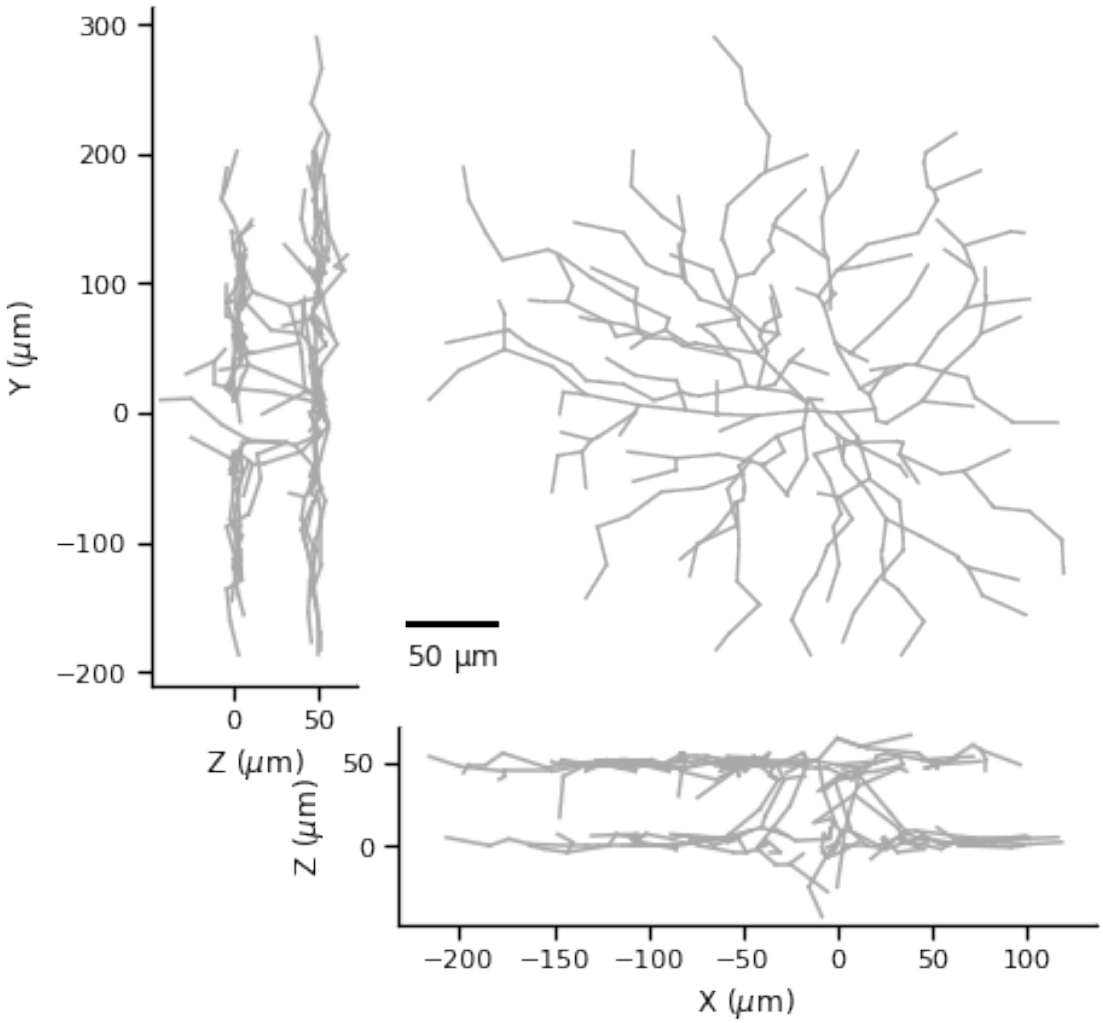
Here, XY shows the planar view on top of the retina, and Z denotes the location within the inner plexiform layer (IPL).

Noted, by default, the `file_manager` loads data with `pca_rot=True` and `soma_center=True`. For the all the exercise in this Coding Lab, it's better to set both of them as `False`.

Grading: 2pts

[5]:
```
from morphopy.computation.file_manager import load_swc_file
files = list(os.walk(data_path))[0][2]
files.sort()

neurons = []
```

```
for f in files:
#     print(data_path+f)
    swc = load_swc(data_path+f)
    N = nt.NeuronTree(swc=swc)
    neurons.append(N)

ix = 19


fig = plt.figure(figsize=(7,7))

# insert your code here

# ------------------------------------------
# load example data (ix=19) with `file_manager`
# from morphology (1 pt)
# ------------------------------------------
N = neurons[ix]

# ----------------------------------
# plot all three planar views (0.5pts)
# ----------------------------------
show_threeview(N, fig=fig)
```



SWC files are a compact way for storing neural morphologies but their graph structure makes them difficult to handle for current machine learning methods. We, therefore, need to convert our reconstructions into a reasonable vector-like representations.

Here we will present two commonly chosen representations: Morphometric statistics and density maps

[6]:
```
# load all reconstructions. Note: files are sorted by cell number
def load_files(path):
    """ Returns an object array of NeuronTrees containing all reconstructions␣
 ↪at `path`.
    The reconstructions are sorted ascendingly by their filename."""
    neurons = []
    path, _, files = list(os.walk(path))[0]
    files.sort()
```

```
    for f in files:

        # insert your code here

        # ----------------------------------------
        # load data with `file_manager` from morphology
        # for all data (0.5 pts)
        # ----------------------------------------
        swc = load_swc(data_path+f)
        N = nt.NeuronTree(swc=swc)

        neurons.append(N)
    return neurons

neurons = load_files(data_path)
print("Number of reconstructions: ", len(neurons))
```

```
Number of reconstructions:  550
```

2.2 Task 2: Morphometric statistics

Morphometric statistics denote a set of hand-crafted single valued features such as `soma radius`, `number of tips` or `average branch angle`. For a more detailed explanation of morphometrics please refer to the MorphoPy documentation.

Grading: 3pts

First, you should compute this representation for each cell using the function `compute_morphometric_statistics` of the MorphoPy package which computes a predefined set of 28 statistics. (1pt)

[7]:
```
from morphopy.computation.feature_presentation import␣
 ↪compute_morphometric_statistics

# insert your code here

# ----------------------------------------------------------------------
# 1. extraction of morphometric statistics on entire data set (0.5 pts)
# ----------------------------------------------------------------------

ms_list = []
for n in neurons:
    cms= compute_morphometric_statistics(n)
    ms_list.append( cms)

#␣
 ↪-------------------------------------------------------------------------------------
```

```
# 2. concatenate data into one pd.DataFrame and set the `Cell_nr`` as index (0.
 ↪5 pts)
#␣
 ↪-----------------------------------------------------------------------------
morphometric_statistics = pd.concat(ms_list).set_index(labels['Cell_nr'])

morphometric_statistics.head()
```

[7]:

Cell_nr	branch_points	width	depth	height	tips	stems	total_length	…
2	66.0	412.0	596.0	54.0	69.0	3.0	6579.358659	…
3	18.0	307.0	266.0	68.0	23.0	1.0	1865.736494	…
4	54.0	505.0	353.0	72.0	61.0	6.0	4134.984734	…
6	36.0	345.0	438.0	49.0	39.0	4.0	3224.958032	…
7	31.0	413.0	386.0	58.0	40.0	7.0	3464.729453	…

Cell_nr	avg_thickness	max_thickness	total_surface	…
2	1.0	1.0	41339.329656	…
3	1.0	1.0	11722.768129	…
4	1.0	1.0	25980.875324	…
6	1.0	1.0	20263.008923	…
7	1.0	1.0	21769.537191	…

Cell_nr	median_intermediate_segment_pl	median_terminal_segment_pl
2	33.479847	45.619642
3	34.501425	34.999147
4	20.211076	28.106939
6	21.708403	41.299541
7	24.839485	53.008633

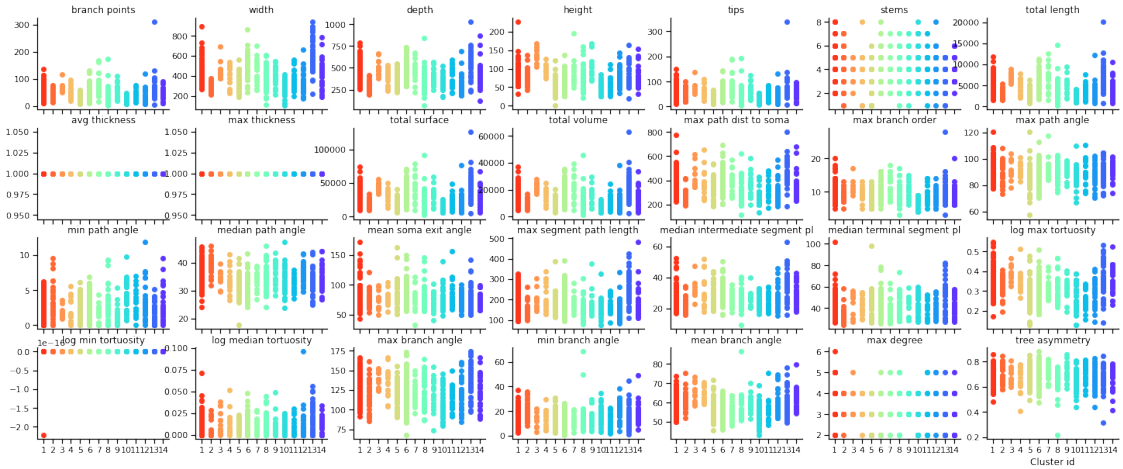
Cell_nr	log_max_tortuosity	log_min_tortuosity	log_median_tortuosity
2	0.253612	0.0	0.005766
3	0.285133	0.0	0.000564
4	0.312270	0.0	0.000000
6	0.251400	0.0	0.017747
7	0.217363	0.0	0.024678

Cell_nr	max_branch_angle	min_branch_angle	mean_branch_angle	max_degree
2	124.974479	7.286952	50.402105	3.0
3	110.367090	18.036960	57.088068	3.0
4	131.508679	25.596621	59.646776	3.0
6	130.034126	16.436535	54.683803	2.0
7	104.036243	2.970445	55.498150	3.0

```
         tree_asymmetry
Cell_nr
2              0.726199
3              0.738502
4              0.706943
6              0.601296
7              0.579942

[5 rows x 28 columns]
```

Now visualize the data: Make a scatter plot for each morphometric statistics for each cluster. (1pt)

[8]:
```
# visualize. Which are the most discriminative ones?

features = morphometric_statistics.columns.values

fig, axes = plt.subplots(4,7, figsize=(25,10), sharex = True)
axes = axes.flatten()

for k, feature in enumerate(features):

    for l in np.unique(labels['cluster']):

        label_index = (labels['cluster'] == l).values

        # insert your code here

        # ----------------------
        # finish the plot (1 pt)
        # ----------------------
        x= [l]*sum(label_index)
        y= morphometric_statistics[label_index][feature]
        axes[k].scatter(x,y, c=[colors[l-1]])

    axes[k].set_title(feature.replace("_", " "))
axes[-1].set_xticks(range(1,15))
axes[-1].set_xlabel('Cluster id')
sns.despine()
```



Q3.a: Which statistics separate clusters well? Which can be removed? (tips: there are 5 uninformative features) (1pt)

Answer: From this perspective there is no feature that directly separates the clusters well, but we can see some diffrences in, branch points,

There are three features that can be directly identidied to be uninformative, as hthey are zero: avg thickness, max thickness and log min tortuosity. Also, total surface and total volume depend directly on total length, which means they carry no further information.

Q3.b: More generally, what do morphometric statistics capture well? What are their advantages, what might be their downsides? Briefly explain. (bonus: 0.5pt)

Answer:

[14]:
```
# For further analysis we will remove uninformative features and z-score along␣
↪each statistic

# ------------------------------------------------------------
# Complete the list with the uninformative features from Q3.a
# ------------------------------------------------------------

features_to_drop = ["avg_thickness", "max_thickness", "total_surface",␣
↪"total_volume", "log_min_tortuosity"]
morphometric_data = morphometric_statistics.drop(features_to_drop, axis=1)

# z-score morphometrics and remove nans and uninformative features
morphometric_data = (morphometric_data - morphometric_data.mean())/
↪morphometric_data.std()
morphometric_data[morphometric_data.isna()] = 0
morphometric_data = morphometric_data.values
```

2.3 Task 3: Density maps

Density maps project a neuron's 3D point cloud (x, y, z) onto a plane or an axis, and bin the projected point cloud into a fixed number of bins. Hereby, the binning controls how much global or local information is kept, which majorly affects the results. While density maps ignore the connectivity between nodes and only assign value to neurite density, this representation can be powerful, when cell type and location in the brain area are highly correlated.

Exercise: Compute the density maps of all neurons onto all cardinal planes and axes using the method `compute_density_maps`. You can manipulate the parameters for the density maps via the dictonary `config`. Make sure that you normalize the density maps globally and bin each direction into 20 bins. You are welcome to explore, how the different projections look like but we will only use the z-projection for further analysis.

Possible parameters to pass are:

- distance: (default=1, in microns) determines the resampling distance.
- bin_size: (default=20, in microns). If set the number of bins will be computed such that one bin spans `bin_size` microns. This is overwritten when `n_bins_x/y/z` is set!
- n_bins_x/y/z: (default=None) specifies the number of bins for each dimension. If set it will overwrite the `bin_size` flag.
- density: (default=True) bool to specify if a density or counts are returned.
- smooth: (default=True) bool to trigger Gaussian smoothing.
- sigma: (default=1) determines std of the Gaussian used for smoothing. The bigger the sigma the more smoothing occurs. If smooth is set to False this parameter is ignored.
- r_min_x/y/z: (in microns) minimum range for binning of x, y, and z. This value will correspond to the minimal histogram edge.
- r_max_x/y/z: (in microns) maximum range for binning for x, y, and z. This value will correspond to the maximal histogram edge.

Grading: 3pts

[15]:
```
# insert your code here

#␣
 ↪-------------------------------------------------------------------------------
# Find the minimal and maximal x, y,z - coordinates in the data set to␣
 ↪normalize
# the density maps globally which corresponds to r_min_x/y/z and r_max_x/y/z (1␣
 ↪pt)
#␣
 ↪-------------------------------------------------------------------------------
# Find the minimal and maximal x, y,z - coordinates in the data set to␣
 ↪normalize the density maps globally



minimal_coordinates = np.min(np.vstack([list(n.get_node_attributes('pos').
 ↪values()) for n in neurons]), axis=0)
```

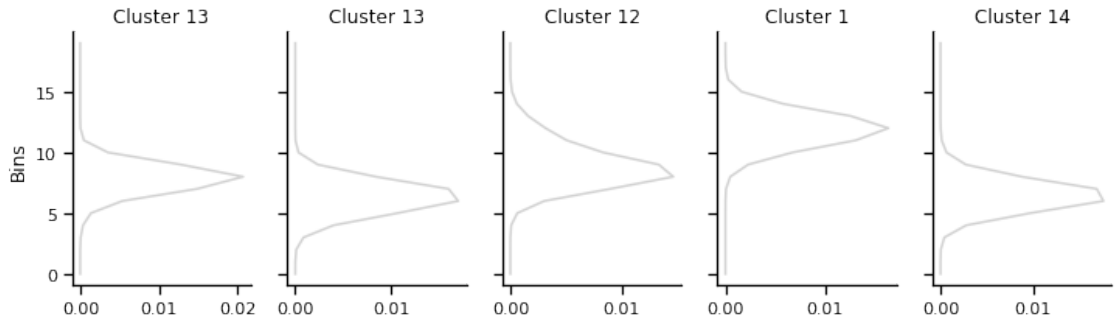
```
maximal_coordinates = np.max(np.vstack([list(n.get_node_attributes('pos').
 ↪values()) for n in neurons]), axis=0)



print('Minimal coordinates:', minimal_coordinates)
print('Maximal coordinates: ', maximal_coordinates)
```

```
Minimal coordinates: [-634. -549. -150.]
Maximal coordinates:  [514. 536. 186.]
```

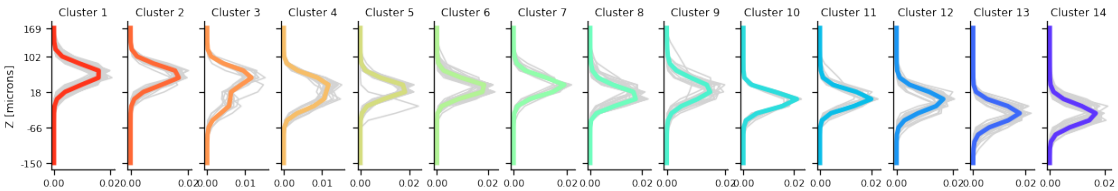
[16]:
```
from morphopy.computation.feature_presentation import compute_density_maps

# ------------------------------
# complete the config dict (1pt)
# ------------------------------

config_global = dict(distance= 1,
                     n_bins_x= 20,
                     n_bins_y= 20,
                     n_bins_z= 20,
                     density=True,
                     smooth=True,
                     sigma=1,
                r_max_x=maximal_coordinates[0], r_max_y=maximal_coordinates[1],␣
 ↪r_max_z=maximal_coordinates[2],
                r_min_x=minimal_coordinates[0], r_min_y=minimal_coordinates[1],␣
 ↪r_min_z=minimal_coordinates[2])

density_maps= [compute_density_maps(n, config_params=config_global) for n in␣
 ↪neurons] # this line might take a while

# extract the z density map
DM_Z = np.array([dm['z_proj']['data'] for dm in density_maps])
```

[17]:
```
# plot z-density maps of 5 randomly chosen individual cells
random_index = np.random.choice(range(len(neurons)), size=5, replace=False)

fig, axes = plt.subplots(1,5,figsize=(12,3), sharey=True)
for k, ix in enumerate(random_index):
    axes[k].plot(DM_Z[ix,:],range(20), c='lightgrey')
    axes[k].set_title('Cluster %i'%labels.iloc[ix]['cluster'])
sns.despine()
axes[0].set_ylabel('Bins')
```

[17]:
```
Text(0, 0.5, 'Bins')
```



[18]:
```
fig, axes = plt.subplots(1, 14, figsize=(21,3), sharey=True)
axes = axes.flatten()

# insert your code here

# ------------------------------------------------------
# plot the Z-density map sorted by class label (1 pt)
# ------------------------------------------------------
for lab in np.unique(labels['cluster']):
        label_index = (labels['cluster'] == lab).values
        y = range(20)
        x_all = DM_Z[label_index].T
        x_mean = np.mean(DM_Z[label_index], axis=0)

        axes[lab-1].plot(x_all, y, c='lightgrey')
        axes[lab-1].plot(x_mean,y, c=colors[lab-1], linewidth=5)
        axes[lab-1].set_title('Cluster %i'%lab)


sns.despine()
axes[0].set_yticks([0,5,10,15,19])
axes[0].set_yticklabels(density_maps[0]['z_proj']['edges'][0][[0,5,10,15,19]].
 ↪astype(int))
_ = axes[0].set_ylabel('Z [microns]')
```



2.4 2D embedding using t-SNE

Embed both data, the morphometric statistics and the density maps, in 2D using t-SNE and color each embedded point by its cluster assignment.

We use `openTSNE` for this part: `pip install opentsne`

Grading: 2pts

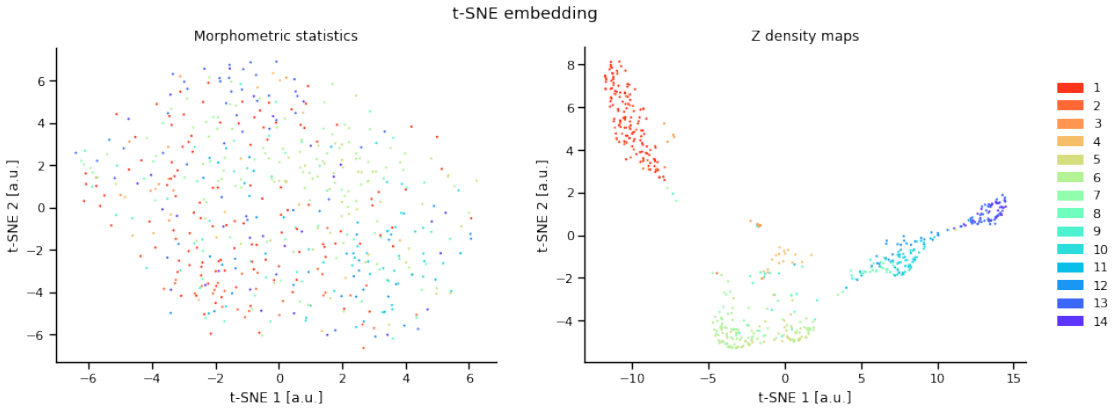
[19]:
```
from openTSNE import TSNE

# insert your code here

# ----------------------------------------------------------------------
# Fit t-SNE with morphometric statistics and density maps (0.5 + 0.5 pt)
# ----------------------------------------------------------------------

tsne = TSNE(
    perplexity=100,
    initialization='pca',
    metric="euclidean",
    n_jobs=20,
    random_state=17,
)

tsne_fit_morph = tsne.fit(morphometric_data)
tsne_fit_dm = tsne.fit(DM_Z)
```

[20]:
```
import matplotlib.patches as mpatches

patch1 = mpatches.Patch(color=colors[0], label='1')
patch2 = mpatches.Patch(color=colors[1], label='2')
patch3 = mpatches.Patch(color=colors[2], label='3')
patch4 = mpatches.Patch(color=colors[3], label='4')
patch5 = mpatches.Patch(color=colors[4], label='5')
patch6 = mpatches.Patch(color=colors[5], label='6')
patch7 = mpatches.Patch(color=colors[6], label='7')
patch8 = mpatches.Patch(color=colors[7], label='8')
patch9 = mpatches.Patch(color=colors[8], label='9')
patch10 = mpatches.Patch(color=colors[9], label='10')
patch11 = mpatches.Patch(color=colors[10], label='11')
patch12 = mpatches.Patch(color=colors[11], label='12')
patch13 = mpatches.Patch(color=colors[12], label='13')
patch14 = mpatches.Patch(color=colors[13], label='14')
```

[21]:
```
fig, axes = plt.subplots(1,2,figsize=(15,5))

# insert your code here
```

```
#␣
 ↪-------------------------------------------------------------------------------
# plot tsne fit of morpometric statistics, color-coded with cluster labels (0.5␣
 ↪pts)
#␣
 ↪-------------------------------------------------------------------------------␣
 ↪
axes[0].scatter(tsne_fit_morph[:,0], tsne_fit_morph[:,1], s=1,c=[colors[i-1]␣
 ↪for i in labels["cluster"]])

sns.despine()

axes[0].set_xlabel('t-SNE 1 [a.u.]')
axes[0].set_ylabel('t-SNE 2 [a.u.]')

axes[0].set_title('Morphometric statistics')

#␣
 ↪-------------------------------------------------------------------------------
# plot tsne fit of z-projected density map , color-coded with cluster labels (0.
 ↪5 pts)
#␣
 ↪-------------------------------------------------------------------------------

sns.despine()
scatter= axes[1].scatter(tsne_fit_dm[:,0], tsne_fit_dm[:,1], s=1,c=[colors[i-1]␣
 ↪for i in labels["cluster"]])
axes[1].set_xlabel('t-SNE 1 [a.u.]')
axes[1].set_ylabel('t-SNE 2 [a.u.]')
plt.
 ↪legend(handles=[patch1,patch2,patch3,patch4,patch5,patch6,patch7,patch8,patch9,patch10,patc
 ↪frameon=False,borderaxespad=0, bbox_to_anchor=(1.2,0.5),loc='right')
axes[1].set_title('Z density maps')

plt.suptitle('t-SNE embedding')
plt.show
```

[21]: <function matplotlib.pyplot.show(close=None, block=None)>



2.5 Bonus: Hierarchical clustering

Typically, we do not have access to ground truth cell type labels but we need to infer the number of classes that are present in the data. In the following we want to explore hierarchical clustering based on morphometric statistics and on density maps. Assume that you know the number of existing cluster ($N_{clus} = 14$). Visualize the clustering dendrogram (use seaborn's `clustermap` method) and then cluster the data (using agglomerative clustering and Ward's method) and assess the cluster label correspondence with ground truth labels using two custer quality metrics of your choice.

Bonus: 2.5 pts

2.5.1 Clustering on morphometric statistics

[ ]:
```
true_labels = labels['cluster'].values
sns.clustermap(morphometric_data, method='ward',
               row_colors=[colors[t-1] for t in true_labels],
               cbar_pos=(1., 0.6, 0.05, 0.18),
               xticklabels = [f.replace("_", " ").capitalize() for f in␣
 ↪features])
```

[ ]:
```
from sklearn.cluster import AgglomerativeClustering
from sklearn.metrics import mutual_info_score, adjusted_rand_score

# insert your code here

# ----------------------------------------------------------------
# Do hierarchical clustering with morphometric statistics (1 pt)
# ----------------------------------------------------------------

agglom_morph =

print('Mutual information score:',  mutual_info_score(labels['cluster'].values,␣
 ↪agglom_morph.labels_))
```

```
print('Adjusted rand index: ', adjusted_rand_score(labels['cluster'].values,␣
 ↪agglom_morph.labels_))
```

2.5.2 Clustering on density maps

[ ]:
```
sns.clustermap(DM_Z, method='ward',
               row_cluster=True,
               row_colors=[colors[t-1] for t in true_labels],
               cbar_pos=(1., 0.6, 0.05, 0.18),
               xticklabels=['Bin %i'%i for i in range(1,21)])
```

[ ]:
```
# insert your code here

# ------------------------------------------------------------
# Do hierarchical clustering with density map (1 pt)
# ------------------------------------------------------------

agglom_dm =

print('Mutual information score:',  mutual_info_score(labels['cluster'].values,␣
 ↪agglom_dm.labels_))
print('Adjusted rand index: ', adjusted_rand_score(labels['cluster'].values,␣
 ↪agglom_dm.labels_))
```

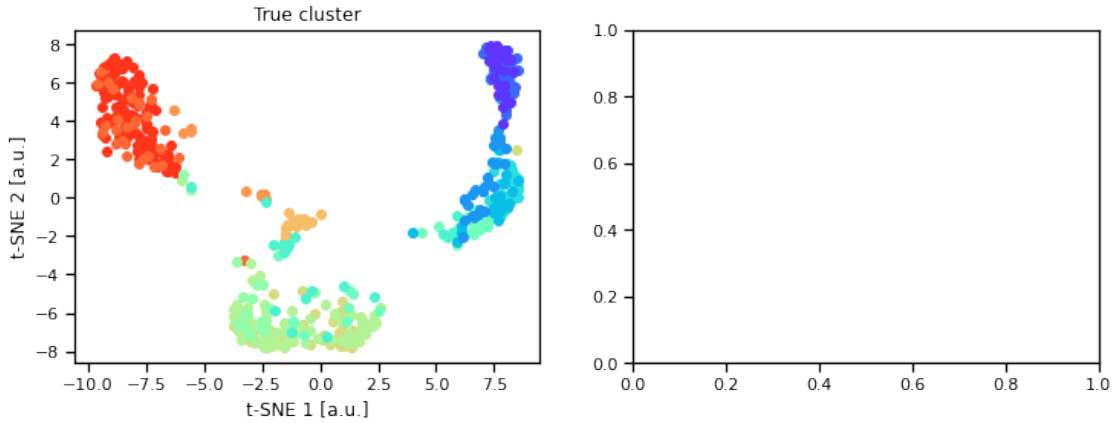
Now plot the t-SNE embeddings from before and label each embedding with its true and, next to it, with its newly inferred cluster labels from the agglomerative clustering.

[133]:
```
# plot t-sne with cluster labels of both representations

new_cluster_colors = sns.color_palette('tab20', n_colors=14)

### density maps ######

fig, axes = plt.subplots(1,2, figsize=(12,4))
axes = axes.flatten()
# plot the true cluster labels
for k, l in enumerate(np.unique(labels['cluster'])):
    label_index = (labels['cluster'] == l).values
    axes[0].scatter(tsne_fit_dm[label_index,0], tsne_fit_dm[label_index,1],␣
 ↪c=[colors[k]], label='Cluster %i'%l)

axes[0].set_xlabel('t-SNE 1 [a.u.]')
axes[0].set_ylabel('t-SNE 2 [a.u.]')
axes[0].set_title('True cluster')

# plot the new clustering
for k,l in enumerate(np.unique(agglom_dm.labels_)):
```

```
    label_index = agglom_dm.labels_ == l
    axes[1].scatter(tsne_fit_dm[label_index,0], tsne_fit_dm[label_index,1],␣
 ↪c=[new_cluster_colors[k]])
axes[1].set_xlabel('t-SNE 1 [a.u.]')
axes[1].set_ylabel('t-SNE 2 [a.u.]')
axes[1].set_title('Inferred cluster')
sns.despine()
plt.suptitle('Z-density map')

### morphometrics ######

fig, axes = plt.subplots(1,2, figsize=(12,4), sharex=True)
axes = axes.flatten()

# plot the true cluster labels
for k, l in enumerate(np.unique(labels['cluster'])):
    label_index = (labels['cluster'] == l).values
    axes[0].scatter(tsne_fit_morph[label_index,0],␣
 ↪tsne_fit_morph[label_index,1], c=[colors[k]], label='Cluster %i'%l)

axes[0].set_xlabel('t-SNE 1 [a.u.]')
axes[0].set_ylabel('t-SNE 2 [a.u.]')
axes[0].set_title('True cluster')

# plot the new clustering
for k,l in enumerate(np.unique(agglom_morph.labels_)):
    label_index = agglom_morph.labels_ == l
    axes[1].scatter(tsne_fit_morph[label_index,0],␣
 ↪tsne_fit_morph[label_index,1], c=[new_cluster_colors[k]])

axes[1].set_xlabel('t-SNE 1 [a.u.]')
axes[1].set_ylabel('t-SNE 2 [a.u.]')
axes[1].set_title('Inferred cluster')
sns.despine()
plt.suptitle('Morphometric statistics')
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-133-d5465c90459c> in <module>
     17
     18 # plot the new clustering
---> 19 for k,l in enumerate(np.unique(agglom_dm.labels_)):
     20     label_index = agglom_dm.labels_ == l
     21     axes[1].scatter(tsne_fit_dm[label_index,0],␣
 ↪tsne_fit_dm[label_index,1], c=[new_cluster_colors[k]])
```

```
NameError: name 'agglom_dm' is not defined
```



Q: Which representation allows for better recovery of the cell type labels? Why is that so? (0.5pt)

Answer:

2.6 Extra: Predicting the projection site

Instead of its cell type, we now want to predict each cell's projection site in the thalamus which is stored in the column `labels['projection_site']`. Fit a logistic regression on both morphological representations and report its average cross validated (cv=5) prediction accuracy for each. Which representation works better to recover the prediction target? Which features are most relevant for that prediction?

You can use `LogisticRegressionCV` of the scikit-learn library directly. Since the classes are imbalanced make sure to report the balanced prediction accuracy. To understand the relevance of individual features plot the fitted linear coefficients.

[ ]:
```
from sklearn.linear_model import LogisticRegressionCV


# Fit logistic regression on density maps
lr_dm = LogisticRegressionCV(scoring='balanced_accuracy')
lr_dm.fit(DM_Z, labels['projection_site'].values)
print('Average projection site prediction accuracy on density maps: %i␣
 ↪percent'%(lr_dm.score(DM_Z, labels['projection_site'].values)*100))


# Fit logistic regression on morphometric data
lr_morph = LogisticRegressionCV(scoring='balanced_accuracy')
lr_morph.fit(morphometric_data, labels['projection_site'].values)
```

```
print('Average projection site prediction accuracy on morphometric data: %i␣
 ↪percent'%(lr_morph.score(morphometric_data, labels['projection_site'].
 ↪values)*100))
```

While Z density maps allow for better recovery of cell type labels, they are worse than morphometric statistics on predicting the projection target.

[ ]:
```
lr = lr_morph

plt.plot(lr.coef_.T)
plt.gca().set_xticks(range(28))
_ = plt.gca().set_xticklabels([f.replace("_", " ").capitalize() for f in␣
 ↪features], rotation=90)
plt.plot(range(lr.n_features_in_), [0]*lr.n_features_in_, c='grey', ls='--')
sns.despine()
plt.ylabel('Coefficient value')
plt.title('Linear weights vector')
```

Q: Which morphometrics are informative on the projection site?

Answer: All that are related to size. The cells projecting to Pbg tend to be bigger in dendritic diameter ($width \times depth$) and have wider branch angles. This is also in line with what Reinhard et al. find.

2.7 Further references

Other ways to represent and compare morphologies are * Persistence: Description and application on somatosensory pyramidal cell dendrites by Kanari et al. 2018

- Tree edit distance: Heumann et al. 2009
- Sequential encoding inspired by BLAST: Encoding and similarity analysis on cortical dendrites by Gilette et al. 2015
- Vector point clouds: BlastNeuron: Wan et al. 2015

[ ]:
```

```