White matter recipe generation - documentation

Michael W. Reimann

February 14, 2019

1 About

This repository is one of two repositories that deal with generating a *whole-neocortex, neuron-to-neuron* connectome in a morphologically detailed model of mouse neocortex. The connectome generation process has two steps: The first step solves the *scientific* problem of extracting the various constraints on long-range neocortical connectivity, based on openly available data. The second step solves the *engineering* problem of using these constraints in a stochastic algorithm that places the long-range synapses into a model of mouse neocortex. This repository deals with the first step. Its purpose is to generate a *white-matter projection recipe* that parameterizes and encodes the identified constraints in the YAML format and is meant to be used by the second step.

Additionally, this repository provides tools to validate that the output of the second step (i.e. a connectome instance) is compliant with what is prescribed in the recipe. This part is currently only compatible with neocortex and connectome models in data formats used in Blue Brain.

1.1 Quick start

To generate the default recipe simply install the python package, for example with pip:

```
cd white_matter
pip install .
```

This installs an executable script that will write the white matter projection recipe to the current directory:

```
write_wm_recipe.py /path/to/configuration.json
```

The only input required is a configuration file in .json format (see below). A configuration with the parameters used for the official version of the recipe is bundled with this repository under configurations/default_template.json.

1.2 Dependencies

A large part of the constraints in the recipe will be parameterized by the meso-scale connectome of the Allen Institute for Brain Science. As such, it requires the python packages *allensdk* and *mouse-connectivity-models* that can be acquired from their github at https://github.com/AllenInstitute.

It further requires the packages h5py, simplejson, numpy, progressbar, PyYAML and scipy that can be acquired from the python package index (pypi.org).

2 Custom configuration

As mentioned above, a default configuration is bundled with this code repository. Alternatively, a user can specify or modify their own configuration file as argument:

```
write_wm_recipe.py /home/username/white_matter/configurations/my_config.json
```

This configuration is a json file with five sections, corresponding to five types of constraints that will be put into the recipe file:

1. **BrainParcellation** specifies the brain parcellation scheme used.
2. **ProjectionStrength** parameterizes how the biological data is used to constrain the strengths of projections on the macroscopic level.
3. **LayerProfiles** parameterizes how the biological data is used to determine the layer profiles of projections, i.e. it constrains on the mesoscopic level.
4. **ProjectionMapping** parameterizes how the topographical mapping between brain regions is determined
5. **PTypes** parameterizes how the single cell logic of projections is constrained
6. **SynapseTypes** parameterizes the physiology of placed synapses

The specific parameters for each section and their meaning are listed in the appendix. A configuration file with the default parameters can be found in the repository under */configurations/default_template.json*. If you want to modify the configuration I recommend starting from that file.

2.1 Regenerating data caches

For three of the five sections, data is cached in hdf5 files. Therefore, if you change their configuration, you first have to delete and then re-generate the corresponding cache files. The paths to the cache files are specified in the configuration file as well (at "cache name" in the table below). Executable scripts to re-generate them are installed together

with the repository. Each script takes as only argument the path to the configuration file.

Section	Cache name	Generator script
PTypes	PTypes/23/json_cache PTypes/23/h5_cache PTypes/4/json_cache PTypes/4/h5_cache PTypes/5pt/json_cache PTypes/5pt/h5_cache PTypes/5it/json_cache PTypes/5it/h5_cache PTypes/6/json_cache PTypes/6/h5_cache	write_ptype_tree_model_cache.py
ProjectionMapping	ProjectionMapping/h5_fn	write_projection_mapping_cache.py
ProjectionStrength	ProjectionStrength/h5_cache	write_projection_strength_cache.py

If a specified path to a cache file in the configuration is not an absolute path it will be interpreted relative to the location of the configuration file itself. For example, consider a configuration file with the following contents at
/home/username/projections/configuration.json :

```
{
  "ProjectionMapping" :
    {
    "h5_fn": "mapping/mapping_cache.h5"
    ...
    }
  ...
}
```

Running the following command:

```
write_projection_mapping_cache.py /home/username/projections/configuration.json
```

will generate the mapping cache under
/home/username/projections/mapping/mapping_cache.h5

Important:
Cache files consistent with the default configuration are bundled together with the repository and used when executing

```
write_wm_recipe.py configuration/default_template.json
```

Consequently, if you do not plan to change the configuration from the default values, you will never have to worry about the cache generator scripts.

3 The white_matter python package

- **white_matter.utils** contains various utilities that are used in the rest of the package or might be useful in the future.
- **white_matter.wm_recipe** the contents of this package largely mirrors the five sections of the configuration file:
  - **white_matter.wm_recipe.layer_profiles** contains the functionality to predict which layer profile to assign to a projection.
  - **white_matter.wm_recipe.p_types** contains the functionality to constrain the single cell logic of individual projections.
  - **white_matter.wm_recipe.projection_mapping** contains the functionality to predict a topographical mapping for a projection, based on the voxelized connectome model of the Allen Institute.
  - **white_matter.wm_recipe.projection_strength** contains the functionality to predict the average density of projection synapses for individual projection classes. Also based on the voxelized connectome model.
  - **white_matter.wm_recipe.synapse_types** determines the physiological parameters prescribed for synapses in projections.
  - **white_matter.wm_recipe.writers** contains the functionality to wrap the above together and write it to a yaml file.
  - **white_matter.wm_recipe.yaml** and **white_matter.wm_recipe.img** are data directories that contain static yaml and image files respectively.

4 Appendix

4.1 Configuration parameters

4.1.1 BrainParcellation

Defaults:

```
{
    "region_names": ["FRP","MOs","ACAd","ACAv","PL","ILA","ORBl","ORBm","ORBvl",
                           "AId","AIv","AIp","GU","VISC",
                           "SSs","SSp-bfd","SSp-tr","SSp-ll","SSp-ul","SSp-un",
                           "SSp-n","SSp-m","MOp",
                           "VISal","VISl","VISp","VISpl","VISli","VISpor","VISrl",
                           "VISa","VISam","VISpm","RSPagl","RSPd","RSPv",
                           "AUDd","AUDp","AUDpo","AUDv","TEa","PERI","ECT"],
    "module_names": ["prefrontal", "anterolateral", "somatomotor", "visual",
                                "medial", "temporal"],
    "projection_classes": ["23", "4", "5it", "5pt", "6"],
    "module_idx": {"prefrontal": [0, 9], "anterolateral": [9, 14],
```

```
                        "somatomotor": [14, 23], "visual": [23, 30],
                        "medial": [30, 36], "temporal": [36, 43]},
    "class_to_layer": {"23": ["l2", "l3"], "4": ["l4"], "5it": ["l5"],
                           "5pt": ["l5"], "6": ["l6"]},
    "projection_class_fltrs": {"23": {"synapse_type": "EXC"},
                            "4": {"synapse_type": "EXC"},
                           "5it": {"synapse_type": "EXC", "proj_type":
                               "intratelencephalic"},
                           "5pt": {"synapse_type": "EXC", "proj_type":
                               "pyramidal tract"},
                           "6": {"synapse_type": "EXC"}}
}
```

- **region_names**: The list of brain regions for which to calculate the recipe for. Each combination of regions will be parameterized independently.
- **module_names**: The list of brain "modules" (contiguous group of brain regions)
- **projection_classes**: The list of classes of projections from within the same brain region (i.e. layers, etc.)
- **module_idx**: Indices into "region_names" that belong to the indicated module. Example: region_names[0:9] corresponds to module "prefrontal".
- **class_to_layer**: Specifies which layers are covered by the individual "projection_classes".
- **projection_class_fltrs**: Filters to apply to neurons belonging to individual "projection_classes". For example, all projecting neurons are excitatory.

4.1.2 LayerProfiles

Defaults:

```
"LayerProfiles":
{
    "layer_profiles": {
      "source": "digitize",
      "parameters":{
        "filename": "img/layer_profile%d.png",
        "cbar_filename": "img/relative_density_cbar.png",
        "shape": [1, 5],
        "cbar_width": [0, 1635],
        "cbar_height": 30,
        "cbar_values": [0, 3],
        "cbar_kwargs": {},
        "number": 6
      }
    },
```

```
  "frequency_per_source":{
    "source": "digitize",
    "parameters":{
      "filename": "img/relative_frequency_%s.png",
      "cbar_filename": "img/relative_frequency_cbar.png",
      "shape": [6, 1],
      "cbar_width": [15, 1160],
      "cbar_height": 15,
      "cbar_values": {"0.0": 0.0, "0.5": 1.0, "1.0": 3.0},
      "cbar_kwargs": {}
    }
  },
  "frequency_per_module":{
    "source": "digitize",
    "parameters": {
      "filename": "img/module_rel_frequency_%s.png",
      "cbar_filename": "img/relative_frequency_cbar.png",
      "shape": [6, 1],
      "cbar_width": [15, 1160],
      "cbar_height": 15,
      "cbar_values": {"0.0": 0.0, "0.5": 1.0, "1.0": 3.0},
      "cbar_kwargs": {},
      "reorder": [0, 3, 1, 4, 2, 5]
    }
  },
  "layer_profile_number": 6,
  "layer_profile_layers": [["11"], ["12", "13"], ["14"], ["15"], ["16"]],
  "layer_profile_split_23": 1
}
```

- **layer_profiles**: Specifies the actual layer profiles, i.e. where their synapse densities peak.
- **frequency_per_source**: Specifies how often each profile is used by each projection class.
- **frequency_per_module**: Specifies how often each profile is used by each module.
- ***/source**: The above entries can be filled in either directly ("source": "config") or digitized from image plots ("source": "digitize"). In both cases, the associated parameters go into the structure under "parameters". (For details, see below.)
- **layer_profile_number**: The number of layer profiles to use.
- **layer_profile_layers**: Which actual layer each point in a layer profile corresponds to.
- **layer_profile_split_23**: (Deprecated) Whether layers 2 and 3 share a single entry in the profiles (layer 2/3).

4.1.3 PTypes

Defaults:

```
"PTypes":
  {
    "23":
    {
      "mat_tree_topology": "normalized connection density",
      "mat_predict_innervation": "normalized connection strength",
      "func_predict_innervation": "(x ** 0.5) / 2",
      "json_cache": "tree_model_final_23.json",
      "h5_cache": "tree_model_final.h5",
      "h5_dset": "23",
      "p_func": "10 ** -x"
    },
    "4":
    {
      "mat_tree_topology": "normalized connection density",
      "mat_predict_innervation": "normalized connection strength",
      "func_predict_innervation": "(x ** 0.5) / 3",
      "json_cache": "tree_model_final_4.json",
      "h5_cache": "tree_model_final.h5",
      "h5_dset": "4",
      "p_func": "10 ** -x"
    },
    "5it":
    {
      "mat_tree_topology": "normalized connection density",
      "mat_predict_innervation": "normalized connection strength",
      "func_predict_innervation": "(x ** 0.5) / 4.5",
      "json_cache": "tree_model_final_5it.json",
      "h5_cache": "tree_model_final.h5",
      "h5_dset": "5it",
      "p_func": "10 ** -x"
    },
    "5pt":
    {
      "mat_tree_topology": "normalized connection density",
      "mat_predict_innervation": "normalized connection strength",
      "func_predict_innervation": "(x ** 0.5) / 3",
      "json_cache": "tree_model_final_5pt.json",
      "h5_cache": "tree_model_final.h5",
      "h5_dset": "5pt",
      "p_func": "10 ** -x"
    },
    "6":
    {
      "mat_tree_topology": "normalized connection density",
```

```
    "mat_predict_innervation": "normalized connection strength",
    "func_predict_innervation": "(x ** 0.5) / 2",
    "json_cache": "tree_model_final_6.json",
    "h5_cache": "tree_model_final.h5",
    "h5_dset": "6",
    "p_func": "10 ** -x"
  }
}
```

Each projection class is independently parameterized.

- **mat_tree_topology**: Which measurement of the Allen voxelized connectivity model is used to generate the topology of the p-type generating tree.
- **mat_predict_innervation**: Which measurement of the Allen voxelized connectivity model is used to fit the edge lengths of the p-type generating tree.
- **func_predict_innervation**: Function to predict the first order innervation probabilities from the measurement specified in **mat_predict_innervation**
- **json_cache**: A cache file to store the topology of the p-type generating tree. Non-absolute paths will be interpreted relative to the location of the configuration file.
- **h5_cache**: A cache file to store the edge lengths of the p-type generating tree. Non-absolute paths will be interpreted relative to the location of the configuration file.
- **h5_dset**: The dataset within **h5_cache** to use to store the edge lengths
- **p_func**: Function that returns the probability to cross an edge from its length in the p-type generating model

4.1.4 ProjectionMapping

Defaults:

```
"ProjectionMapping":
  {
    "class": "VoxelArrayBaryMapper",
    "target_args": {"normalize": 0},
    "pp_use": {"exponent": 1.0, "relative_cutoff": [95, 0.25], "equalize":
        [0.175, 0.5]},
    "pp_display": {"exponent": 1.0, "relative_cutoff": [95, 0.25], "equalize":
        [0.175, 0.5]},
    "fit_args": {"exponent": 1.0, "mul_overlap": 10.0, "mul_angle": 10.0},
    "cache_manifest": "data/connectivity/voxel_model_manifest.json",
    "prepare_args": {},
    "plot_dir": "plots/white_matter/mapping",
    "plot_extension": ".png",
```

```
  "h5_fn": "voxel_mappings.h5",
  "hemi_mirror_at": 136
}
```

- **class**: Which basic method to use to predict the mapping. There are two options: "VoxelArrayBaryMapper" uses the voxelized connectivity model and "VoxelNodeBaryMapper" uses raw, voxelized experiment data. The second (non-default) option yields better results for source regions with many experiments, but unusable results for regions with few experiments.
- **target_args**: Do not change this
- **pp_use**: Post-processing done on the image generated for a projection (N_{src}^{tgt}).
  - exponent: innervation strength is taken to this power (enhances contrast)
  - relative_cutoff: How it is determined that a part of a target region is not innervated at all. The first argument is a percentile of the strengths of the projection over pixels of the target region. The second specifies a fraction of that percentile. Any part of the target region where the projection strength is larger or equal to the resulting value is considered fully innervated. Where the strength is lower it is considered weakly and eventually not innervated.
  - equalize: Specifies equalization of the individual color channels. Let the two parameters be a and b. Further, let the relative strengths in the three color channels, integrated over a target region be $[S_r, S_g, S_b]$, i.e. they sum up to 1. If $min([S_r, S_g, S_b]) < a$ then the channels are scaled such that the relative contributions become $nrmlz([S_r, S_g, S_b] * *b)$, where $nrmlz$ normalizes the result to a sum of 1.
- **pp_display**: If for some reason you want different post-processing for the plots of M_{src}^{tgt}.
- **fit_args**: Parameterizes how N_{src}^{tgt} is fit from M_{src}^{tgt}.
- **cache_manifest**: Where the manifest file used by the Allen Institute's *mouse-connectivity-models* package is kept. Data downloaded from the Allen Institute's servers will be placed at the same location. Non-absolute paths will be interpreted relative to the location of the configuration file.
- **prepare_args**: Must be empty.
- **plot_dir**: Where the plots of M_{src}^{tgt} will be placed
- **plot_extension**: The format used for the plots
- **h5_fn**: A file to cache the mapping results. Non-absolute paths will be interpreted relative to the location of the configuration file.

- **hemi_mirror_at**: In the flat map projection of brain regions, x-coordinates smaller than this will be considered in the left hemisphere, others in the right hemisphere.

4.1.5 ProjectionStrength

Defaults:

```
"ProjectionStrength":
  {
    "per_projection_class_ipsi":{
      "source": "config",
      "parameters": {
        "patterns": [[...]],
        "vals_nan": [-100],
        "values": "array"
      }
    },
    "per_projection_class_contra":{
      "source": "config",
      "parameters": {
        "patterns": [[...]],
        "vals_nan": [-100],
        "values": "array"
      }
    },
    "module_separators_source": [0, 8, 9, 17, 22, 26, 27],
    "module_separators_target": [ 0, 9, 14, 23, 30, 36, 43],
    "threshold_fraction": 0.05,
    "scaling": {"region": "SSp-ll", "value": 0.05534342,
                "measurement": "connection density"},
    "cache_manifest": "data/connectivity/voxel_model_manifest.json",
    "h5_cache": "cache/strength/projection_matrices.h5"
  }
```

- **per_projection_class_ipsi**: Parameterizes the strengths of ipsi-lateral projections from individual projection classes. (The total strength from all classes is taken from the mcmodels package instead). As before, can be done using "source": "config" or "source": "digitize".
- **per_projection_class_contra**: As above, for contra-lateral projections.
- **module_separators_source**: At what indices to split the "per_projection_class*" matrices into modules along the vertical axis. This overrides "module_idx" in section "BrainParcellation" for this specific use case.
- **module_separates_target**: See above, but for splitting along the horizontal axis.

- **threshold_fraction**: This fraction of synapses will be lost when the weakest projection pathways are removed.
- **scaling**: The projection matrix of the specified "measurement" is scaled such that the entry corresponding to connectivity within "region" is the specified "value". The default is calibrated such that overall the biological synapse density is reached (Schuz and Palm).
- **cache_manifest**: See the same entry under "ProjectionMapping"
- **h5_cache**: A file to cache the projection strengths. Non-absolute paths will be interpreted relative to the location of the configuration file.

4.1.6 SynapseTypes

Defaults:

```
"SynapseTypes":
  {
    "synapse_type_yaml": "../yaml/synapse_type.yaml",
    "synapse_type_mapping":{
      "23": "type_1",
      "4": "type_2",
      "5pt": "type_3",
      "5it": "type_4",
      "6": "type_5"
    }
  }
```

- **synapse_type_mapping**: Specifies which projection type uses which synapse type (for synapse types see below).
- **synapse_type_yaml**: Path to a yaml file that specifies the physiological parameters of each synapse type (see above). Non-absolute paths are interpreted relative to the installed location of the *white_matter.wm_recipe.SynapseTypes* package. Note that synapse physiological is just a placeholder in this version!

4.2 Parameterizing directly in the config

Some biological data points are directly given in the config file, such as the layer profiles of synapse densities. Parameters are the following:

- **patterns**: The actual data goes here. Must be a dict.
- **vals_nan**: The data points equal to this value will be set to NaN.
- **values**: If set to "array", the values of the dict will be converted to numpy.arrays. Don't change this.

- **keys**: What to convert the keys of the dict into. Can be "float", "int", or "str". Don't change this.

4.3 Parameterizing by digitizing from image plots

In the default configuration, all required data are directly specified in the configuration file ("source": "config"). Alternatively, they can be digitized from image plots, such as the ones in Harris et al., 2018. Example for layer profiles:

```
{
      "source": "digitize",
      "parameters":{
        "filename": "img/layer_profile%d.png",
        "cbar_filename": "img/relative_density_cbar.png",
        "shape": [1, 5],
        "cbar_width": [0, 1635],
        "cbar_height": 30,
        "cbar_values": [0, 3],
        "cbar_kwargs": {},
        "number": 6
}
```

- **filename**: Pattern for the file names of image files with the data to digitize (here layer profiles).
- **cbar_filename**: file name of the image file of the color bar for the plots.
- **shape**: Shape and number of data points to digitize.
- **cbar_width**: Indices from and to where in the color bar image data points shall be sampled. 100 data points between the indices will be sampled.
- **cbar_height**: At what height in the color bar image the data points shall be sampled. Note: This example with a list of length 2 for cbar_widht and a single entry for cbar_height is for a horizontal color bar. For a vertical color bar it is the other way around.
- **cbar_values**: The actual data values associated with the points sampled from the color bar. If it is a list of length 2, they will be linearly interpolated between the two values. If it is a dict, the values of the dict will be piecewise interpolated between its keys.
- **number**: The number of images to digitize. In this example, six layer profiles will be digitized: img/layer_profile1.png, ..., img/layer_profile6.png.
- **cbar_kwargs**: Possible entries:

– vals_nan: a list of RGB values (list of integers between 0 and 255 of length 3) that are associated with NaN.