# *Implement a new Neural Network Data Point*

*The BRAPH 2 Developers*

*September 13, 2023*

This is the developer tutorial for implementing a new neural network data point. In this Tutorial, we will explain how to create the generator file `*.gen.m` for a new neural network data point, which can then be compiled by `braph2genesis`. All kinds of neural network data point are (direct or indirect) extensions of the base element `NNDataPoint`. Here, we will use as examples the neural network data point `NNDataPoint_CON_REG` (connectivity data for regression), `NNDataPoint_CON_CLA` (connectivity data for classification) `NNDataPoint_Graph_REG` (adjacency matrix for regression), `NNDataPoint_Graph_CLA` (adjacency matrix for classification), `NNDataPoint_Measure_REG` (graph measure for regression), and `NNDataPoint_Measure_CLA` (graph measure for classification).

## *Contents*

## *Implementation of a Data Point with Connectivity Data*

### *Connectivity Data Point for Regression (*NNDataPoint_CON_REG*)*

We will start by implementing in detail NNDataPoint_CON_REG, which is a direct extension of NNDataPoint. A data point for regression with connectivity data NNDataPoint_CON_REG contains the input and target for neural network analysis with a subject with connectivity data (SubjectCON), where the input is the subject's connectivity data and the target is the subject's variables of interest.

Code 1: **NNDataPoint_CON_REG element header.** The header section of the generator code for _NNDataPoint_CON_REG.gen.m provides the general information about the NNDataPoint_CON_REG element.

```
%% iheader!
NNDataPoint_CON_REG < NNDataPoint (dp, connectivity regression data point)
     is a data point for regression with connectivity data.
 (1)

%%% idescription!
A data point for regression with connectivity data (NNDataPoint_CON_REG)
contains the input and target for neural network analysis with a subject
     with connectivity data (SubjectCON).
The input is the connectivity data of the subject.
The target is obtained from the variables of interest of the subject.
```

(1) defines NNDataPoint_CON_REG as a subclass of NNDataPoint. The moniker will be dp.

Code 2: **NNDataPoint_CON_REG element prop update.** The props_update section of the generator code for _NNDataPoint_CON_REG.gen.m updates the properties of the NNDataPoint_CON_REG element. This defines the core properties of the data point.

```
%% iprops_update!

%%% iprop!
NAME (constant, string) is the name of a data point for regression with
     connectivity data.
%%%% idefault!
'NNDataPoint_CON_REG'

%%% iprop!
DESCRIPTION (constant, string) is the description of a data point for
     regression with connectivity data.
%%%% idefault!
'A data point for regression with connectivity data (NNDataPoint_CON_REG)
     contains the input and target for neural network analysis with a
     subject with connectivity data (SubjectCON). The input is the
     connectivity data of the subject. The target is obtained from the
     variables of interest of the subject.'

%%% iprop!
TEMPLATE (parameter, item) is the template of a data point for regression
     with connectivity data.
%%%% isettings!
'NNDataPoint_CON_REG'
```

```
17
18 %%% iprop!
19 ID (data, string) is a few-letter code for a data point for regression with
       connectivity data.
20 %%%% idefault!
21 'NNDataPoint_CON_REG ID'
22
23 %%% iprop!
24 LABEL (metadata, string) is an extended label of a data point for regression
        with connectivity data.
25 %%%% idefault!
26 'NNDataPoint_CON_REG label'
27
28 %%% iprop!
29 NOTES (metadata, string) are some specific notes about a data point for
       regression with connectivity data.
30 %%%% idefault!
31 'NNDataPoint_CON_REG notes'
32
33 %%% iprop! (1)
34 INPUT (result, cell) is the input value for this data point.
35 %%%% icalculate!
36 value = {dp.get('SUB').get('CON')};
37
38 %%% iprop! (2)
39 TARGET (result, cell) is the target value for this data point.
40 %%%% icalculate!
41 value = cellfun(@(x) dp.get('SUB').get('VOI_DICT').get('IT', x).get('V'), dp
       .get('TARGET_IDS'), 'UniformOutput', false);
```

(1) The property `INPUT` is the input value for this data point, which is obtained directly from the connectivity of `Subject_CON` by the code under `icalculate!`.

(2) The property `TARGET` is the target value for this data point, which is obtained directly from the variables of interest of `VOI_DICT` by the code under `icalculate!`.

Code 3: **NNDataPoint_CON_REG element props.** The `props` section of generator code for `_NNDataPoint_CON_REG.gen.m` defines the properties to be used in `NNDataPoint_CON_REG`.

```
1 %% iprops!
2
3 %%% iprop! (1)
4 SUB (data, item) is a subject with connectivity data.
5 %%%% isettings!
6 'SubjectCON'
7
8 %%% iprop! (2)
9 TARGET_IDS (parameter, stringlist) is a list of variable-of-interest IDs to
       be used as regression targets.
```

(1) The property `SUB` is a subject with connectivity data (`Subject_CON`), which is used to calculated the mentioned properties `INPUT` and `TARGET`.

(2) The property `TARGET_IDS` defines the targets, where the targets should be from the subject's variable-of-interest IDs.

Code 4: **NNDataPoint_CON_REG element tests.** The `tests` section from the element generator `_NNDataPoint_CON_REG.gen.m`. A test for creating example files should be prepared to test the properties of the data point. Furthermore, additional test should be prepared for validating the value of input and target for the data point.

```
%% itests!

%%% iexcluded_props! (1)
[NNDataPoint_CON_REG.SUB]

%%% itest!
%%%% iname!
Create example files for regression (2)
%%%% icode!
data_dir = [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data NN
      REG CON XLS'];
if ~isdir(data_dir)
    mkdir(data_dir);

    % Brain Atlas (3)
    im_ba = ImporterBrainAtlasXLS('FILE', 'desikan_atlas.xlsx');
    ba = im_ba.get('BA');
    ex_ba = ExporterBrainAtlasXLS( ...
        'BA', ba, ...
        'FILE', [data_dir filesep() 'atlas.xlsx'] ...
        );
    ex_ba.get('SAVE')
    N = ba.get('BR_DICT').get('LENGTH');

    % saves RNG
    rng_settings_ = rng(); rng('default')

    sex_options = {'Female' 'Male'};

    % Group (4)
    K = 2; % degree (mean node degree is 2K)
    beta = 0.3; % Rewiring probability
    gr_name = 'CON_Group_XLS';
    gr_dir = [data_dir filesep() gr_name];
    mkdir(gr_dir);
    vois = [
        {{'Subject ID'} {'Age'} {'Sex'}}
        {{} {} cell2str(sex_options)}
        ];
    for i = 1:1:100 % subject number
        sub_id = ['SubjectCON_' num2str(i)];
        % create WS graphs with random beta
        beta(i) = rand(1); (5)
        h = WattsStrogatz(N, K, beta(i)); % create WS graph (6)

        A = full(adjacency(h)); A(1:length(A)+1:numel(A)) = 0; % extract the
      adjacency matrix
        r = 0 + (0.5 - 0) * rand(size(A)); diffA = A - r; A(A ~= 0) = diffA(
     A ~= 0); % make the adjacency matrix weighted
        A = max(A, transpose(A)); % make the adjacency matrix symmetric

        writetable(array2table(A), [gr_dir filesep() sub_id '.xlsx'], '
```

(1) List of properties that are excluded from testing.

(2) creates the example connectivity data files for regression analysis.

(3) creates and exports the brain atlas file to the example directory.

(4) creates one group of subjects with specified degree and rewiring probability configurations.

(5) generates random rewiring probability settings for each subject.

(6) and (10) utilize the provided degree and rewiring probability settings to generate corresponding Watts-Strogatz model graphs.

```
     WriteVariableNames', false) (7)
50
51       % variables of interest
52       age_upperBound = 80;
53       age_lowerBound = 50;
54       age = age_lowerBound + beta(i)*(age_upperBound - age_lowerBound);
     (8)
55       vois = [vois; {sub_id, age, sex_options(randi(2))}];
56   end
57   writetable(table(vois), [data_dir filesep() gr_name '.vois.xlsx'], '
     WriteVariableNames', false) (9)
58
59   % reset RNG
60   rng(rng_settings_)
61 end
62 %%% itest_functions!
63 function h = WattsStrogatz(N, K, beta) (10)
64 % H = WattsStrogatz(N,K,beta) returns a Watts-Strogatz model graph with N
65 % nodes, N*K edges, mean node degree 2*K, and rewiring probability beta.
66 %
67 % beta = 0 is a ring lattice, and beta = 1 is a random graph.
68
69 % Connect each node to its K next and previous neighbors. This constructs
70 % indices for a ring lattice.
71     s = repelem((1:N)', 1, K);
72     t = s + repmat(1:K, N, 1);
73     t = mod(t - 1, N) + 1;
74
75     % Rewire the target node of each edge with probability beta
76     for source = 1:N
77         switchEdge = rand(K, 1) < beta;
78
79         newTargets = rand(N, 1);
80         newTargets(source) = 0;
81         newTargets(s(t == source)) = 0;
82         newTargets(t(source, ~switchEdge)) = 0;
83
84         [~, ind] = sort(newTargets, 'descend');
85         t(source, switchEdge) = ind(1:nnz(switchEdge));
86     end
87
88     h = graph(s,t);
89 end
90
91 %%% itest!
92 %%%% iname! (11)
93 Create a NNDataset containg NNDataPoint_CON_REG with simulated data
94 %%%% icode!
95 % Load BrainAtlas
96 im_ba = ImporterBrainAtlasXLS( ...
97     'FILE', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data
      NN REG CON XLS' filesep 'atlas.xlsx'], ...
98     'WAITBAR', true ...
99     );
100
101 ba = im_ba.get('BA');
102
103 % Load Group of SubjectCON
104 im_gr = ImporterGroupSubjectCON_XLS( ...
```

(7) exports the adjacency matrix of the graph to an Excel file.

(8) associates the age value with each individual rewiring probability setting.

(9) exports the variables of interest to an Excel file.

(11) validates the data point by using assertions to confirm that the input and target calculated values match the connectivity data and the variables of interest in the example files.

```
105     'DIRECTORY', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example
         data NN REG CON XLS' filesep 'CON_Group_XLS'], ...
106     'BA', ba, ...
107     'WAITBAR', true ...
108     );
109
110 gr = im_gr.get('GR');
111
112 % create an item list of NNDataPoint_CON_REG (12)
113 it_list = cellfun(@(x) NNDataPoint_CON_REG( ...
114     'ID', x.get('ID'), ...
115     'SUB', x, ...
116     'TARGET_IDS', x.get('VOI_DICT').get('KEYS')), ...
117     gr.get('SUB_DICT').get('IT_LIST'), ...
118     'UniformOutput', false);
119
120 % create a NNDataPoint_CON_REG DICT (13)
121 dp_list = IndexedDictionary(...
122         'IT_CLASS', 'NNDataPoint_CON_REG', ...
123         'IT_LIST', it_list ...
124         );
125
126 % create a NNDataset containing the NNDataPoint_CON_REG DICT (14)
127 d = NNDataset( ...
128     'DP_CLASS', 'NNDataPoint_CON_REG', ...
129     'DP_DICT', dp_list ...
130     );
131
132 % Check whether the number of inputs matches (14)
133 assert(length(d.get('INPUTS')) == gr.get('SUB_DICT').get('LENGTH'), ...
134     [BRAPH2.STR ':NNDataPoint_CON_REG:' BRAPH2.FAIL_TEST], ...
135     'NNDataPoint_CON_REG does not construct the dataset correctly. The
         number of the inputs should be the same as the number of imported
         subjects.' ...
136     )
137
138 % Check whether the number of targets matches (15)
139 assert(length(d.get('TARGETS')) == gr.get('SUB_DICT').get('LENGTH'), ...
140     [BRAPH2.STR ':NNDataPoint_CON_REG:' BRAPH2.FAIL_TEST], ...
141     'NNDataPoint_CON_REG does not construct the dataset correctly. The
         number of the targets should be the same as the number of imported
         subjects.' ...
142     )
143
144 % Check whether the content of input for a single datapoint matches (16)
145 for index = 1:1:gr.get('SUB_DICT').get('LENGTH')
146     individual_input = d.get('DP_DICT').get('IT', index).get('INPUT');
147     known_input = {gr.get('SUB_DICT').get('IT', index).get('CON')};
148
149     assert(isequal(individual_input, known_input), ...
150         [BRAPH2.STR ':NNDataPoint_CON_REG:' BRAPH2.FAIL_TEST], ...
151         'NNDataPoint_CON_REG does not construct the dataset correctly. The
          input value is not derived correctly.' ...
152         )
153 end
154
155 %%% itest!
```

(12), (13), and (14) creates an item list for the data points, subsequently generates the data point dictionary using the list, and then constructs the neural network dataset containing these data points.

(14) tests the number of inputs from the dataset matches the number of subjects in the group.

(15) tests the number of targets from the dataset matches the number of subjects in the group.

(16) tests the value of each input from the data point matches the subject's connectivity data.

```
%%%% iname! (17)
Example training-test regression
%%%% icode!
% ensure the example data is generated
if ~isfile([fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data NN
      REG CON XLS' filesep 'atlas.xlsx'])
    test_NNDataPoint_CON_REG % create example files
end

example_NN_CON_REG

%%% itest!
%%%% iname! (18)
Example cross-validation regression
%%%% icode!
% ensure the example data is generated
if ~isfile([fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data NN
      REG CON XLS' filesep 'atlas.xlsx'])
    test_NNDataPoint_CON_REG % create example files
end

example_NNCV_CON_REG
```

(17) and (18) executes the corresponding example scripts to ensure the functionalities.

*Connectivity Data Point for Classification (*`NNDataPoint_CON_CLA`*)*

### *Implementation of a Data Point with Graphs*

#### *Graph Data Point for Regression (*`NNDataPoint_Graph_REG`*)*

### *Implementation of a Data Point with Graph Measures*

*Graph Measure Data Point for Classification (*`NNDataPoint_Measure_CLA`*)*