

# *Implement a New Neural Network Regressor*

*The BRAPH 2 Developers*

*October 16, 2024*

This is the developer tutorial for implementing a new neural network regressor. In this tutorial, you will learn how to create the generator file `*.gen.m` for a new neural network regressor, which can then be compiled by `braph2genesis`. All kinds of neural network models are (direct or indirect) extensions of the base element `NNBase`. Here, you will use as example the neural network regressor `NNRegressorMLP` (multi-layer perceptron regressor).

## *Contents*

## Implementation of a neural network regressor (NNRegressorMLP)

You will start by implementing in detail NNRegressorMLP, which is a direct extension of NNBase. A multi-layer perceptron regressor NNRegressorMLP comprises a multi-layer perceptron regressor model and a given dataset.

**Code 1: NNRegressorMLP element header.** The header section of the generator code in `_NNRegressorMLP.gen.m` provides the general information about the NNRegressorMLP element.

---

```

1 %% iheader!
2 NNRegressorMLP < NNBase (nn, multi-layer perceptron regressor) comprises a
    multi-layer perceptron regressor model and a given dataset. ①
3
4 %%% idescription!
5 A neural network multi-layer perceptron regressor (NNRegressorMLP) comprises
    a multi-layer perceptron regressor model and a given dataset.
6 NNRegressorMLP trains the multi-layer perceptron regressor with a formatted
    inputs ("CB", channel and batch) derived from the given dataset.
7
8 %%% ibuild!
9 1

```

---

① defines NNRegressorMLP as a subclass of NNBase. The moniker will be nn.

**Code 2: NNRegressorMLP element prop update.** The `props_update` section of the generator code in `_NNRegressorMLP.gen.m` updates the properties of the NNRegressorMLP element. This defines the core properties of the data point.

---

```

1 %% iprops_update!
2
3 %%% iprop!
4 NAME (constant, string) is the name of the neural network multi-layer
    perceptron regressor.
5 %%% idefault!
6 'NNRegressorMLP'
7
8 %%% iprop!
9 DESCRIPTION (constant, string) is the description of the neural network
    multi-layer perceptron regressor.
10 %%% idefault!
11 'A neural network multi-layer perceptron regressor (NNRegressorMLP)
    comprises a multi-layer perceptron regressor model and a given dataset.
    NNRegressorMLP trains the multi-layer perceptron regressor with a
    formatted inputs ("CB", channel and batch) derived from the given
    dataset.'
12
13 %%% iprop!
14 TEMPLATE (parameter, item) is the template of the neural network multi-layer
    perceptron regressor.
15 %%% isettings!
16 'NNRegressorMLP'
17
18 %%% iprop!
19 ID (data, string) is a few-letter code for the neural network multi-layer
    perceptron regressor.
20 %%% idefault!

```

```

21 'NNRegressorMLP ID'
22
23 %%% iprop!
24 LABEL (metadata, string) is an extended label of the neural network multi-
    layer perceptron regressor.
25 %%% idefault!
26 'NNRegressorMLP label'
27
28 %%% iprop!
29 NOTES (metadata, string) are some specific notes about the neural network
    multi-layer perceptron regressor.
30 %%% idefault!
31 'NNRegressorMLP notes'
32
33 %%% iprop! ①
34 D (data, item) is the dataset to train the neural network model, and its
    data point class DP_CLASS defaults to one of the compatible classes
    within the set of DP_CLASSES.
35 %%% isettings!
36 'NNDataset'
37 %%% idefault!
38 NNDataset('DP_CLASS', 'NNDatapoint_CON_REG')
39
40 %%% iprop!
41 DP_CLASSES (parameter, classlist) is the list of compatible data points.
42 %%% idefault! ②
43 {'NNDatapoint_CON_REG' 'NNDatapoint_CON_FUN_MP_REG' 'NNDatapoint_Graph_REG'
    'NNDatapoint_Measure_REG'}
44
45 %%% iprop!
46 INPUTS (query, cell) constructs the data in the CB (channel-batch) format.
47 %%% icalculate! ③
48 % inputs = nn.get('inputs', D) returns a cell array with the
49 % inputs for all data points in dataset D.
50 if isempty(varargin)
51     value = {};
52     return
53 end
54 d = varargin{1};
55 inputs_group = d.get('INPUTS');
56 if isempty(inputs_group)
57     value = {};
58 else
59     flattened_inputs_group = [];
60     for i = 1:length(inputs_group)
61         inputs_individual = inputs_group{i};
62         flattened_inputs_individual = [];
63         while ~isempty(inputs_individual)
64             currentData = inputs_individual{end}; % Get the last element
        from the stack
65             inputs_individual = inputs_individual(1:end-1); % Remove the
        last element
66
67             if iscell(currentData)
68                 % If it's a cell array, add its contents to the stack
69                 inputs_individual = [inputs_individual currentData{:}];
70             else
71                 % If it's numeric or other data, append it to the vector
72                 flattened_inputs_individual = [currentData(:);
        flattened_inputs_individual];

```

① defines NNDataset which contains the NNDatapoint to train this regressor.

② defines the compatible NNDatapoint classes with this NNRegressorMLP.

③ is a query that transforms the input data of NNDatapoint to the CB (channel-batch) format by flattening its included cells.

```

73         end
74     end
75     flattened_inputs_group = [flattened_inputs_group;
76         flattened_inputs_individual'];
77     end
78     value = {flattened_inputs_group};
79 end
80 %%% iprop!
81 TARGETS (query, cell) constructs the targets in the CB (channel-batch)
82     format.
83 %%% icalculate! ④
84 % targets = nn.get('PREDICT', D) returns a cell array with the
85 % targets for all data points in dataset D.
86 if isempty(varargin)
87     value = {};
88 return
89 end
90 d = varargin{1};
91 targets = d.get('TARGETS');
92 if isempty(targets)
93     value = {};
94 else
95     nn_targets = [];
96     for i = 1:length(targets)
97         target = cell2mat(targets{i});
98         nn_targets = [nn_targets; target(:)'];
99     end
100     value = {nn_targets};
101 end
102 %%% iprop!
103 MODEL (result, net) is a trained neural network model.
104 %%% icalculate! ⑤
105 inputs = cell2mat(nn.get('INPUTS', nn.get('D'))); ⑥
106 targets = cell2mat(nn.get('TARGETS', nn.get('D'))); ⑦
107 if isempty(inputs) || isempty(targets)
108     value = network();
109 else
110     number_features = size(inputs, 2);
111     number_targets = size(targets, 2);
112     layers = nn.get('LAYERS'); ⑧
113
114     nn_architecture = [featureInputLayer(number_features, 'Name', 'Input')];
115     for i = 1:length(layers)
116         nn_architecture = [nn_architecture
117             fullyConnectedLayer(layers(i), 'Name', ['Dense_' num2str(i)])
118             batchNormalizationLayer('Name', ['BatchNormalization_' num2str(i)
119             ])
120             dropoutLayer('Name', ['Dropout_' num2str(i)])
121         ];
122     end
123     nn_architecture = [nn_architecture
124         reluLayer('Name', 'Relu_output')
125         fullyConnectedLayer(number_targets, 'Name', 'Dense_output')
126         regressionLayer('Name', 'Output')
127     ];
128 % specify trianing options ⑨
129 options = trainingOptions( ...

```

④ is a query that collects all the target values from all data points.

⑤ trains the regressor.

⑥ and ⑦ extract the inputs and targets.

⑧ defines the neural network architecture with user-specified number of neurons and number of layers.

⑨ defines the neural network training options.

```

130     nn.get('SOLVER'), ...
131     'MiniBatchSize', nn.get('BATCH'), ...
132     'MaxEpochs', nn.get('EPOCHS'), ...
133     'Shuffle', nn.get('SHUFFLE'), ...
134     'Plots', nn.get('PLOT_TRAINING'), ...
135     'Verbose', nn.get('VERBOSE') ...
136 );
137
138 % train the neural network ⑩
139 value = trainNetwork(inputs, targets, nn_architecture, options);
140 end

```

---

⑩ trains the model with those parameters and the neural network architecture.

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

---

```

1 %% iprops!
2
3 %% iprop! ①
4 LAYERS (data, rvector) defines the number of layers and their neurons.
5 %%%% idefault!
6 [32 32]
7 %%%% igui!
8 pr = PanelPropRVectorSmart('EL', nn, 'PROP', NNRegressorMLP.LAYERS, ...
9     'MIN', 0, 'MAX', 2000, ...
10    'DEFAULT', NNRegressorMLP.getPropDefault('LAYERS'), ...
11    varargin{:});
12
13 %%% iprop!
14 WAITBAR (gui, logical) determines whether to show the waitbar.
15 %%%% idefault!
16 true

```

---

① defines the number of neuron per layer. For example, [32 32] represents two layers, each containing 32 neurons.

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

---

```

1 %% itests!
2
3 %%% itest!
4 %%% iname!
5 train the regressor with example data
6 %%% icode!
7
8 % ensure the example data is generated
9 if ~isfile([fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data NN
  REG CON XLS' filesep 'atlas.xlsx'])
10     test_NNDataPoint_CON_REG % create example files
11 end
12
13 % Load BrainAtlas
14 im_ba = ImporterBrainAtlasXLS( ...
15     'FILE', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data
  NN REG CON XLS' filesep 'atlas.xlsx'], ...
16     'WAITBAR', true ...
17 );
18
19 ba = im_ba.get('BA');
20
21 % Load Groups of SubjectCON
22 im_gr = ImporterGroupSubjectCON_XLS( ...
23     'DIRECTORY', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example
  data NN REG CON XLS' filesep 'CON_Group_XLS'], ...
24     'BA', ba, ...
25     'WAITBAR', true ...
26 );
27
28 gr = im_gr.get('GR');
29
30 % create a item list of NNDataPoint_CON_REG
31 it_list = cellfun(@(x) NNDataPoint_CON_REG( ...
32     'ID', x.get('ID'), ...
33     'SUB', x, ...
34     'TARGET_IDS', x.get('VOI_DICT').get('KEYS')), ...
35     gr.get('SUB_DICT').get('IT_LIST'), ...
36     'UniformOutput', false);
37
38 % create a NNDataPoint_CON_REG DICT
39 dp_list = IndexedDictionary(...
40     'IT_CLASS', 'NNDataPoint_CON_REG', ...
41     'IT_LIST', it_list ...
42 );
43
44 % create a NNData containing the NNDataPoint_CON_REG DICT
45 d = NNDataset( ...
46     'DP_CLASS', 'NNDataPoint_CON_REG', ...
47     'DP_DICT', dp_list ...
48 );
49
50 nn = NNRegressorMLP('D', d, 'LAYERS', [20 20]);
51 trained_model = nn.get('MODEL');
```

```
52
53 % Check whether the number of fully-connected layers matches (excluding
    Dense_output layer) ①
54 assert(length(nn.get('LAYERS')) == sum(contains({trained_model.Layers.Name},
    'Dense')) - 1, ...
55 [BRAPH2.STR ':NNRegressorMLP:' BRAPH2.FAIL_TEST], ...
56 'NNRegressorMLP does not construct the layers correctly. The number of
    the inputs should be the same as the length of dense layers the
    property.' ...
57 )
```

---

① checks whether the number of layers from the trained model is correctly set.