Implement a new Neural Network Regressor

The BRAPH 2 Developers

September 20, 2023

This is the developer tutorial for implementing a new neural network regressor. In this Tutorial, we will explain 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, we will use as examples the neural network regressor `NNRegressorMLP` (multi-layer perceptron regressor).

Contents

Implementation of a Neural Network Regressor

We 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 for `_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.
```

(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 for `_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 = 1:1: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];
73             end
74         end
75         flattened_inputs_group = [flattened_inputs_group;
```

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.

```matlab
            flattened_inputs_individual'];
76      end
77      value = {flattened_inputs_group};
78 end
79
80 %%% ¡prop!
81 TARGETS (query, cell) constructs the targets in the CB (channel-batch)
         format.
82 %%%% ¡calculate!  ④
83 % targets = nn.get('PREDICT', D) returns a cell array with the
84 %  targets for all data points in dataset D.
85 if isempty(varargin)
86     value = {};
87     return
88 end
89 d = varargin{1};
90 targets = d.get('TARGETS');
91 if isempty(targets)
92     value = {};
93 else
94     nn_targets = [];
95     for i = 1:1:length(targets)
96         target = cell2mat(targets{i});
97         nn_targets = [nn_targets; target(:)'];
98     end
99     value = {nn_targets};
100 end
101
102 %%% ¡prop!
103 MODEL (result, net) is a trained neural network model.
104 %%%% ¡calculate!  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 = 1:1:length(layers)
116         nn_architecture = [nn_architecture
117             fullyConnectedLayer(layers(i), 'Name', ['Dense_' num2str(i)])
118             batchNormalizationLayer('Name', ['BatchNormalization_' num2str(i
     )])
119             dropoutLayer('Name', ['Dropout_' num2str(i)])
120             ];
121     end
122     nn_architecture = [nn_architecture
123         reluLayer('Name', 'Relu_output')
124         fullyConnectedLayer(number_targets, 'Name', 'Dense_output')
125         regressionLayer('Name', 'Output')
126         ];
127
128     % specify trianing options   ⑨
129     options = trainingOptions( ...
130         nn.get('SOLVER'), ...
131         'MiniBatchSize', nn.get('BATCH'), ...
132         'MaxEpochs', nn.get('EPOCHS'), ...
```

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

⑤ trains the regressor with the defined dataset by the code under ¡calculate!.

⑥ and ⑦ firstly extract the inputs and targets with the corresponding format.

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

⑨ defines the neural network training options.

```
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 for _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) detemines whether to show the waitbar.
15 %%%% idefault!
16 true
17
18 %%% iprop!
19 INTERRUPTIBLE (gui, scalar) sets whether the comparison computation is
        interruptible for multitasking.
20 %%%% idefault!
21 .001
22
23 %%% iprop! 2
24 FEATURE_IMPORTANCE (query, cell) evaluates the average significance of each
        feature by iteratively shuffling its values P times and measuring the
        resulting average decrease in model performance.
25 %%%% icalculate!
26 % fi = nn.get('FEATURE_IMPORTANCE', D) retrieves a cell array containing
27 %  the feature importance values for the trained model, as assessed by
28 %  evaluating it on the input dataset D.
29 if isempty(varargin)
30     value = {};
31     return
32 end
33 d = varargin{1};
34 P = varargin{2};
35 seeds = varargin{3};
36
37 inputs = cell2mat(nn.get('INPUTS', d));
38 if isempty(inputs)
39     value = {};
40     return
41 end
42 targets = cell2mat(nn.get('TARGETS', d));
43 net = nn.get('MODEL');
```

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

(2) is a query that calculates the permuation feature importance. Note that, other neural network architectures, such as convolutional neural network, have other techniques to obtain feature importance.

```
44
45  number_features = size(inputs, 2);
46  original_loss = crossentropy(net.predict(inputs), targets);
47
48  wb = braph2waitbar(nn.get('WAITBAR'), 0, ['Feature importance permutation
         ...']);
49
50  start = tic;
51  for i = 1:1:P  (4)
52      rng(seeds(i), 'twister')
53      parfor j = 1:1:number_features  (5)
54          scrambled_inputs = inputs;
55          permuted_value = squeeze(normrnd(mean(inputs(:, j)), std(inputs(:, j
         )), squeeze(size(inputs(:, j))))) + squeeze(randn(size(inputs(:, j))))
         + mean(inputs(:, j));
56          scrambled_inputs(:, j) = permuted_value;
57          scrambled_loss = crossentropy(net.predict(scrambled_inputs), targets
         );
58          feature_importance(j) = scrambled_loss;
59      end
60
61      feature_importance_all_permutations{i} = feature_importance /
         original_loss;
62
63      braph2waitbar(wb, i / P, ['Feature importance permutation ' num2str(i) '
          of ' num2str(P) ' - ' int2str(toc(start)) '.' int2str(mod(toc(start),
         1) * 10) 's ...'])
64      if nn.get('VERBOSE')
65          disp(['** PERMUTATION FEATURE IMPORTANCE - sampling #' int2str(i) '/
         ' int2str(P) ' - ' int2str(toc(start)) '.' int2str(mod(toc(start), 1) *
          10) 's'])
66      end
67      if nn.get('INTERRUPTIBLE')
68          pause(nn.get('INTERRUPTIBLE'))
69      end
70  end
71
72  braph2waitbar(wb, 'close')
73
74  value = feature_importance_all_permutations;
```

(4) and (5) iteratively shuffle the feature values from any given dataset P times and measuring the resulting average decrease in model performance.

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, additional test should be prepared for validating the value of input and target for the data point.

```matlab
%% ¡tests!

%%% ¡test!
%%%% ¡name!
train the regressor with example data
%%%% ¡code!

% 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

% Load BrainAtlas
im_ba = ImporterBrainAtlasXLS( ...
    'FILE', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example data
      NN REG CON XLS' filesep 'atlas.xlsx'], ...
    'WAITBAR', true ...
    );

ba = im_ba.get('BA');

% Load Groups of SubjectCON
im_gr = ImporterGroupSubjectCON_XLS( ...
    'DIRECTORY', [fileparts(which('NNDataPoint_CON_REG')) filesep 'Example
      data NN REG CON XLS' filesep 'CON_Group_XLS'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr = im_gr.get('GR');

% create a item list of NNDataPoint_CON_REG
it_list = cellfun(@(x) NNDataPoint_CON_REG( ...
    'ID', x.get('ID'), ...
    'SUB', x, ...
    'TARGET_IDS', x.get('VOI_DICT').get('KEYS')), ...
    gr.get('SUB_DICT').get('IT_LIST'), ...
    'UniformOutput', false);

% create a NNDataPoint_CON_REG DICT
dp_list = IndexedDictionary(...
        'IT_CLASS', 'NNDataPoint_CON_REG', ...
        'IT_LIST', it_list ...
        );

% create a NNData containing the NNDataPoint_CON_REG DICT
d = NNDataset( ...
    'DP_CLASS', 'NNDataPoint_CON_REG', ...
    'DP_DICT', dp_list ...
    );

nn = NNRegressorMLP('D', d, 'LAYERS', [20 20]);
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.