

Implement a new Neural Network Classifier

The BRAPH 2 Developers

October 16, 2024

This is the developer tutorial for implementing a new neural network classifier. In this tutorial, you will learn how to create the generator file `*.gen.m` for a new neural network classifier, 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 classifier `NNClassifierMLP` (multi-layer perceptron classifier).

Contents

<i>Implementation of a neural network classifier (NNClassifierMLP)</i>	2
--	---

Implementation of a neural network classifier (NNClassifierMLP)

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

Code 1: NNClassifierMLP element header. The header section of the generator code for _NNClassifierMLP.gen.m provides the general information about the NNClassifierMLP element.

```

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

```

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

Code 2: NNClassifierMLP element prop update. The props_update section of the generator code for _NNClassifierMLP.gen.m updates the properties of the NNClassifierMLP 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 classifier.
5 %%% idefault!
6 'NNClassifierMLP'
7
8 %%% iprop!
9 DESCRIPTION (constant, string) is the description of the neural network
    multi-layer perceptron classifier.
10 %%% idefault!
11 'A neural network multi-layer perceptron classifier (NNClassifierMLP)
    comprises a multi-layer perceptron classifier model and a given dataset
    . NNClassifierMLP trains the multi-layer perceptron classifier 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 classifier.
15 %%% isettings!
16 'NNClassifierMLP'
17
18 %%% iprop!
19 ID (data, string) is a few-letter code for the neural network multi-layer
    perceptron classifier.

```

```

20 %%% idefault!
21 'NNClassifierMLP ID'
22
23 %%% iprop!
24 LABEL (metadata, string) is an extended label of the neural network multi-
    layer perceptron classifier.
25 %%% idefault!
26 'NNClassifierMLP label'
27
28 %%% iprop!
29 NOTES (metadata, string) are some specific notes about the neural network
    multi-layer perceptron classifier.
30 %%% idefault!
31 'NNClassifierMLP 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_CLA')
39
40 %%% iprop!
41 DP_CLASSES (parameter, classlist) is the list of compatible data points.
42 %%% idefault! ②
43 {'NNDatapoint_CON_CLA' 'NNDatapoint_CON_FUN_MP_CLA' 'NNDatapoint_Graph_CLA'
    'NNDatapoint_Measure_CLA'}
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(:)];

```

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

② defines the compatible NNDatapoint classes with this NNClassifierMLP.

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

```

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

```

④ is a query that constructs the one-hot vectors for the target classes.

⑤ trains the classifier with the defined dataset by the code under icalculate!.

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.

```

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

```

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

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

```

1 %% iprops!
2
3 %% iprop!
4 TARGET_CLASSES (query, stringlist) constructs the target classes which
   represent the class of each data point.
5 %%% icalculate! ①
6 % target_classes = nn.get('TARGET_CLASSES', D) returns a cell array with the
7 % target classes for all data points in dataset D.
8 if isempty(varargin)
9     value = {''};
10    return
11 end
12 d = varargin{1};
13 dp_dict = d.get('DP_DICT');
14 if dp_dict.get('LENGTH') == 0
15     value = {''};
16 else
17     nn_targets = [];
18     for i = 1:1:dp_dict.get('LENGTH')
19         target = dp_dict.get('IT', i).get('TARGET_CLASS');
20         nn_targets = [nn_targets; target];
21     end
22     value = nn_targets;
23 end
24
25 %%% iprop! ②
26 LAYERS (data, rvector) defines the number of layers and their neurons.
27 %%% idefault!
28 [32 32]
29 %%% igui!
30 pr = PanelPropRVectorSmart('EL', nn, 'PROP', NNClassifierMLP.LAYERS, ...
31     'MIN', 0, 'MAX', 2000, ...
32     'DEFAULT', NNClassifierMLP.getPropDefault('LAYERS'), ...
33     varargin{:});
34
35 %%% iprop!
36 WAITBAR (gui, logical) determines whether to show the waitbar.
37 %%% idefault!
38 true

```

① is a query that collects all the target class for all data points.

② defines the number of neuron per layer. For example, [32 32] represents two layers, each containing 32 neurons.

Code 4: **NNClassifierMLP element tests**. The tests section from the element generator `_NNClassifierMLP.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.

```

1 %% itests!
2
3 %%% itest!
4 %%% iname!
5 train the classifier with example data
6 %%% icode!
7
8 % ensure the example data is generated
9 if ~isfile([fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data NN
   CLA CON XLS' filesep 'atlas.xlsx'])
10     test_NNDataPoint_CON_CLA % create example files
11 end
12
13 % Load BrainAtlas
14 im_ba = ImporterBrainAtlasXLS( ...
15     'FILE', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data
   NN CLA CON XLS' filesep 'atlas.xlsx'], ...
16     'WAITBAR', true ...
17 );
18
19 ba = im_ba.get('BA');
20
21 % Load Groups of SubjectCON
22 im_gr1 = ImporterGroupSubjectCON_XLS( ...
23     'DIRECTORY', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example
   data NN CLA CON XLS' filesep 'CON_Group_1_XLS'], ...
24     'BA', ba, ...
25     'WAITBAR', true ...
26 );
27
28 gr1 = im_gr1.get('GR');
29
30 im_gr2 = ImporterGroupSubjectCON_XLS( ...
31     'DIRECTORY', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example
   data NN CLA CON XLS' filesep 'CON_Group_2_XLS'], ...
32     'BA', ba, ...
33     'WAITBAR', true ...
34 );
35
36 gr2 = im_gr2.get('GR');
37
38 % create item lists of NNDataPoint_CON_CLA
39 [~, group_folder_name] = fileparts(im_gr1.get('DIRECTORY'));
40 it_list1 = cellfun(@(x) NNDataPoint_CON_CLA( ...
41     'ID', x.get('ID'), ...
42     'SUB', x, ...
43     'TARGET_CLASS', {group_folder_name}), ...
44     gr1.get('SUB_DICT').get('IT_LIST'), ...
45     'UniformOutput', false);
46
47 [~, group_folder_name] = fileparts(im_gr2.get('DIRECTORY'));
48 it_list2 = cellfun(@(x) NNDataPoint_CON_CLA( ...
49     'ID', x.get('ID'), ...
50     'SUB', x, ...

```

```

51     'TARGET_CLASS', {group_folder_name}), ...
52     gr2.get('SUB_DICT').get('IT_LIST'), ...
53     'UniformOutput', false);
54
55 % create NNDataPoint_CON_CLA DICT items
56 dp_list1 = IndexedDictionary(...
57     'IT_CLASS', 'NNDataPoint_CON_CLA', ...
58     'IT_LIST', it_list1 ...
59     );
60
61 dp_list2 = IndexedDictionary(...
62     'IT_CLASS', 'NNDataPoint_CON_CLA', ...
63     'IT_LIST', it_list2 ...
64     );
65
66 % create a NNDataset containing the NNDataPoint_CON_CLA DICT
67 d1 = NNDataset( ...
68     'DP_CLASS', 'NNDataPoint_CON_CLA', ...
69     'DP_DICT', dp_list1 ...
70     );
71
72 d2 = NNDataset( ...
73     'DP_CLASS', 'NNDataPoint_CON_CLA', ...
74     'DP_DICT', dp_list2 ...
75     );
76
77 % combine the two datasets
78 d = NNDatasetCombine('D_LIST', {d1, d2}).get('D');
79
80 nn = NNClassifierMLP('D', d, 'LAYERS', [10 10 10]);
81 trained_model = nn.get('MODEL');
82
83 % Check whether the number of fully-connected layer matches (excluding
    Dense_output layer) ①
84 assert(length(nn.get('LAYERS')) == sum(contains({trained_model.Layers.Name},
    'Dense')) - 1, ...
85     [BRAPH2.STR 'NNClassifierMLP:' BRAPH2.FAIL_TEST], ...
86     'NNClassifierMLP does not construct the layers correctly. The number of
    the inputs should be the same as the length of dense layers the
    property.' ...
87 )

```

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