# *Implement a New Neural Network Classifier*

*The BRAPH 2 Developers*

*January 8, 2025*

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*

## *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.

```
%% iheader!
NNClassifierMLP < NNBase (nn, multi-layer perceptron classifier) comprises a
      multi-layer perceptron classifier model and a given dataset.  (1)

%%% idescription!
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.

%%% ibuild!
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.

```
%% iprops_update!

%%% iprop!
NAME (constant, string) is the name of the neural network multi-layer
     perceptron classifier.
%%%% idefault!
'NNClassifierMLP'

%%% iprop!
DESCRIPTION (constant, string) is the description of the neural network
     multi-layer perceptron classifier.
%%%% idefault!
'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.'

%%% iprop!
TEMPLATE (parameter, item) is the template of the neural network multi-layer
      perceptron classifier.
%%%% isettings!
'NNClassifierMLP'

%%% iprop!
ID (data, string) is a few-letter code for the neural network multi-layer
     perceptron classifier.
```

```
%%%% idefault!
'NNClassifierMLP ID'

%%% iprop!
LABEL (metadata, string) is an extended label of the neural network multi-
     layer perceptron classifier.
%%%% idefault!
'NNClassifierMLP label'

%%% iprop!
NOTES (metadata, string) are some specific notes about the neural network
     multi-layer perceptron classifier.
%%%% idefault!
'NNClassifierMLP notes'

%%% iprop! (1)
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.
%%%% isettings!
'NNDataset'
%%%% idefault!
NNDataset('DP_CLASS', 'NNDataPoint_CON_CLA')

%%% iprop!
DP_CLASSES (parameter, classlist) is the list of compatible data points.
%%%% idefault! (2)
{'NNDataPoint_CON_CLA' 'NNDataPoint_CON_FUN_MP_CLA' 'NNDataPoint_Graph_CLA'
     'NNDataPoint_Measure_CLA'}

%%% iprop!
INPUTS (query, cell) constructs the data in the CB (channel-batch) format.
%%%% icalculate! (3)
% inputs = nn.get('inputs', D) returns a cell array with the
%  inputs for all data points in dataset D.
if isempty(varargin)
    value = {};
    return
end
d = varargin{1};
inputs_group = d.get('INPUTS');
if isempty(inputs_group)
    value = {};
else
    flattened_inputs_group = [];
    for i = 1:1:length(inputs_group)
        inputs_individual = inputs_group{i};
        flattened_inputs_individual = [];
        while ~isempty(inputs_individual)
            currentData = inputs_individual{end};  % Get the last element
     from the stack
            inputs_individual = inputs_individual(1:end-1);   % Remove the
     last element

            if iscell(currentData)
                % If it's a cell array, add its contents to the stack
                inputs_individual = [inputs_individual currentData{:}];
            else
                % If it's numeric or other data, append it to the vector
                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 = 1:1:length(layers)
112         nn_architecture = [nn_architecture
113             fullyConnectedLayer(layers(i), 'Name', ['Dense_' num2str(i)])
114             batchNormalizationLayer('Name', ['BatchNormalization_' num2str(i
      )])
115             dropoutLayer('Name', ['Dropout_' num2str(i)])
116             ];
117     end
118     nn_architecture = [nn_architecture
119         reluLayer('Name', 'Relu_output')
120         fullyConnectedLayer(number_classes, 'Name', 'Dense_output')
121         softmaxLayer
122         classificationLayer('Name', 'Output')
123         ];
124
125     % specify trianing options  (9)
126     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.

```
        'MiniBatchSize', nn.get('BATCH'), ...
        'MaxEpochs', nn.get('EPOCHS'), ...
        'Shuffle', nn.get('SHUFFLE'), ...
        'Plots', nn.get('PLOT_TRAINING'), ...
        'Verbose', nn.get('VERBOSE'));

    % train the neural network (10)
    value = trainNetwork(inputs, targets, nn_architecture, options);
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`.

```
%% iprops!

%%% iprop!
TARGET_CLASSES (query, stringlist) constructs the target classes which
     represent the class of each data point.
%%%% icalculate! (1)
% target_classes = nn.get('TARGET_CLASSES', D) returns a cell array with the
%  target classes for all data points in dataset D.
if isempty(varargin)
    value = {''};
    return
end
d = varargin{1};
dp_dict = d.get('DP_DICT');
if dp_dict.get('LENGTH') == 0
    value = {''};
else
    nn_targets = [];
    for i = 1:1:dp_dict.get('LENGTH')
        target = dp_dict.get('IT', i).get('TARGET_CLASS');
        nn_targets = [nn_targets; target];
    end
    value = nn_targets;
end

%%% iprop! (2)
LAYERS (data, rvector) defines the number of layers and their neurons.
%%%% idefault!
[32 32]
%%%% igui!
pr = PanelPropRVectorSmart('EL', nn, 'PROP', NNClassifierMLP.LAYERS, ...
    'MIN', 0, 'MAX', 2000, ...
    'DEFAULT', NNClassifierMLP.getPropDefault('LAYERS'), ...
    varargin{:});

%%% iprop!
WAITBAR (gui, logical) detemines whether to show the waitbar.
%%%% idefault!
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.

```
%% itests!

%%% itest!
%%%% iname!
train the classifier with example data
%%%% icode!

% ensure the example data is generated
if ~isfile([fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example data NN
      CLA CON XLS' filesep 'atlas.xlsx'])
    create_data_NN_CLA_CON_XLS() % create example files
end

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

ba = im_ba.get('BA');

% Load Groups of SubjectCON
im_gr1 = ImporterGroupSubjectCON_XLS( ...
    'DIRECTORY', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example
     data NN CLA CON XLS' filesep 'CON_Group_1_XLS'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr1 = im_gr1.get('GR');

im_gr2 = ImporterGroupSubjectCON_XLS( ...
    'DIRECTORY', [fileparts(which('NNDataPoint_CON_CLA')) filesep 'Example
     data NN CLA CON XLS' filesep 'CON_Group_2_XLS'], ...
    'BA', ba, ...
    'WAITBAR', true ...
    );

gr2 = im_gr2.get('GR');

% create item lists of NNDataPoint_CON_CLA
[~, group_folder_name] = fileparts(im_gr1.get('DIRECTORY'));
it_list1 = cellfun(@(x) NNDataPoint_CON_CLA( ...
    'ID', x.get('ID'), ...
    'SUB', x, ...
    'TARGET_CLASS', {group_folder_name}), ...
    gr1.get('SUB_DICT').get('IT_LIST'), ...
    'UniformOutput', false);

[~, group_folder_name] = fileparts(im_gr2.get('DIRECTORY'));
it_list2 = cellfun(@(x) NNDataPoint_CON_CLA( ...
    'ID', x.get('ID'), ...
    '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.