OMLT Documentation

Release 0.0.0

The OMLT Developers

Feb 11, 2023

CONTENTS:

OMLT is a Python package for representing machine learning models (neural networks and gradient-boosted trees) within the Pyomo optimization environment. The package provides various optimization formulations for machine learning models (such as full-space, reduced-space, and MILP) as well as an interface to import sequential Keras and general ONNX models.

Further details can be found in our arXiv preprint, presentation, and YouTube playlist.

CHAPTER

ONE

INSTALLATION

OMLT requires Python >= 3.6. The most stable OMLT version can be installed using the PyPI package index. This will also install the required depencies. Simply run:

```
pip install omlt
```

If using the latest un-released version, install from the github repository and install locally

```
git clone https://github.com/cog-imperial/OMLT.git
cd OMLT
pip install .
```

1.1 Optional Requirements

OMLT can import sequential Keras models which requires a working installation of tensorflow:

```
pip install tensorflow
```

OMLT can also import neural network and gradient boosted tree models using ONNX. This requires installing the ONNX interface:

```
pip install onnx
```

CHAPTER

TWO

QUICK-START

The quick-start uses the same model shown on the OMLT README, but it provides a little more detail. The example imports a neural network trained in TensorFlow, formulates a Pyomo model, and seeks the neural network input that produces the desired output.

We begin by importing the necessary packages. These include *tensorflow* to import the neural network and *pyomo* to build the optimization problem. We also import the necessary objects from *omlt* to formulate the neural network in Pyomo.

```
import tensorflow
import pyomo.environ as pyo
from omlt import OmltBlock, OffsetScaling
from omlt.neuralnet import FullSpaceNNFormulation, NetworkDefinition
from omlt.io import load_keras_sequential
```

We first load a simple neural network from the tests directory that contains 1 input, 1 output, and 3 hidden nodes with sigmoid activation functions.

```
#load a Keras model
nn = tensorflow.keras.models.load_model('tests/models/keras_linear_131_sigmoid',␣
↪compile=False)
```

We next create a Pyomo model and attach an *OmltBlock* which will be used to formulate the neural network. An *OmltBlock* is a custom Pyomo block that we use to build machine learning model formulations. We also create Pyomo model variables to represent the input and output of the neural network.

```
#create a Pyomo model with an OMLT block
model = pyo.ConcreteModel()
model.nn = OmltBlock()

#the neural net contains one input and one output
model.input = pyo.Var()
model.output = pyo.Var()
```

OMLT supports the use of scaling and input bound information. This information informs how the Pyomo model applies scaling and unscaling to the neural network inputs and outputs. It also informs variable bounds on the inputs.

```
#apply simple offset scaling for the input and output
scale_x = (1, 0.5)       #(mean,stdev) of the input
scale_y = (-0.25, 0.125) #(mean,stdev) of the output
scaler = OffsetScaling(offset_inputs=[scale_x[0]],
                       factor_inputs=[scale_x[1]],
                       offset_outputs=[scale_y[0]],
```

(continues on next page)

(continued from previous page)

```
                    factor_outputs=[scale_y[1]])

#provide bounds on the input variable (e.g. from training)
scaled_input_bounds = {0:(0,5)}
```

We now create a *NetworkDefinition* using the *load_keras_sequential* function where we provide the scaler object and input bounds. Once we have a *NetworkDefinition*, we can pass it to various formulation objects which decide how to build the neural network within the *OmltBlock*. Here, we use the *FullSpaceNNFormulation*, but others are also possible (see *formulations*).

```
#load the keras model into a network definition
net = load_keras_sequential(nn,scaler,scaled_input_bounds)

#multiple formulations of a neural network are possible
#this uses the default NeuralNetworkFormulation object
formulation = FullSpaceNNFormulation(net)

#build the formulation on the OMLT block
model.nn.build_formulation(formulation)
```

We can query the input and output pyomo variables that the *build_formulation* method produces (as well as scaled input and output varialbes). We lastly create pyomo constraints that connect our input and output variables defined earlier to the neural network input and output variables on the *OmltBlock*.:

```
#query inputs and outputs, as well as scaled inputs and outputs
model.nn.inputs
model.nn.outputs
model.nn.scaled_inputs
model.nn.scaled_outputs

#connect pyomo model input and output to the neural network
@model.Constraint()
def connect_input(mdl):
    return mdl.input == mdl.nn.inputs[0]

@model.Constraint()
def connect_output(mdl):
    return mdl.output == mdl.nn.outputs[0]
```

Lastly, we formulate an objective function and use Ipopt to solve the optimization problem.:

```
#solve an inverse problem to find that input that most closely matches the output value␣
↪of 0.5
model.obj = pyo.Objective(expr=(model.output - 0.5)**2)
status = pyo.SolverFactory('ipopt').solve(model, tee=False)
print(pyo.value(model.input))
print(pyo.value(model.output))
```

CHAPTER

THREE

JUPYTER NOTEBOOKS

OMLT provides Jupyter notebooks to demonstrate its core capabilities. All notebooks can be found on the OMLT github page. The notebooks are summarized as follows:

- build_network.ipynb shows how to manually create a *NetworkDefinition* object. This notebook is helpful for understanding the details of the internal layer structure that OMLT uses to represent neural networks.
- import_network.ipynb shows how to import neural networks from Keras and PyTorch using ONNX interoperability. The notebook also shows how to import variable bounds from data.
- neural_network_formulations.ipynb showcases the different neural network formulations available in OMLT.
- mnist_example_dense.ipynb trains a fully dense neural network on MNIST and uses OMLT to find adversarial examples.
- mnist_example_convolutional.ipynb trains a convolutional neural network on MNIST and uses OMLT to find adversarial examples.
- auto-thermal-reformer.ipynb develops a neural network surrogate (using sigmoid activations) with data from a process model built using IDAES-PSE.
- auto-thermal-reformer-relu.ipynb develops a neural network surrogate (using ReLU activations) with data from a process model built using IDAES-PSE.
- bo_with_trees.ipynb incorporates gradient-boosted-trees into a Bayesian optimization loop to optimize the Rosenbrock function.

CHAPTER

FOUR

API DOCUMENTATION

4.1 OMLT Block

The omlt.block module contains the implementation of the OmltBlock class. This class is used in combination with a formulation object to construct the necessary constraints and variables to represent ML models.

Example

```
import tensorflow.keras as keras
from omlt import OmltBlock
from omlt.neuralnet import FullSpaceNNFormulation
from omlt.io import load_keras_sequential

nn = keras.models.load_model(keras_fname)
net = load_keras_sequential(nn)

m = pyo.ConcreteModel()
m.neural_net_block = OmltBlock()
m.neural_net_block.build_formulation(FullSpaceNNFormulation(net))

m.obj = pyo.Objective(expr=(m.neural_net_block.outputs[2]-4.0)**2)
status = pyo.SolverFactory('ipopt').solve(m, tee=True)
pyo.assert_optimal_termination(status)
```

class omlt.block.**OmltBlock**(**args*, ***kwds*)

Bases: CustomBlock

Note: *OmltBlock* is the name used to declare the custom Pyomo block which is exposed to the user. The block functionality is given by *OmltBlockData* which inherits from Pyomo *_BlockData*.

class omlt.block.**OmltBlockData**(*component*)

Bases: _BlockData

_setup_inputs_outputs(*, *input_indexes*, *output_indexes*)

This function should be called by the derived class to create the inputs and outputs on the block

Parameters

- **input_indexes** – list list of indexes (can be tuples) defining the set to be used for the input variables

- **output_indexes** – list list of indexes (can be tuples) defining the set to be used for the input variables

build_formulation(*formulation*)

Call this method to construct the constraints (and possibly intermediate variables) necessary for the particular neural network formulation. The formulation object can be accessed later through the "formulation" attribute.

Parameters

formulation (*instance of* _PyomoFormulation) – see, for example, FullSpaceNNFormulation

4.2 Scaling

The omlt.scaling module describes the interface for providing different scaling expressions to the Pyomo model for the inputs and outputs of an ML model. An implementation of a common scaling approach is included with *OffsetScaling*.

class omlt.scaling.**ScalingInterface**

Bases: ABC

abstract get_scaled_input_expressions(*input_vars*)

This method returns a list of expressions for the scaled inputs from the unscaled inputs

abstract get_unscaled_output_expressions(*scaled_output_vars*)

This method returns a list of expressions for the unscaled outputs from the scaled outputs

class omlt.scaling.**OffsetScaling**(*offset_inputs*, *factor_inputs*, *offset_outputs*, *factor_outputs*)

Bases: *ScalingInterface*

This scaling object represents the following scaling equations for inputs (x) and outputs (y)

$$x_i^{scaled} = \frac{(x_i - x_i^{offset})}{x_i^{factor}}$$

$$y_i^{scaled} = \frac{(y_i - y_i^{offs}}{y_i^{factor}}$$

Parameters

- **offset_inputs** (*array-like*) – Array of the values of the offsets for each input to the network
- **factor_inputs** (*array-like*) – Array of the scaling factors (division) for each input to the network
- **offset_outputs** (*array-like*) – Array of the values of the offsets for each output from the network
- **factor_outputs** (*array-like*) – Array of the scaling factors (division) for each output from the network

get_scaled_input_expressions(*input_vars*)

Get the scaled input expressions of the input variables.

get_scaled_output_expressions(*output_vars*)

Get the scaled output expressions of the output variables.

get_unscaled_input_expressions(*scaled_input_vars*)
Get the unscaled input expressions of the scaled input variables.

get_unscaled_output_expressions(*scaled_output_vars*)
Get the unscaled output expressions of the scaled output variables.

4.3 File IO

4.3.1 Input Bounds

omlt.io.input_bounds.**load_input_bounds**(*input_bounds_filename*)
Read the input bounds from the given file.

omlt.io.input_bounds.**write_input_bounds**(*input_bounds_filename*, *input_bounds*)
Write the specified input bounds to the given file.

4.3.2 Keras Reader

4.3.3 ONNX

omlt.io.onnx.**load_onnx_neural_network**(*onnx*, *scaling_object=None*, *input_bounds=None*)
Load a NetworkDefinition from an onnx object.

Parameters

- **onnx** – onnx model
- **scaling_object** (*instance of object supporting ScalingInterface*) –
- **input_bounds** (*list of tuples*) –

Return type
NetworkDefinition

omlt.io.onnx.**load_onnx_neural_network_with_bounds**(*filename*)
Load a NetworkDefinition with input bounds from an onnx object.

Parameters
filename (*str*) – the path where the ONNX model and input bounds file are written

Return type
NetworkDefinition

omlt.io.onnx.**write_onnx_model_with_bounds**(*filename*, *onnx_model=None*, *input_bounds=None*)
Write the ONNX model to the given file.

If *input_bounds* is not None, write it alongside the ONNX model.

Parameters

- **filename** (*str*) – the path where the ONNX model is written
- **onnx_model** (*onnx model or None*) – the onnx model
- **input_bounds** (*None or dict-like or list*) – bounds on the input variables

class omlt.io.onnx_parser.**NetworkParser**
Bases: object

References

- https://github.com/onnx/onnx/blob/master/docs/Operators.md

parse_network(*graph*, *scaling_object*, *input_bounds*)

4.4 Gradient Boosted Trees

We use the following notation to describe the gradient-boosted trees formulation:

$$
\begin{aligned}
\hat{\mu} &:= \text{Mean prediction of tree ensemble} \\
T &:= \text{Set of trees in ensemble} \\
L_t &:= \text{Set of leaves in tree } t \\
z_{t,l} &:= \text{Binary variable indicating if leaf } l \text{ in tree } t \text{ is active} \\
\text{Left}_{t,s} &:= \text{Set of leaf variables left of split } s \text{ in tree } t \\
\text{Right}_{t,s} &:= \text{Set of leaf variables right of split } s \text{ in tree } t \\
y_{i(s),j(s)} &:= \text{Binary variable indicating if split } s \text{ is active} \\
i(s) &:= \text{feature of split } s \\
j(s) &:= \text{index of split } s \\
V_t &:= \text{Set of splits in tree } t \\
n &:= \text{Index set of input features} \\
m_i &:= \text{Index set of splits for feature } i \\
F_{t,l} &:= \text{Weight of leaf } l \text{ in tree } t
\end{aligned}
$$

class omlt.gbt.gbt_formulation.**GBTBigMFormulation**(*gbt_model*)

Bases: *_PyomoFormulation*

This class is the entry-point to build gradient-boosted trees formulations.

This class iterates over all trees in the ensemble and generates constraints to enforce splitting rules according to:

References

- Misic, V. "Optimization of tree ensembles." Operations Research 68.5 (2020): 1605-1624.
- Mistry, M., et al. "Mixed-integer convex nonlinear optimization with gradient-boosted trees embedded." INFORMS Journal on Computing (2020).

Parameters

tree_ensemble_structure (GradientBoostedTreeModel) – the tree ensemble definition

property input_indexes

The indexes of the formulation inputs.

property output_indexes

The indexes of the formulation output.

omlt.gbt.gbt_formulation.**add_formulation_to_block**(*block*, *model_definition*, *input_vars*, *output_vars*)

Adds the gradient-boosted trees formulation to the given Pyomo block.

$$
\begin{aligned}
\hat{\mu} &= \sum_{t \in T} \sum_{l \in L_t} F_{t,l} z_{t,l}, & \\
\sum_{l \in L_t} z_{t,l} &= 1, & \forall t \in T, \\
\sum_{l \in \text{Left}_{t,s}} z_{t,l} &\leq y_{i(s),j(s)}, & \forall t \in T, \forall s \in V_t, \\
\sum_{l \in \text{Right}_{t,s}} z_{t,l} &\leq 1 - y_{i(s),j(s)}, & \forall t \in T, \forall s \in V_t, \\
y_{i,j} &\leq y_{i,j+1}, & \forall i \in [n], \forall j \in [m_i - 1], \\
x_i &\geq v_{i,0} + \sum_{j=1}^{m_i} (v_{i,j} - v_{i,j-1})(1 - y_{i,j}), & \forall i \in [n], \\
x_i &\leq v_{i,m_i+1} + \sum_{j=1}^{m_i} (v_{i,j} - v_{i,j+1}) y_{i,j}, & \forall i \in [n].
\end{aligned}
$$

References

- Misic, V. "Optimization of tree ensembles." Operations Research 68.5 (2020): 1605-1624.
- Mistry, M., et al. "Mixed-integer convex nonlinear optimization with gradient-boosted trees embedded." INFORMS Journal on Computing (2020).

Parameters

- **block** (*Block*) – the Pyomo block
- **tree_ensemble_structure** (GradientBoostedTreeModel) – the tree ensemble definition
- **input_vars** (*Var*) – the input variables of the Pyomo block
- **output_vars** (*Var*) – the output variables of the Pyomo block

class omlt.gbt.model.**GradientBoostedTreeModel**(*onnx_model*, *scaling_object=None*, *scaled_input_bounds=None*)

Bases: object

property n_inputs

Returns the number of input variables

property n_outputs

Returns the number of output variables

property onnx_model

Returns underlying onnx model of the tree model being used

property scaled_input_bounds

Return a list of tuples containing lower and upper bounds of tree ensemble inputs

property scaling_object

Return an instance of the scaling object that supports the ScalingInterface

4.5 Neural Networks

4.5.1 OMLT Layers

Neural network layer classes.

class omlt.neuralnet.layer.**ConvLayer2D**(*input_size*, *output_size*, *strides*, *kernel*, ***, *activation=None*, *input_index_mapper=None*)

Bases: *Layer2D*

Two-dimensional convolutional layer.

Parameters

- **input_size** (*tuple*) – the size of the input.
- **output_size** (*tuple*) – the size of the output..
- **strides** (*matrix-like*) – stride of the cross-correlation kernel.
- **kernel** (*matrix-like*) – the cross-correlation kernel.
- **activation** (*str or None*) – activation function name
- **input_index_mapper** (*IndexMapper or None*) – map indexes from this layer index to the input layer index size

property kernel

Return the cross-correlation kernel

property kernel_depth

Return the depth of the cross-correlation kernel

property kernel_shape

Return the shape of the cross-correlation kernel

kernel_with_input_indexes(*out_d*, *out_r*, *out_c*)

Returns an iterator over the kernel value and input index for the output at index *(out_d, out_r, out_c)*.

Parameters

- **out_d** (*int*) – the output depth.
- **out_r** (*int*) – the output row.
- **out_c** (*int*) – the output column.

class omlt.neuralnet.layer.**DenseLayer**(*input_size*, *output_size*, *weights*, *biases*, ***, *activation=None*, *input_index_mapper=None*)

Bases: *Layer*

Dense layer implementing *output = activation(dot(input, weights) + biases)*.

Parameters

- **input_size** (*tuple*) – the size of the input.
- **output_size** (*tuple*) – the size of the output.

- **weight** (*matrix-like*) – the weight matrix.
- **biases** (*array-like*) – the biases.
- **activation** (*str or None*) – activation function name
- **input_index_mapper** (IndexMapper *or None*) – map indexes from this layer index to the input layer index size

property biases

Return the vector of node biases

property weights

Return the matrix of node weights

class omlt.neuralnet.layer.**IndexMapper**(*input_size*, *output_size*)

Bases: object

Map indexes from one layer to the other.

Parameters

- **input_size** (*tuple*) – the input size
- **output_size** (*tuple*) – the mapped input layer's output size

property input_size

Return the size of the input tensor

property output_size

Return the size of the output tensor

class omlt.neuralnet.layer.**InputLayer**(*size*)

Bases: *Layer*

The first layer in any network.

Parameters

size (*tuple*) – the size of the input.

class omlt.neuralnet.layer.**Layer**(*input_size*, *output_size*, *, *activation=None*, *input_index_mapper=None*)

Bases: object

Base layer class.

Parameters

- **input_size** (*tuple*) – size of the layer input
- **output_size** (*tuple*) – size of the layer output
- **activation** (*str or None*) – activation function name
- **input_index_mapper** (IndexMapper *or None*) – map indexes from this layer index to the input layer index size

property activation

Return the activation function

eval_single_layer(*x*)

Evaluate the layer at x.

Parameters

x (*array-like*) – the input tensor. Must have size *self.input_size*.

property input_index_mapper

Return the index mapper

property input_indexes

Return a list of the input indexes

property input_indexes_with_input_layer_indexes

Return an iterator generating a tuple of local and input indexes.

Local indexes are indexes over the elements of the current layer. Input indexes are indexes over the elements of the previous layer.

property input_size

Return the size of the input tensor

property output_indexes

Return a list of the output indexes

property output_size

Return the size of the output tensor

class omlt.neuralnet.layer.**Layer2D**(*input_size*, *output_size*, *strides*, *, *activation=None*, *input_index_mapper=None*)

Bases: *Layer*

Abstract two-dimensional layer that downsamples values in a kernel to a single value.

Parameters

- **input_size** (*tuple*) – the size of the input.
- **output_size** (*tuple*) – the size of the output.
- **strides** (*matrix-like*) – stride of the kernel.
- **activation** (*str or None*) – activation function name
- **input_index_mapper** (*IndexMapper or None*) – map indexes from this layer index to the input layer index size

get_input_index(*out_index*, *kernel_index*)

Returns the input index corresponding to the output at *out_index* and the kernel index *kernel_index*.

property kernel_depth

Return the depth of the kernel

kernel_index_with_input_indexes(*out_d*, *out_r*, *out_c*)

Returns an iterator over the index within the kernel and input index for the output at index *(out_d, out_r, out_c)*.

Parameters

- **out_d** (*int*) – the output depth.
- **out_r** (*int*) – the output row.
- **out_c** (*int*) – the output column.

property kernel_shape

Return the shape of the kernel

property strides

Return the stride of the layer

class omlt.neuralnet.layer.**PoolingLayer2D**(*input_size*, *output_size*, *strides*, *pool_func_name*, *kernel_shape*, *kernel_depth*, *, *activation=None*, *input_index_mapper=None*)

Bases: *Layer2D*

Two-dimensional pooling layer.

Parameters

- **input_size** (*tuple*) – the size of the input.
- **output_size** (*tuple*) – the size of the output.
- **strides** (*matrix-like*) – stride of the kernel.
- **pool_func** (*str*) – name of function used to pool values in a kernel to a single value.
- **transpose** (*bool*) – True iff input matrix is accepted in transposed (i.e. column-major) form.
- **activation** (*str or None*) – activation function name
- **input_index_mapper** (IndexMapper *or None*) – map indexes from this layer index to the input layer index size

property kernel_depth

Return the depth of the kernel

property kernel_shape

Return the shape of the kernel

4.5.2 Neural Network Definition

class omlt.neuralnet.network_definition.**NetworkDefinition**(*scaling_object=None*, *scaled_input_bounds=None*, *unscaled_input_bounds=None*)

Bases: object

add_edge(*from_layer*, *to_layer*)

Add an edge between two layers.

Parameters

- **from_layer** (Layer) – the layer with the outbound connection
- **to_layer** (Layer) – the layer with the inbound connection

add_layer(*layer*)

Add a layer to the network.

Parameters

layer (Layer) – the layer to add to the network

property input_layers

Return an iterator over the input layers

property input_nodes

An alias for input_layers

layer(*layer_id*)

Return the layer with the given id

property layers

Return an iterator over all the layers

property output_layers

Return an iterator over the output layer

property output_nodes

An alias for output_layers

predecessors(*layer*)

Return an iterator over the layers with outbound connections into the layer

property scaled_input_bounds

Return a dict of tuples containing lower and upper bounds of neural network inputs

property scaling_object

Return an instance of the scaling object that supports the ScalingInterface

successors(*layer*)

Return an iterator over the layers with an inbound connection from the layer

property unscaled_input_bounds

Return a dict of tuples containing lower and upper bounds of unscaled neural network inputs

4.5.3 Neural Network Formulations

Base Formulation

class omlt.formulation.**_PyomoFormulation**

Bases: _PyomoFormulationInterface

This is a base class for different Pyomo formulations. To create a new formulation, inherit from this class and implement the abstract methods and properties.

property block

The underlying block containing the constraints / variables for this formulation.

abstract property input_indexes

Return the indices corresponding to the inputs of the ML model. This is a list of entries (which may be tuples for higher dimensional inputs).

abstract property output_indexes

Return the indices corresponding to the outputs of the ML model. This is a list of entries (which may be tuples for higher dimensional outputs).

Provided Formulations

class omlt.neuralnet.nn_formulation.**FullSpaceNNFormulation**(*network_structure*, *layer_constraints=None*, *activation_constraints=None*)

Bases: *_PyomoFormulation*

This class is the entry-point to build neural network formulations.

This class iterates over all nodes in the neural network and for each one them, generates the constraints to represent the layer and its activation function.

Parameters

- **network_structure** (NetworkDefinition) – the neural network definition
- **layer_constraints** (*dict-like or None*) – overrides the constraints generated for the specified layer types
- **activation_constraints** (*dict-like or None*) – overrides the constraints generated for the specified activation functions

property block

The underlying block containing the constraints / variables for this formulation.

property input_indexes

The indexes of the formulation inputs.

property output_indexes

The indexes of the formulation output.

class omlt.neuralnet.nn_formulation.**ReducedSpaceNNFormulation**(*network_structure*, *activation_functions=None*)

Bases: *_PyomoFormulation*

This class is used to build reduced-space formulations of neural networks.

Parameters

- **network_structure** (NetworkDefinition) – the neural network definition
- **activation_functions** (*dict-like or None*) – overrides the actual functions used for particular activations

property block

The underlying block containing the constraints / variables for this formulation.

property input_indexes

The indexes of the formulation inputs.

property output_indexes

The indexes of the formulation output.

class omlt.neuralnet.nn_formulation.**FullSpaceSmoothNNFormulation**(*network_structure*)

Bases: *FullSpaceNNFormulation*

property block

The underlying block containing the constraints / variables for this formulation.

property input_indexes

The indexes of the formulation inputs.

property output_indexes

The indexes of the formulation output.

class omlt.neuralnet.nn_formulation.**ReducedSpaceSmoothNNFormulation**(*network_structure*)

Bases: *ReducedSpaceNNFormulation*

This class is used to build reduced-space formulations of neural networks with smooth activation functions.

Parameters
network_structure (NetworkDefinition) – the neural network definition

property block

The underlying block containing the constraints / variables for this formulation.

property input_indexes

The indexes of the formulation inputs.

property output_indexes

The indexes of the formulation output.

class omlt.neuralnet.nn_formulation.**ReluBigMFormulation**(*network_structure*)

Bases: *FullSpaceNNFormulation*

property block

The underlying block containing the constraints / variables for this formulation.

property input_indexes

The indexes of the formulation inputs.

property output_indexes

The indexes of the formulation output.

class omlt.neuralnet.nn_formulation.**ReluComplementarityFormulation**(*network_structure*)

Bases: *FullSpaceNNFormulation*

property block

The underlying block containing the constraints / variables for this formulation.

property input_indexes

The indexes of the formulation inputs.

property output_indexes

The indexes of the formulation output.

class omlt.neuralnet.nn_formulation.**ReluPartitionFormulation**(*network_structure*, *split_func=None*)

Bases: *_PyomoFormulation*

This class is used to build partition-based formulations of neural networks.

Parameters

- **network_structure** (NetworkDefinition) – the neural network definition
- **split_func** (*callable*) – the function used to compute the splits

property block

The underlying block containing the constraints / variables for this formulation.

property input_indexes

The indexes of the formulation inputs.

property output_indexes

The indexes of the formulation output.

4.5.4 Layer and Activation Functions

We use the following notation to describe layer and activation functions:

$$
\begin{aligned}
N &:= \text{Set of nodes (i.e. neurons in the neural network)} \\
M_i &:= \text{Number of inputs to node } i \\
\hat{z}_i &:= \text{pre-activation value on node } i \\
z_i &:= \text{post-activation value on node } i \\
w_{ij} &:= \text{weight from input } j \text{ to node } i \\
b_i &:= \text{bias value for node } i
\end{aligned}
$$

Layer Functions

omlt.neuralnet.layers.full_space.**full_space_conv2d_layer**(*net_block*, *net*, *layer_block*, *layer*)

Add full-space formulation of the 2-D convolutional layer to the block

A 2-D convolutional layer applies cross-correlation kernels to a 2-D input. Specifically, the input is convolved by sliding the kernels along the input vertically and horizontally. At each location, the preactivation is computed as the dot product of the kernel weights and the input plus a bias term.

omlt.neuralnet.layers.full_space.**full_space_dense_layer**(*net_block*, *net*, *layer_block*, *layer*)

Add full-space formulation of the dense layer to the block

$$\hat{z}_i = \sum_{j=1}^{M_i} w_{ij} z_j + b_i \qquad \forall i \in N$$

omlt.neuralnet.layers.full_space.**full_space_maxpool2d_layer**(*net_block*, *net*, *layer_block*, *layer*)

Add Big-M max pooling formulation.

$$
\begin{aligned}
\hat{z}_i &\leq w \cdot x_i^l + \sum_{k=1}^{d} M_i^{l,k} q_i^k && \forall i \in N,\ \forall l \in \{1, ..., d\} \\
\hat{z}_i &\geq w \cdot x_i^l && \forall i \in N,\ \forall l \in \{1, ..., d\} \\
(x_i, \hat{z}_i, q_i) &\in [L_i, U_i] \times \mathbb{R} \times \Delta^d && \forall i \in N \\
q_i &\in \{0, 1\}^d && \forall i \in N \\
M_i^{l,k} &= w \cdot max\{L_i^l - L_i^k, \\
& L_i^l - U_i^k, U_i^l - L_i^k, U_i^l - U_i^k\} && \forall i \in N,\ \forall l \in \{1, ..., d\},\ \forall k \in \{1, ..., d\}
\end{aligned}
$$

where w is the convolution kernel on the preceding convolutional layer; d is the number of features in each of the N max pooling windows; x_i is the set of d features in the i-th max pooling window; Δ^d is the d-dimensional simplex; and [L_{i},U_{i}] are the bounds on x_{i}.

NOTE This formulation is adapted from the Anderson et al. (2020) formulation, section 5.1, with the following changes:

- OMLT presently does not support biases on convolutional layers. Bias terms from the original formulation are removed.
- The original formulation formulates the max of $w^l \cdot x + b^l$, varying the weights w and biases b and keeping the input x constant. Since convolutional layers have constant weights and biases convolved with varying portions of the feature map, this formulation formulates the max of $w \cdot x^l + b$.
- Due to the above 2 changes, the calculation of $N^{l,k}$ is changed.

omlt.neuralnet.layers.reduced_space.**reduced_space_dense_layer**(*net_block*, *net*, *layer_block*, *layer*, *activation*)

Add reduced-space formulation of the dense layer to the block

$$\hat{z}_i = \sum_{j=1}^{M_i} w_{ij} z_j + b_i \qquad \forall i \in N$$

omlt.neuralnet.layers.partition_based.**default_partition_split_func**(*w*, *n*)

Default function for partitioning weights in w into n partitions.

Weights in w are sorted and partitioned evenly.

omlt.neuralnet.layers.partition_based.**partition_based_dense_relu_layer**(*net_block*, *net*, *layer_block*, *layer*, *split_func*)

Partition-based ReLU activation formulation.

Generates the constraints for the ReLU activation function.

$$z_i = \max(0, \hat{z}_i) \qquad \forall i \in N$$

The partition-based formulation for the i-th node is given by:

$$\begin{aligned}
&\sum_n \big(\sum_{j \in \mathbb{S}_n} w_{ij} x_j - p_n \big) + \sigma b \leq 0 \\
&\sum_n p_n + (1 - \sigma) b \geq 0 \\
&z_i = \sum_n p_n + (1 - \sigma) b \\
&\sigma l_n \leq \sum_{j \in \mathbb{S}_n} w_{ij} x_j - p_n \leq \sigma u_n \\
&(1 - \sigma) l_n \leq p_n \leq (1 - \sigma) u_n \\
&\sigma \in \{0, 1\}
\end{aligned}$$

where l_n and u_n are, respectively, lower and upper bounds the n-th partition.

Activation Functions

omlt.neuralnet.activations.linear.**linear_activation_constraint**(*net_block*, *net*, *layer_block*, *layer*, *add_constraint=True*)

Linear activation constraint generator

Generates the constraints for the linear activation function.

$$z_i = \hat{z}_i \qquad \forall i \in N$$

omlt.neuralnet.activations.linear.**linear_activation_function**(*zhat*)

class omlt.neuralnet.activations.relu.**ComplementarityReLUActivation**(*transform=None*)

Bases: object

Complementarity-based ReLU activation forumlation.

Generates the constraints for the ReLU activation function.

$$z_i = \max(0, \hat{z}_i) \qquad \forall i \in N$$

The complementarity-based formulation for the i-th node is given by:

$$0 \leq z_i \perp (z - \hat{z}_i) \geq 0$$

omlt.neuralnet.activations.relu.**bigm_relu_activation_constraint**(*net_block*, *net*, *layer_block*, *layer*)

Big-M ReLU activation formulation.

Generates the constraints for the ReLU activation function.

$$z_i = \max(0, \hat{z}_i) \qquad \forall i \in N$$

The Big-M formulation for the i-th node is given by:

$$\begin{aligned} & z_i \geq \hat{z}_i \\ & z_i \leq \hat{z}_i - l(1-\sigma) \\ & z_i \leq u(\sigma) \\ & \sigma \in \{0, 1\} \end{aligned}$$

where l and u are, respectively, lower and upper bounds of $\hat{z}_i$.

omlt.neuralnet.activations.smooth.**sigmoid_activation_constraint**(*net_block*, *net*, *layer_block*, *layer*)

Sigmoid activation constraint generator

Generates the constraints for the sigmoid activation function.

$$z_i = \frac{1}{1 + exp(-\hat{z}_i)} \qquad \forall i \in N$$

omlt.neuralnet.activations.smooth.**sigmoid_activation_function**(*x*)

Applies the sigmoid function.

$$y = \frac{1}{1 + exp(-\hat{x})}$$

omlt.neuralnet.activations.smooth.**smooth_monotonic_activation_constraint**(*net_block*, *net*, *layer_block*, *layer*, *fcn*)

Activation constraint generator for a smooth monotonic function

Generates the constraints for the activation function fcn if it is smooth and monotonic

$$z_i = fcn(\hat{z}_i) \qquad \forall i \in N$$

omlt.neuralnet.activations.smooth.**softplus_activation_constraint**(*net_block*, *net*, *layer_block*, *layer*)

Softplus activation constraint generator

Generates the constraints for the softplus activation function.

$$z_i = log(exp(\hat{z}_i) + 1) \qquad \forall i \in N$$

omlt.neuralnet.activations.smooth.**softplus_activation_function**(*x*)

Applies the softplus function.

$$y = log(exp(\hat{x}) + 1)$$

omlt.neuralnet.activations.smooth.**tanh_activation_constraint**(*net_block*, *net*, *layer_block*, *layer*)

tanh activation constraint generator

Generates the constraints for the tanh activation function.

$$z_i = tanh(\hat{z}_i) \qquad \forall i \in N$$

omlt.neuralnet.activations.smooth.**tanh_activation_function**(*x*)

Applies the tanh function.

$$y = tanh(x)$$

CHAPTER

FIVE

LICENSE

5.1 Copyright Notice

Copyright 2021 National Technology & Engineering Solutions of Sandia, LLC (NTESS). Under the terms of Contract DE-NA0003525 with NTESS, the U.S. Government retains certain rights in this software.

Copyright (c) 2021, CG - Imperial College London All rights reserved.

Copyright (c) 2021, Carnegie Mellon University (Author: Carl Laird) All rights reserved.

5.1.1 Revised BSD License

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Sandia National Laboratories, nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS “AS IS” AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

PYTHON MODULE INDEX

O

INDEX

Symbols

A

B

C

D

E

F

G

I

K

L

M

N

O

P

R

S

T

U

W