Imperial College London

Carnegie Mellon University

Sandia National Laboratories

OMLT: Optimization & Machine Learning Toolkit

Francesco Ceccon*, Jordan Jalving*, Joshua Haddad, Alexander Thebelt, Calvin Tsay, Carl D Laird†, Ruth Misener†

*These authors contributed equally, †These authors contributed equally

Funding EPSRC EP/P016871/1 & EP/T001577/1
Sandia LDRD program
Institute for the Design of Advanced Energy Systems

Monday 21st March, 2022

Paper Ceccon*, Jalving*, Haddad, Thebelt, Tsay, Laird†, Misener†, *arXiv*, 2022.

Team members https://github.com/cog-imperial/OMLT

Francesco Ceccon
Imperial

Jordan Jalving
Sandia

Joshua Haddad
Sandia

Alexander Thebelt
Imperial

Calvin Tsay
Imperial

Carl D Laird
CMU

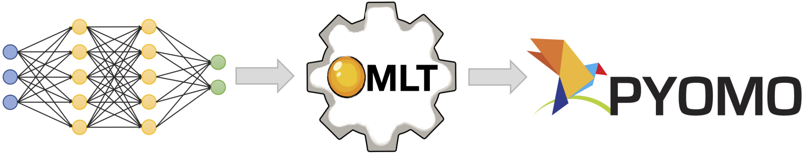
Ruth Misener
Imperial

You?
Join us on GitHub!

OMLT: Optimization & Machine Learning Toolkit

`https://github.com/cog-imperial/OMLT`

CI passing | codecov 94% | docs passing



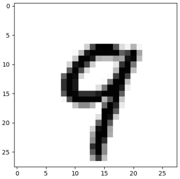
Why represent trained machine learning models as Pyomo [Bynum et al., 2021] formulations?

- **Adversarial examples** Verification [Lomuscio and Maganti, 2017], optimal adversary [Anderson et al., 2020], minimally-distorted adversary [Croce and Hein, 2020], lossless compression [Serra et al., 2020]
- **Machine learning** Maximize a neural acquisition function [Volpp et al., 2019], Bayesian optimization [Thebelt et al., 2021]
- **Engineering** Machine learning models may replace complicated constraints or serve as surrogates in larger design & operations problems.

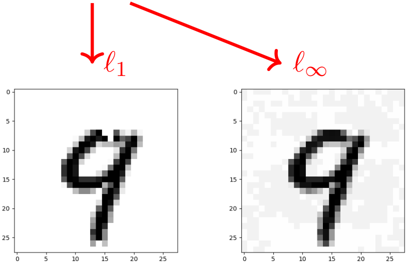
Optimization challenges to analyze trained neural networks

Example: Classification of MNIST digits

[Tsay et al., 2021]



Given	
Trained NN	
Image	$\bar{\boldsymbol{x}}$
Label	$j = 9$
Adversary?	$k = 4$



$\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_1 = 4$ $\qquad$ $\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_\infty = 0.05$

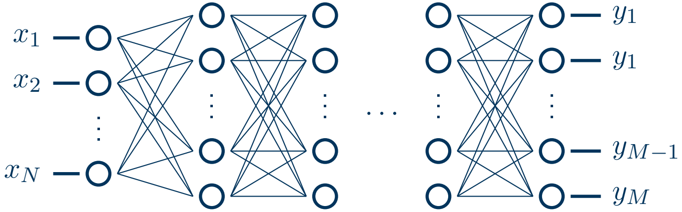
- **Verification** *[Feasibility]* Is there an adversary labeled k within a given perturbation (e.g., by ℓ_1- or ℓ_∞-norm)?
- **Optimal adversary** [Anderson et al., 2020] What image within a perturbation radius maximizes the prediction difference?
- **Minimally distorted adversary** [Croce and Hein, 2020] Smallest perturbation over which the NN can predict adversarial label k?
- **Lossless compression** [Serra et al., 2020] Can I safely remove NN nodes or layers?

What type of optimization problem do we want to solve?

Hybridize mechanistic, model-based optimization with surrogate models learned from data

$$\min_{\boldsymbol{x},\boldsymbol{y}} \quad f_0(\boldsymbol{x},\boldsymbol{y})$$

$$f_i(\boldsymbol{x},\boldsymbol{y}) \leq 0 \quad \forall i \in \{1,2,\ldots,C\}$$

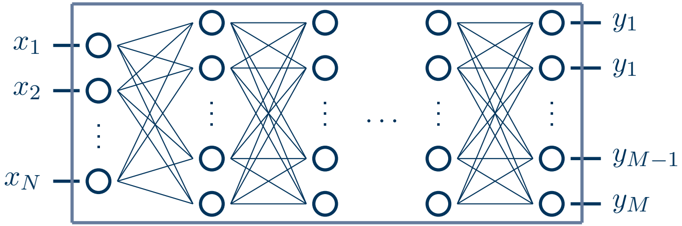


What type of optimization problem do we want to solve?

Hybridize mechanistic, model-based optimization with surrogate models learned from data

$$\min_{\boldsymbol{x},\boldsymbol{y}} \quad f_0(\boldsymbol{x},\boldsymbol{y})$$

$$f_i(\boldsymbol{x},\boldsymbol{y}) \le 0 \quad \forall i \in \{1,2,\ldots,C\}$$



What type of optimization problem do we want to solve?

Hybridize mechanistic, model-based optimization with surrogate models learned from data

$$\min_{\boldsymbol{x},\boldsymbol{y}} \quad f_0(\boldsymbol{x},\boldsymbol{y})$$
$$f_i(\boldsymbol{x},\boldsymbol{y}) \leq 0 \quad \forall i \in \{1,2,\ldots,C\}$$

What are the expressions f_i?

Affine • Have discrete variables (may want ReLU NN or GBT surrogates) • Nonlinear (may want smooth NN activations)

In OMLT v 1.0, what can the `OmltBlock` abstraction encapsulate?

Dense neural networks • Convolutional neural networks (CNN) • Gradient boosted trees (GBT)

Interface with ONNX for interoperability

PyTorch

scikit learn

TensorFlow

Keras

ONNX

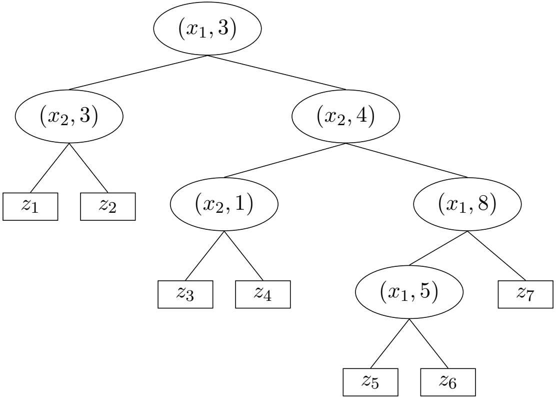
OMLT

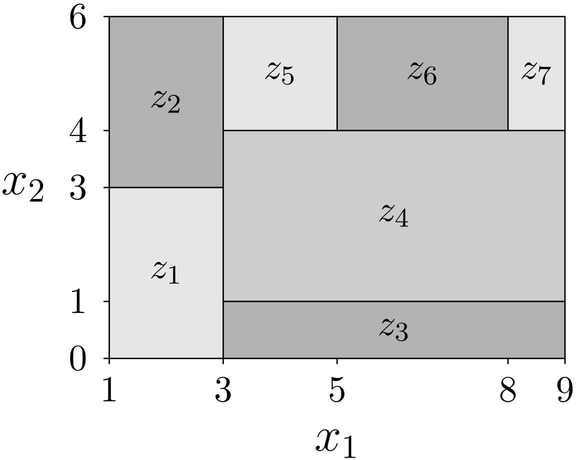
OMLT 1.0 Keras interface supports NN only (not GBT)

Want to join us?
OMLT 1.0 needs another GBT interface, e.g., LightGBM. ONNX doesn't support categorical variables

Most software *evaluates* a trained model . . . [Mistry et al., 2021]

Consider gradient boosted trees (GBTs). *Evaluating* a single tree is easy!

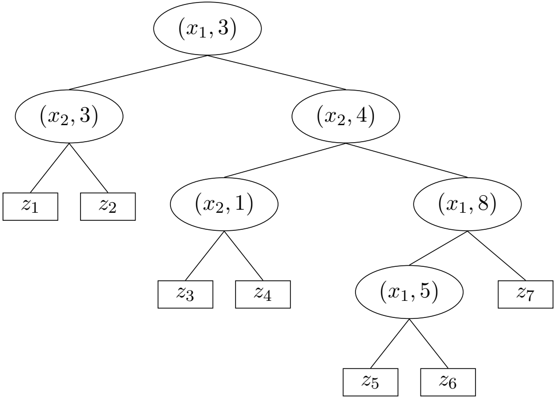


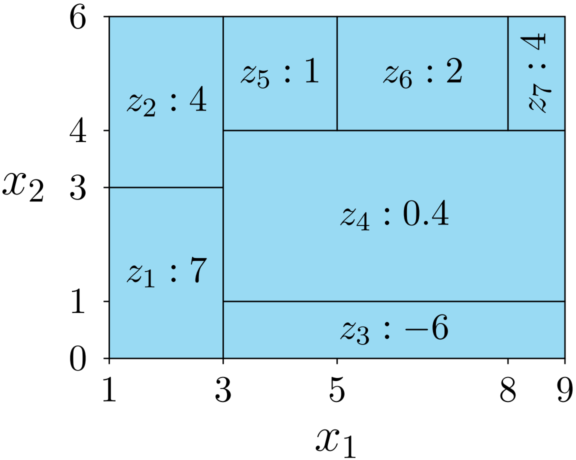


Most software *evaluates* a trained model . . . [Mistry et al., 2021]

Consider gradient boosted trees (GBTs). *Evaluating* a single tree is easy!

- Calculate for $\boldsymbol{x} = (4, 2)$

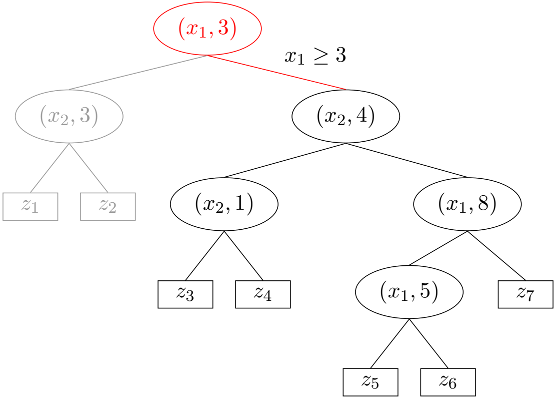


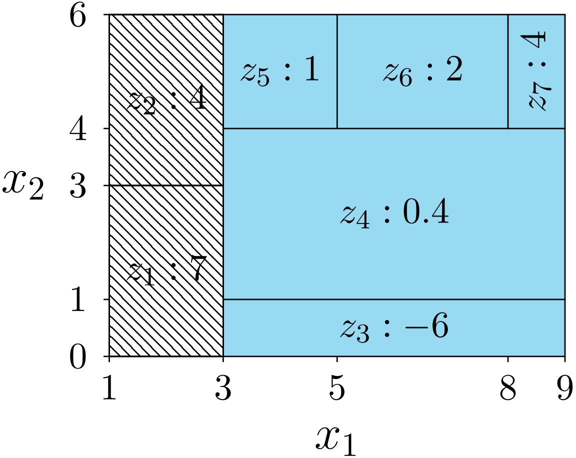


Most software *evaluates* a trained model . . . [Mistry et al., 2021]

Consider gradient boosted trees (GBTs). *Evaluating* a single tree is easy!

- Calculate for $\boldsymbol{x} = (4, 2)$

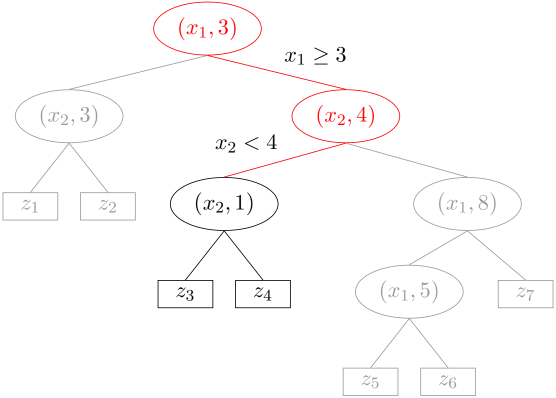


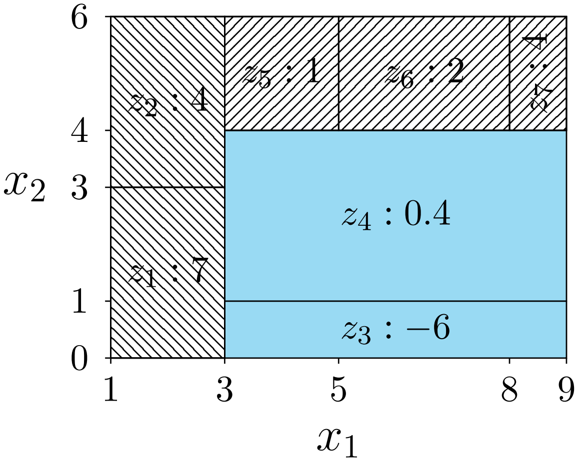


Most software *evaluates* a trained model ...

Consider gradient boosted trees (GBTs). *Evaluating* a single tree is easy!

- Calculate for $\boldsymbol{x} = (4, 2)$

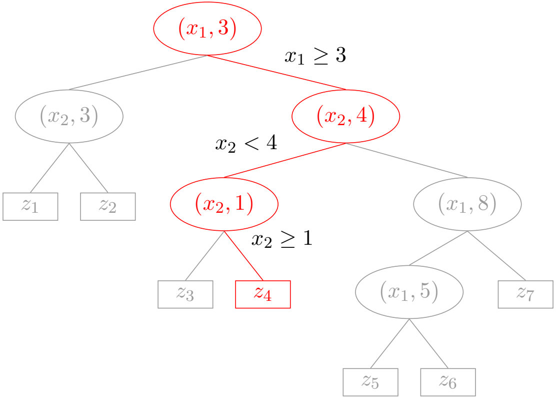


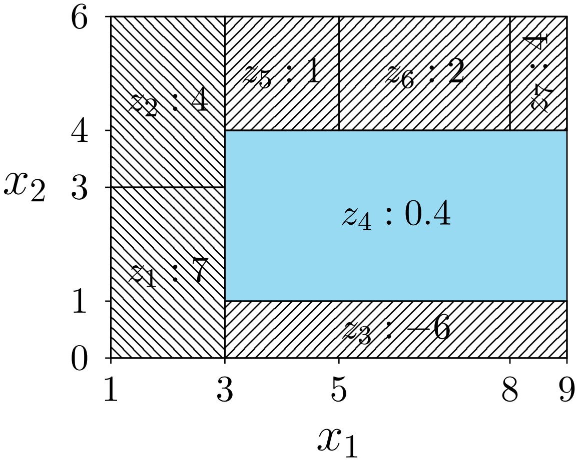


Most software *evaluates* a trained model . . . [Mistry et al., 2021]

Consider gradient boosted trees (GBTs). *Evaluating* a single tree is easy!

- Calculate for $\boldsymbol{x} = (4, 2)$
  - $t(\boldsymbol{x}) = 0.4$

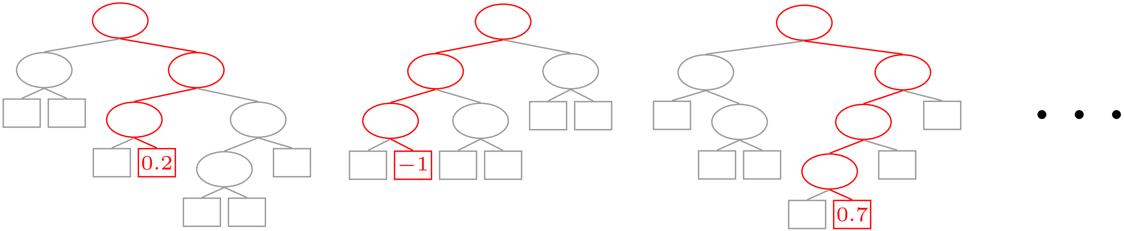




Most software *evaluates* a trained model . . . [Mistry et al., 2021]

Continuing the gradient boosted tree (GBT) example, it's also straightforward to *evaluate* the entire ensemble!

- GBT function defined by a set of trees, $\mathcal{T}$, and a constant, C
- To evaluate at $\boldsymbol{x} \in [\boldsymbol{L}, \boldsymbol{U}] \subset \mathbb{R}^n \longrightarrow y = \mathrm{GBT}(\boldsymbol{x}) = C + \sum_{t \in \mathcal{T}} t(\boldsymbol{x})$

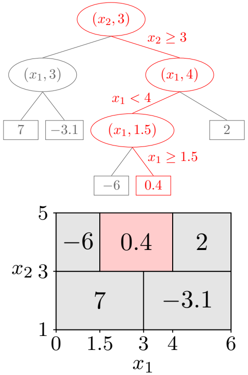


What if $\mathrm{GBT}(\boldsymbol{x})$ is in an optimization problem where $\boldsymbol{x}$ are decision variables, i.e., not fixed?

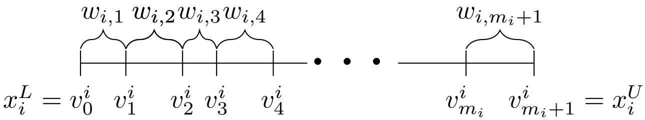
There are mathematical programming formulations incorporating function $y = \mathrm{GBT}(\boldsymbol{x})$ with decision variables $\boldsymbol{x}$ into a larger optimization problem [Mišić, 2020, Mistry et al., 2021, Thebelt et al., 2021]. OMLT automates the translation from ML model to optimization model!

Here's what OMLT allows users to ignore . . .

The `OMLTBlock` abstraction holds optimization formulations, e.g., by Mišić [2020] and Mistry et al. [2021]

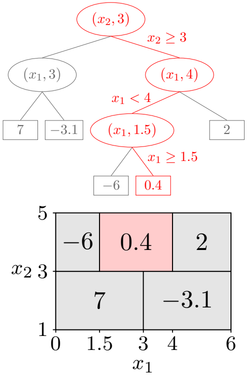


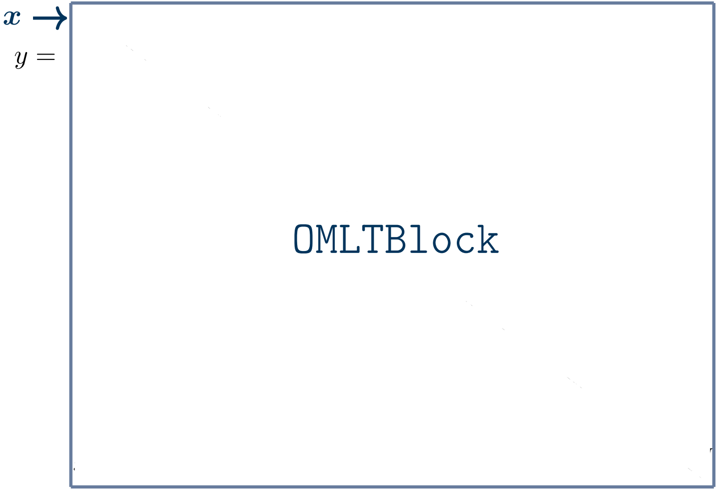
$$
\begin{aligned}
y = \quad & \mathrm{GBT}(\boldsymbol{x}) = \sum_{t \in \mathcal{T}} \sum_{l \in \mathcal{L}_t} F_{t,l} z_{t,l} && \\
& \sum_{l \in \mathcal{L}_t} z_{t,l} = 1, && \forall t \in \mathcal{T}, \\
& \sum_{l \in \mathrm{Left}_{t,s}} z_{t,l} \leq \quad w_{i(s),j(s)}, && \forall t \in \mathcal{T},\ s \in \mathcal{V}_t, \\
& \sum_{l \in \mathrm{Right}_{t,s}} z_{t,l} \leq 1 - w_{i(s),j(s)}, && \forall t \in \mathcal{T},\ s \in \mathcal{V}_t, \\
& w_{i,j} \leq w_{i,j+1}, && \forall i \in [n], j \in [m_i - 1], \\
& w_{i,j} \in \{0, 1\}, && \forall i \in [n],\ j \in [m_i], \\
& z_{t,l} \geq 0, && \forall t \in \mathcal{T},\ l \in \mathcal{L}_t,
\end{aligned}
$$



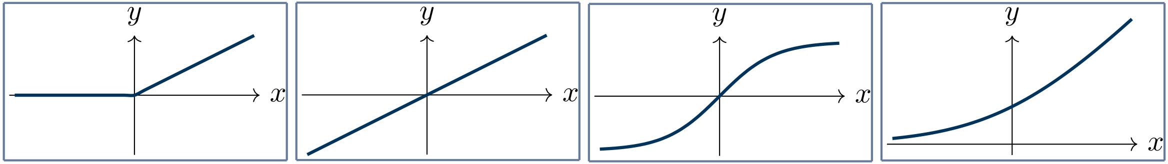
Here's what OMLT allows users to ignore . . .

The `OMLTBlock` abstraction holds optimization formulations, e.g., by Mišić [2020] and Mistry et al. [2021]





How NN activation functions map onto OMLT formulations . . .



ReLU

linear

tanh

softplus

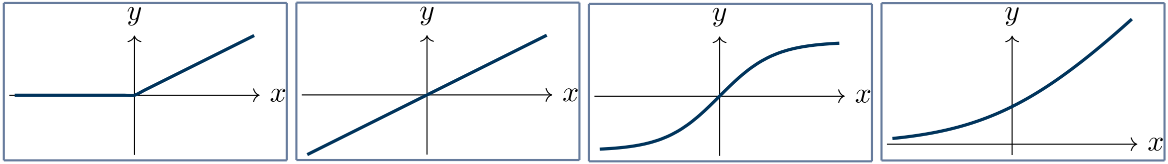
`ReluBigMFormulation`
`ReluComplementarityFormulation`
`ReluPartitionFormulation`

`FullSpaceSmoothNNFormulation`
`ReducedSpaceSmoothNNFormulation`

Formulations of activations [Schweidtmann and Mitsos, 2019, Anderson et al., 2020, Tsay et al., 2021]

Non-smooth [`ReluBigMFormulation`, `ReluComplementarityFormulation`, `Relu PartitionFormulation`] ReLU • **Smooth** [`{Full, Reduced}SpaceSmoothNNFormulation`] Linear • Tanh • Sigmoid • Softplus • Smooth monotonic

How NN activation functions map onto OMLT formulations . . .



ReLU linear tanh softplus

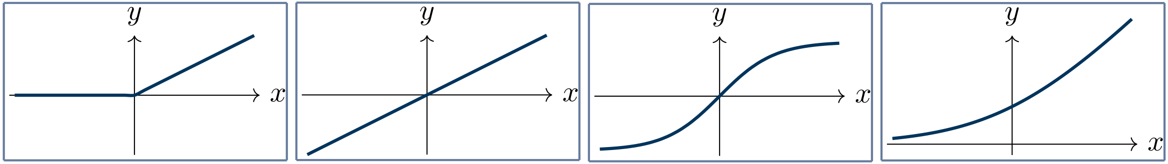
ReluBigMFormulation
`ReluComplementarityFormulation`
ReluPartitionFormulation

`FullSpaceSmoothNNFormulation`
`ReducedSpaceSmoothNNFormulation`

Optimization solver software EPL $\equiv$ Eclipse Public License; Prop $\equiv$ Proprietary

Mixed-integer linear [`Relu{BigM,Partition}Formulation`] CBC [EPL] • Gurobi [Prop] • Xpress [Prop] • CPLEX [Prop] **Nonlinear** [`{Full, Reduced}SpaceSmoothNNFormulation`, `ReluComplementarityFormulation`] Ipopt [EPL] • SNOPT [Prop] • MINOS [Prop]

How NN activation functions map onto OMLT formulations . . .



ReLU linear tanh softplus

`ReluBigMFormulation`
`ReluComplementarityFormulation`
`ReluPartitionFormulation`

`FullSpaceSmoothNNFormulation`
`ReducedSpaceSmoothNNFormulation`

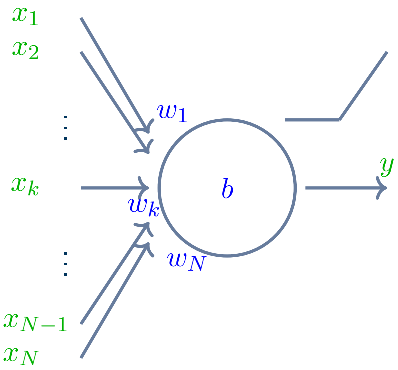
Optimization solver software EPL ≡ Eclipse Public License; Prop ≡ Proprietary

Mixed-integer linear [`Relu{BigM,Partition}Formulation`] CBC [EPL] • Gurobi [Prop] • Xpress [Prop] • CPLEX [Prop] **Nonlinear** [`{Full, Reduced}SpaceSmoothNNFormulation`, `ReluComplementarityFormulation`] Ipopt [EPL] • SNOPT [Prop] • MINOS [Prop]

OMLT puts *optimization formulations* in competition

Key idea One optimization formulation may be more effective than another

- Algebraic modelling languages, e.g., Pyomo, make switching optimization *solvers* easy
- OMLT makes switching formulations as easy as changing a couple lines of code



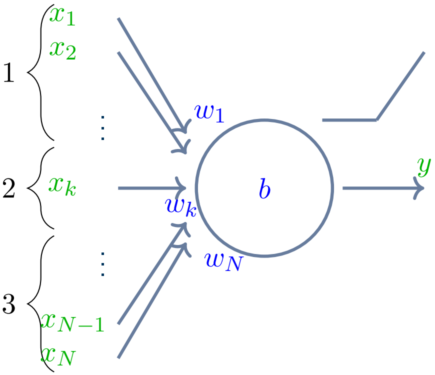
Big-M formulation [Anderson et al., 2020]

```
formulation = ReluBigMFormulation(net_relu)
```

OMLT puts *optimization formulations* in competition

Key idea One optimization formulation may be more effective than another

- Algebraic modelling languages, e.g., Pyomo, make switching optimization *solvers* easy
- OMLT makes switching formulations as easy as changing a couple lines of code



Big-M formulation [Anderson et al., 2020]

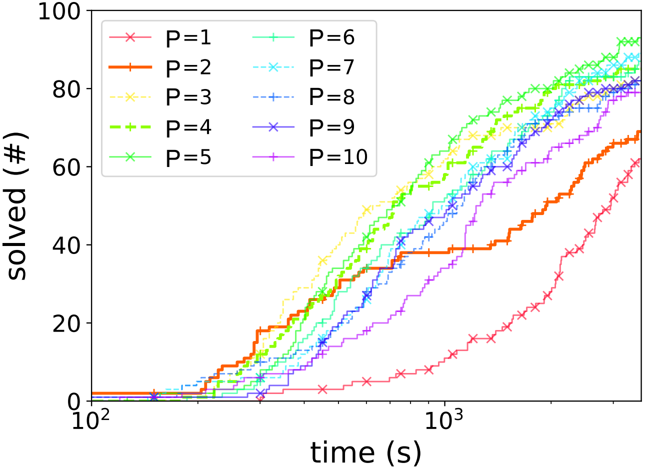
```
formulation = ReluBigMFormulation(net_relu)
```

Partition-based formulation [Tsay et al., 2021]

```
P = 3
split_func = lambda w: partition_split_func(w, P)
formulation = ReluPartitionFormulation(
    net_relu, split_func=split_func)
```

Importance of parameter P: Number solved vs. run time $\left(\|\boldsymbol{x} - \bar{\boldsymbol{x}}\|_1 = 5\right)$

Optimal adversary CIFAR-10; NN with $n_{\text{Layers}} = 2$ & $n_{\text{Hidden}} = 100$; Each line averages 100 examples $\bar{\boldsymbol{x}}$

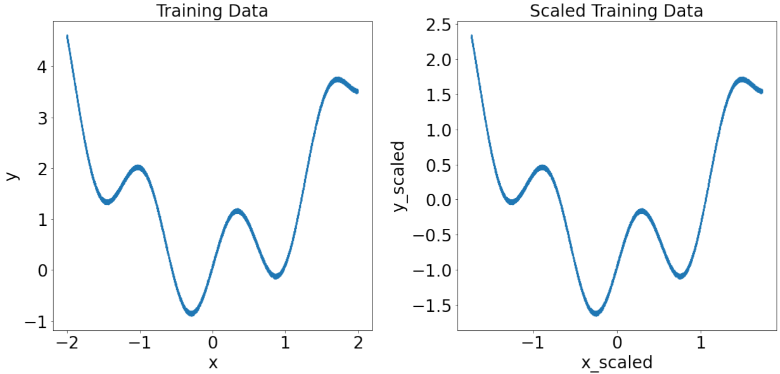


Observations [Tsay et al., 2021]

- $P = 1$ (equivalent to big-M) performs worst;
- $P = 2$ good for easy problems;
- Intermediate P balances model size vs. tightness;
- Performance declines $P \geq 7$

Neural Network Formulation Example: Data

neural_network_formulations.ipynb

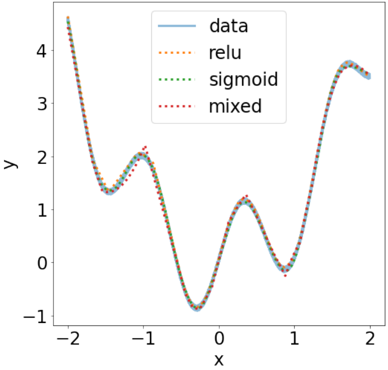


Read in the data 1 input x, 1 output y, 10^4 samples, Scaled has mean 0 & stdev 1

```
df = pd.read_csv("../data/sin_quadratic.csv",index_col=[0]);
```

Neural Network Formulation Example: Trained Neural Networks

`neural_network_formulations.ipynb`

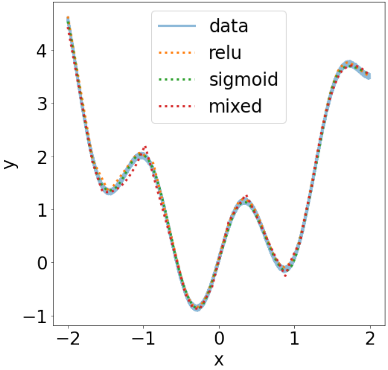


Build a Keras NN with ReLU activation

```
nn = Sequential(name='sin_wave_relu')
nn.add(Input(1))
nn.add(Dense(30, activation='relu'))
nn.add(Dense(30, activation='relu'))
nn.add(Dense(1))
nn.compile(optimizer=Adam(), loss='mse')
history = nn.fit(x=df['x_scaled'], y=df['y_scaled'],
    verbose=1, epochs=75)
```

Neural Network Formulation Example: Trained Neural Networks

neural_network_formulations.ipynb

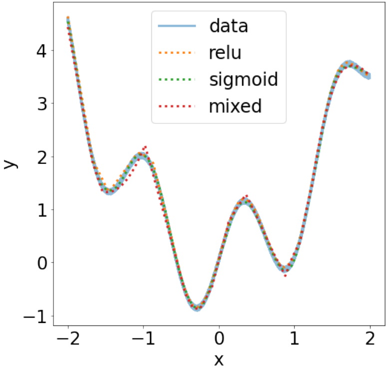


Build a Keras NN with sigmoid activation

```
nn = Sequential(name='sin_wave_sigmoid')
nn.add(Input(1))
nn.add(Dense(50, activation='sigmoid'))
nn.add(Dense(50, activation='sigmoid'))
nn.add(Dense(1))
nn.compile(optimizer=Adam(), loss='mse')
history = nn.fit(x=df['x_scaled'], y=df['y_scaled'],
    verbose=1, epochs=75)
```

Neural Network Formulation Example: Trained Neural Networks

`neural_network_formulations.ipynb`



Build a Keras NN with mixed (sigmoid/ReLU) activation

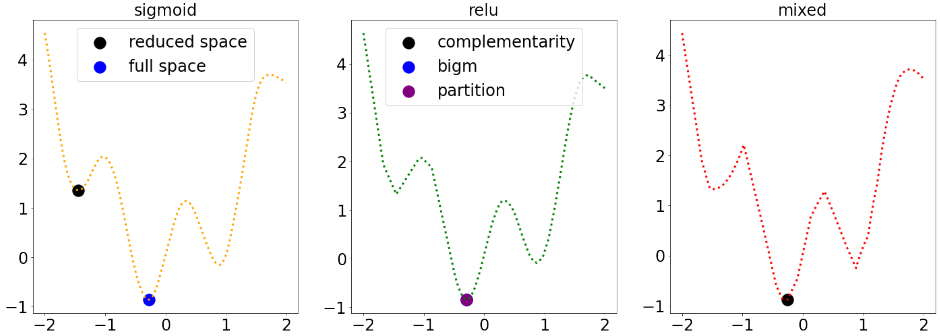
```
nn = Sequential(name='sin_wave_mixed')
nn.add(Input(1))
nn.add(Dense(50, activation='sigmoid'))
nn.add(Dense(50, activation='relu'))
nn.add(Dense(1))
nn.compile(optimizer=Adam(), loss='mse')
history = nn.fit(x=df['x_scaled'], y=df['y_scaled'],
    verbose=1, epochs=150)
```

Neural Network Formulation Example: Set up the optimization problem

```
net_sigmoid = keras_reader.load_keras_sequential(nn,scaler,input_bounds)
model = pyo.ConcreteModel()
model.x = pyo.Var(initialize = 0)
model.y = pyo.Var(initialize = 0)
model.obj = pyo.Objective(expr=(model.y))
model.nn = OmltBlock()
formulation = FullSpaceSmoothNNFormulation(net_sigmoid) #or ReducedSpaceSmoothNNFormulation
model.nn.build_formulation(formulation)

@model.Constraint()
def connect_inputs(mdl):
    return mdl.x == mdl.nn.inputs[0]

@model.Constraint()
def connect_outputs(mdl):
    return mdl.y == mdl.nn.outputs[0]

status = pyo.SolverFactory('ipopt').solve(model, tee=True)
solution = (pyo.value(model.x),pyo.value(model.y))
```

Neural Network Formulation Example: Optimization results

`neural_network_formulations.ipynb`



FullSpaceSmoothNNFormulation [Ipopt]

variables: 209, # constraints: 208

$x = -0.28, y = -0.86$

Solve Time: 0.14s

ReducedSpaceSmoothNNFormulation [Ipopt]

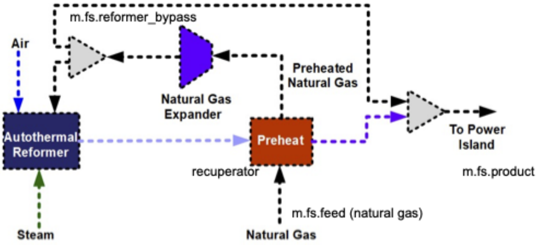
variables: 6, # constraints: 5

$x = -1.44, y = 1.36$

Solve Time: 0.08s

Other notebook examples . . .

https://github.com/cog-imperial/OMLT/tree/main/docs/notebooks



auto-thermal-reformer{-relu}.ipynb

develops an NN surrogate with data from a process model built using IDAES-PSE [Lee et al., 2021]

Even more notebook examples . . .

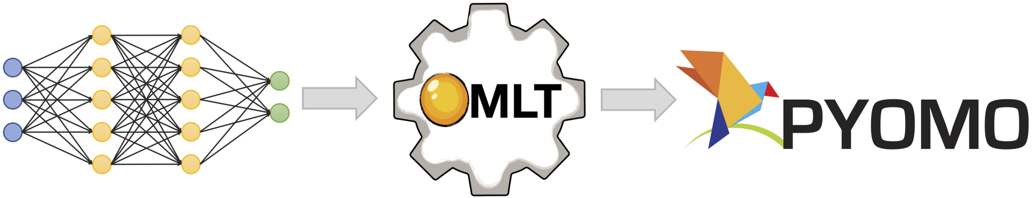
- `import_network.ipynb` imports NN models directly from Keras & ONNX. Using ONNX interoperability, it imports a NN model from PyTorch.
- `build_network.ipynb` builds a `NetworkDefinition` manually.
- `mnist_example_{dense, cnn}.ipynb` train fully dense and convolutional NNs on MNIST [LeCun et al., 2010] and find adversarial examples [Tjeng et al., 2017].
- `bo_with_trees.ipynb` optimizes the Rosenbrock function.

OMLT v 1.0 Summary

https://github.com/cog-imperial/OMLT

Key Contributions

- Automatically translate a trained machine learning model (neural network or gradient boosted tree) into Pyomo optimization constraints
- Achieve interoperability via the ONNX interface
- Easily switch and compare optimization formulations



References I

Michael Akintunde, Alessio Lomuscio, Lalit Maganti, and Edoardo Pirovano. Reachability analysis for neural agent-environment systems. In *KR*, pages 184–193, 2018.

Ross Anderson, Joey Huchette, Will Ma, Christian Tjandraatmadja, and Juan Pablo Vielma. Strong mixed-integer programming formulations for trained neural networks. *Mathematical Programming*, pages 1–37, 2020.

Egon Balas. *Disjunctive Programming*. Springer International Publishing, 2018. doi: 10.1007/978-3-030-00148-3.

Elena Botoeva, Panagiotis Kouvaros, Jan Kronqvist, Alessio Lomuscio, and Ruth Misener. Efficient verification of ReLU-based neural networks via dependency analysis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 3291–3299, 2020.

Michael L Bynum, Gabriel A Hackebeil, William E Hart, Carl D Laird, Bethany L Nicholson, John D Siirola, Jean-Paul Watson, and David L Woodruff. *Pyomo—Optimization Modeling in Python*, volume 67. Springer Nature, 2021.

Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. Maximum resilience of artificial neural networks. In *International Symposium on Automated Technology for Verification and Analysis*, pages 251–268. Springer, 2017.

References II

Francesco Croce and Matthias Hein. Minimally distorted adversarial examples with a fast adaptive boundary attack. In *International Conference on Machine Learning*, pages 2196–2205. PMLR, 2020.

Alessandro De Palma, Harkirat Singh Behl, Rudy Bunel, Philip HS Torr, and M Pawan Kumar. Scaling the convex barrier with active sets. *arXiv preprint arXiv:2101.05844*, 2021.

Souradeep Dutta, Susmit Jha, Sriram Sankaranarayanan, and Ashish Tiwari. Output range analysis for deep feedforward neural networks. In *NASA Formal Methods Symposium*, pages 121–138. Springer, 2018.

Krishnamurthy Dvijotham, Robert Stanforth, Sven Gowal, Timothy A Mann, and Pushmeet Kohli. A dual approach to scalable verification of deep networks. In *UAI*, volume 1, page 3, 2018.

Matteo Fischetti and Jason Jo. Deep neural networks and mixed integer linear optimization. *Constraints*, 23(3): 296–309, 2018.

Bjarne Grimstad and Henrik Andersson. ReLU networks as surrogate models in mixed-integer linear programs. *Computers & Chemical Engineering*, 131:106580, 2019.

Gurobi Optimization, LLC. Gurobi optimizer reference manual, 2020. URL `http://www.gurobi.com`.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

References III

Jan Kronqvist, Ruth Misener, and Calvin Tsay. Between steps: Intermediate relaxations between big-M and convex hull formulations. *CPAIOR*, 2021.

Yann LeCun, Corinna Cortes, and CJ Burges. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

Andrew Lee, Jaffer H Ghouse, John C Eslick, Carl D Laird, John D Siirola, Miguel A Zamarripa, Dan Gunter, John H Shinn, Alexander W Dowling, Debangsu Bhattacharyya, et al. The IDAES process modeling framework and model library—Flexibility for process simulation and optimization. *Journal of Advanced Manufacturing and Processing*, page e10095, 2021.

Alessio Lomuscio and Lalit Maganti. An approach to reachability analysis for feed-forward ReLU neural networks. *arXiv preprint arXiv:1706.07351*, 2017.

Velibor V Mišić. Optimization of tree ensembles. *Operations Research*, 68(5):1605–1624, 2020.

Miten Mistry, Dimitrios Letsios, Gerhard Krennrich, Robert M Lee, and Ruth Misener. Mixed-integer convex nonlinear optimization with gradient-boosted trees embedded. *INFORMS Journal on Computing*, 33(3): 1103–1119, 2021.

References IV

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. 2019.

Artur M Schweidtmann and Alexander Mitsos. Deterministic global optimization with artificial neural networks embedded. *Journal of Optimization Theory and Applications*, 180(3):925–948, 2019.

Thiago Serra, Christian Tjandraatmadja, and Srikumar Ramalingam. Bounding and counting linear regions of deep neural networks. In *International Conference on Machine Learning*, pages 4558–4566. PMLR, 2018.

Thiago Serra, Abhinav Kumar, and Srikumar Ramalingam. Lossless compression of deep neural networks. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 417–430. Springer, 2020.

Alexander Thebelt, Jan Kronqvist, Miten Mistry, Robert M Lee, Nathan Sudermann-Merx, and Ruth Misener. ENTMOOT: A framework for optimization over ensemble tree models. *Computers & Chemical Engineering*, 151:107343, 2021.

References V

Vincent Tjeng, Kai Xiao, and Russ Tedrake. Evaluating robustness of neural networks with mixed integer programming. *arXiv preprint arXiv:1711.07356*, 2017.

Calvin Tsay, Jan Kronqvist, Alexander Thebelt, and Ruth Misener. Partition-based formulations for mixed-integer optimization of trained ReLU neural networks. *NeurIPS*, 2021.

Michael Volpp, Lukas P Fröhlich, Kirsten Fischer, Andreas Doerr, Stefan Falkner, Frank Hutter, and Christian Daniel. Meta-learning acquisition functions for transfer learning in Bayesian optimization. In *International Conference on Learning Representations*, 2019.

Ga Wu, Buser Say, and Scott Sanner. Scalable planning with deep neural network learned transition models. *Journal of Artificial Intelligence Research*, 68:571–606, 2020.