Embedding neural networks into optimization models with GAMSPy

André Schnabel,[1] Hamdi Burak Usul,[2]

[1]*GAMS Software GmbH, Germany,* aschnabel@gams.com

[2]*GAMS Software GmbH, Germany,* busul@gams.com

Abstract GAMSPy provides Python's flexibility with GAMS's modeling power. It bridges the gap between machine learning (ML) and traditional mathematical modeling by offering auxiliary classes for common neural network (NN) layers and activation functions, automatically converting network architectures into GAMSPy model expressions. In this article, we demonstrate how GAMSPy enables embedding a pre-trained NN into an optimization model.

Keywords: Modeling Languages, Machine Learning, Neural Networks

1. Introduction

The General Algebraic Modeling System (GAMS) started as the first domain specific language for mathematical modeling in the 1970s at the World Bank. Over the years, the algebraic modeling language evolved and the GAMS distribution grew with many supporting software products. The Python package GAMSPy[1] is a more recent addition, and allows specifying models directly in Python in a way that closely resembles the design philosophy of the GAMS language.

While using ML models to generate or pre-process input parameters might be sufficient for some problems, many problems require a tighter in-

[1]`https://gamspy.readthedocs.io`

tegration of machine learning and optimization models by embedding the ML model. To support this, GAMSPy was extended with the `formulations` package[2] package offering various automatic reformulations.

Similar packages for other modeling solutions include `OMLT` for Pyomo and JuMP [2], `gurobi-machinelearning` for gurobipy, `MathOptAI.jl` for JuMP [4], `PySCIPOpt-ML` for PySCIPOpt [5]. A good comparison of the supported layer types can be found in [3]. In previous benchmarks, the GAMS execution system utilized by GAMSPy often outperforms Pyomo and JuMP in model generation times.[3]

2. Surrogate model example

The following model is an adaption of the section 4.1 of Bhosekar and Ierapetritou [1]. It is a simple artificial problem that exhibits basic properties of real-world problems found in chemical engineering. A stream of input material A $[\frac{\text{mol}}{h}]$ is filled into a reactor r and then converted into two output products B and E by a separator module s. The demands of B $[\frac{\text{mol}}{h}]$ and E $[\frac{\text{mol}}{h}]$ are known and the task is to choose reactor and separator modules that minimize the overall costs. The reactors differ by volume V_r $[m^3]$ and price c_r [k$]. The separators are assumed to be ideal, and are characterized by bounds lb_s and ub_s on the amount of throughput they can process per hour and price c_s. Table 1 shows the specifications of the available reactor and separator modules.

Table 1: Design options for reactor and separator.

Options	**Reactor ($\mathbf{m^3}$)**	c_r **(k$)**	**Separator ($\mathbf{F_A}$ mol/h)**	c_s **(k$)**
Option 1	5	400	30–50	300
Option 2	20	850	40–70	720
Option 3	35	1200	60–100	980
Option 4	50	1650	90–140	1210

The solution methodology in [1] involves obtaining the feasibility constraints via a simulation implemented as nonlinear program. When one instead uses as surrogate model (a NN trained on the simulation) to determine the probability of feasibility for a given reactor volume V, input flow rate F_a, and demanded output flow rates F_B and F_E, the remaining model can be expressed as the following mixed-integer linear program (MILP):

[2] `https://gamspy.readthedocs.io/en/latest/reference/gamspy.formulations.html`
[3] As shown in "Performance in Optimization Models" at `https://tinyurl.com/gamsperf`

$$
\begin{aligned}
\text{minimize} \quad & \sum_r c_r \cdot y_r + \sum_s c_s \cdot y_s + c_a \cdot F_a && (1)\\
\text{s.t.} \quad & \sum_r y_r = 1, \quad \sum_s y_s = 1 && (2)\\
& V = \sum_r V_r \cdot y_r && (3)\\
& NN(V, F_a, F_B, F_E) \geq 0.99 && (4)\\
& F_a \leq ub_s + (1 - y_s) \cdot M \quad \forall s && (5)\\
& F_a + (1 - y_s) \cdot M \geq lb_s \quad \forall s && (6)\\
& y_r \in \{0,1\}, y_s \in \{0,1\}, F_a \in \mathbb{R}_+, V \in \mathbb{R}_+ && (7)
\end{aligned}
$$

The objective function (1) computes the total cost incurred by the choice of reactor r and selector s plus the cost for the input material flow rate F_a. Constraints (2) ensure exactly one reactor and exactly one separator is chosen. Equation (3) links the auxiliary variable V for the available volume with the volume of the chosen reactor. Constraint (4) uses the embedded NN to predict a feasibility probability for a given tuple of reactor volume V, input flow rate F_a and output material flow rate demands F_B and F_E. The feasibility prediction acquired through NN inference must be at least 99%. Equations (5) and (6) make sure the input material flow rate is inside the operational bounds of the chosen selector using a bigM-formulation. The decision variable domains follow in (7), declaring the input material flow rate and available volume auxiliary variable as continuous, and the reactor- y_r and selector-choice y_s indicator variables as binary.

To solve the model with a MILP solver, the parameterized NN term NN in (5) must be expanded into a set of additional constraints and variables, such that the left-hand side of (5) evaluates to the value acquired by doing a forward propagation of input vector (V, F_a, F_B, F_E) through the network layers. The NN is sequential and consists of 4 linear layers connected with ReLU and sigmoid as activation functions for the inner layers and output layer respectively. Listing 1.1 shows relevant Python code for including the trained NN inside the model. The full source code is available on GitHub[4].

[4]`https://github.com/GAMS-dev/surrogate-model`

```
model = nn.Sequential(nn.Linear(4, 10), nn.ReLU(), nn.Linear(10, 10),
   nn.ReLU(), nn.Linear(10, 15), nn.ReLU(), nn.Linear(15, 1),
   nn.Sigmoid())
drop_sigmoid_model = nn.Sequential(*list(model.children())[:-1])
# train network
...
m = gp.Container()
# input to neural network: a0=[V,Fa,Fb,Fe]
a0 = gp.Variable(m, name="a0", domain=gp.math.dim([4]))
a1 = gp.Variable(m, name="a1", domain=gp.math.dim([4])) # a0 normalized
normalize_input[...] = a1 == (a0 - x_mean_par) / x_std_par
...
seq_formulation = gp.formulations.TorchSequential(m, drop_sigmoid_model)
z5, _ = seq_formulation(a1)
check_feasibility[...] = z5[0] >= 4.59511985013459 # 0.99 probability
```

Listing 1.1: Code excerpt for linking the trained NN into the MILP

3. Numerical results

The solver Baron solves the instance from Table 1 to optimality within 0.06 seconds given a white-box monolithic MINLP whereas CPLEX solves the surrogate model MIP in only 0.03 seconds. Solution quality is identical, as the loss of precision from the NN approximation in (4) does not cut the optimal solution from the feasible region. Besides computational gains, embedding NN has more advantages. Simulations are not always available in closed form expression, i.e., when using a proprietary simulator software. Using NN also works when only historical data from the physical plant is available and no simulation model exists.

4. Summary

This article showed how to solve a basic problem by formulating it as a MILP where a NN was integrated as the left-hand side of a constraint. The NN was implemented in PyTorch and the optimization model in GAMSPy. The GAMSPy package `formulations` was used to replicate the structure of the NN in the MILP. Integrating NN and optimization models can be used for many other applications like adversarial input generation, model verification, and customized training of NN.

References

[1] Atharv Bhosekar and Marianthi Ierapetritou. Modular design optimization using machine learning-based flexibility analysis. *Journal of Process Control*, 90:18–34, 2020.

[2] Francesco Ceccon, Jordan Jalving, Joshua Haddad, Alexander Thebelt, Calvin Tsay, Carl D Laird, and Ruth Misener. Omlt: Optimization & machine learning toolkit. *Journal of Machine Learning Research*, 23(349):1–8, 2022.

[3] Oscar Dowson, Robert B Parker, and Russel Bent. Mathoptai.jl: Embed trained machine learning predictors into jump models. *arXiv preprint arXiv:2507.03159*, 2025.

[4] Robert B. Parker, Oscar Dowson, Nicole LoGiudice, Manuel Garcia, and Russell Bent. Formulations and scalability of neural network surrogates in nonlinear optimization problems. *arXiv preprint arXiv:2412.11403*, 2024.

[5] Mark Turner, Antonia Chmiela, Thorsten Koch, and Michael Winkler. Pyscipopt-ml: Embedding trained machine learning models into mixed-integer programs. In *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*, pages 218–234. Springer, 2025.