GAMS
MODEL – SOLVE – DEPLOY

GAMSPy

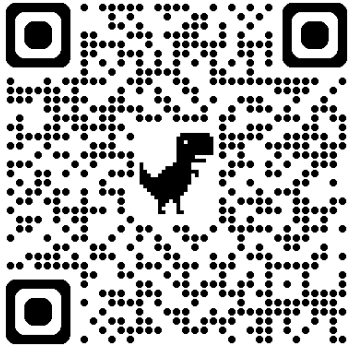
Fast and Flexible Optimization in Python

Adam Christensen
Steve Dirkse

Academic Program 🎓

GAMS
MODEL – SOLVE – DEPLOY

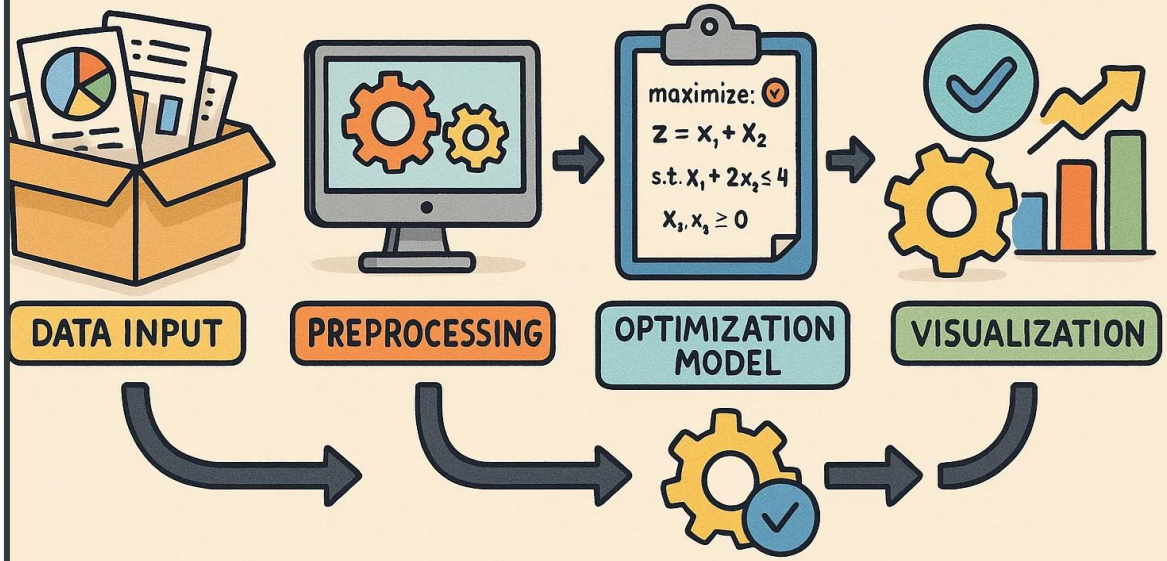
- Free GAMS and GAMSPy licenses
  - No model size limit on GAMSPy
  - Access to all open-source solvers
  - Access to commercial solvers!
    - CPLEX, CONOPT, COPT, MOSEK, XPRESS, GUROBI-link
  - Sign up at 🔗 https://academic.gams.com/ 🔗



- Other resources
  - 💬 Community FORUM 💬 at https://forum.gams.com/
  - 🎥 YouTube channel 🎥 at https://www.youtube.com/@GAMSLessons
  - Ready-to-run notebooks, case studies, blog posts, and more!

GAMS
MODEL – SOLVE – DEPLOY

MATHEMATICAL PROGRAMMING (OPTIMIZATION) PIPELINE



python™
Cython
R
jupyter
NumPy
pandas
NetworkX
Network Analysis in Python

GAMS Advantages

GAMS
MODEL – SOLVE – DEPLOY

- Algebraic modeling language (AML)
- Active development since 1987 - ***dependable!***
- Backward Compatibility - ***stable!***
- Familiar syntax - ***human readable!***
- Data is held in GAMS structures - ***very large models!***
- Optimized execution - ***superior performance!***
- ***Solver Independence – use what you need!***

Python Advantages

GAMS
MODEL – SOLVE – DEPLOY

- Ubiquitous: *everybody's doing it!*
- Huge variety of third-party packages
- Ecosystems (VSCode, JuPyter, PyCharm, etc.)
- Language itself has much to offer
- Supports wrapping of other software

GAMSPy: Best of Both Worlds

GAMS
MODEL – SOLVE – DEPLOY

- Pure Python code
- Algebraic model specification
- Parse/interpret code -> abstract syntax tree
- Execute in GAMS (server mode)

My First GAMSPy

GAMS
MODEL – SOLVE – DEPLOY

```
m = Container()

i = Set(m, "i")
j = Set(m, "j")
a = Parameter(m, "a", i)
b = Parameter(m, "b", j)
c = Parameter(m, "c", [i, j])
x = Variable(m, "x", "positive", domain=[i, j])
supply = Equation(m, "supply", domain=i)
demand = Equation(m, "demand", domain=j)

supply[i] = Sum(j, x[i, j]) <= a[i]
demand[i] = Sum(i, x[i, j]) >= b[j]
```

Abstract Model

Declared/defined over sets

Like “writing on paper”

Compact

Logically consistent (no domain violations, uncontrolled sets)

Completely abstract (no data)

Leverages operator overloading

My First GAMSPy

GAMS
MODEL – SOLVE – DEPLOY

```
m = Container()

i = Set(m)
j = Set(m)
a = Parameter(m, i)
b = Parameter(m, j)
c = Parameter(m, [i, j])
x = Variable(m, "positive", domain=[i, j])
supply = Equation(m, domain=i)
demand = Equation(m, domain=j)

supply[i] = Sum(j, x[i, j]) <= a[i]
demand[i] = Sum(i, x[i, j]) >= b[j]
```

Nameless Syntax!

Abstract Model

Declared/defined over sets

Like “writing on paper”

Compact

Logically consistent (no domain violations, uncontrolled sets)

Completely abstract (no data)

Leverages operator overloading

My First GAMSPy

GAMS
MODEL – SOLVE – DEPLOY

```
m = Container()
```

Early Errors

```
x = Variable(m, "x", "positive", domain=[i, j])
supply = Equation(m, "supply", domain=i)
demand = Equation(m, "demand", domain=j)

supply[i] = Sum(j, x[i, j]) <= a[i]
demand[i] = Sum(i, x[i, i]) >= b[j] // Problem here!
------------------------------------------------------------
ValidationError         Traceback (most recent call last)
Cell In[1], line 15
---> 15 demand[i] = Sum(i, x[i, i]) >= b[j]

ValidationError: `Given set `i` is not a valid domain for
declared domain `j`
```

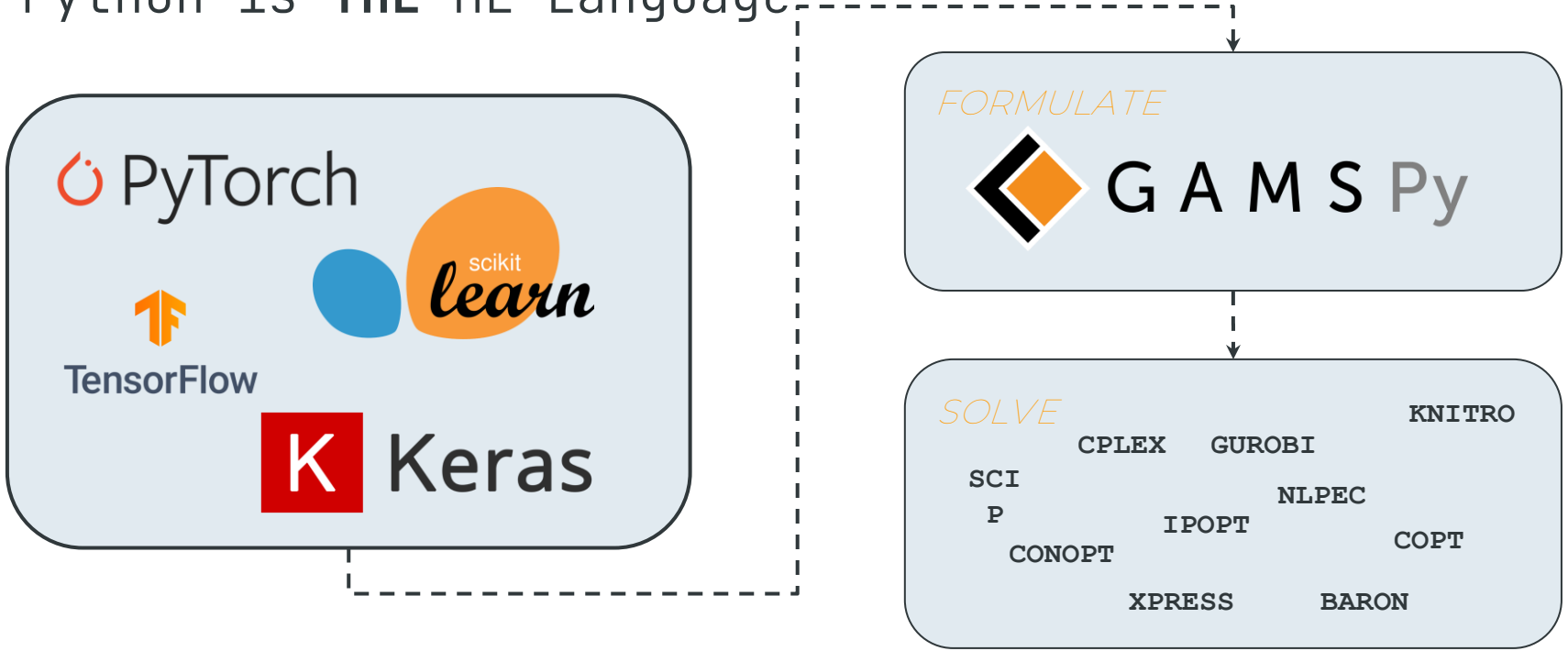
GAMS Philosophy

Tight syntax leads to better modeling

New Opportunities

GAMS
MODEL – SOLVE – DEPLOY

Python is **THE** ML Language



Matrices in GAMSPy

GAMS
MODEL – SOLVE – DEPLOY

```python
from gamspy import Container, Parameter, Variable
from gamspy.math import dim

m = Container

w = Parameter(m, "w", domain=dim([10, 20]))
x = Variable(m, domain=dim([40, 10]))
x = Variable(m, domain=dim([40, 10]))
                    dense data structures


eq[...] = y == x @ w
          Overload of __matmult__
```

GAMS Philosophy

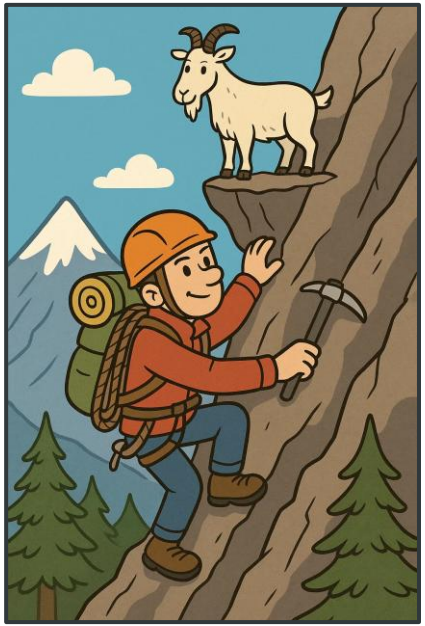
Tight syntax leads to better modeling

Formulations

GAMS
MODEL – SOLVE – DEPLOY

ML in MP – *tedious and error prone*

- No single right answer
- Will evolve over time
- Downstream implications
- Common constructs in ML
- Automation is key

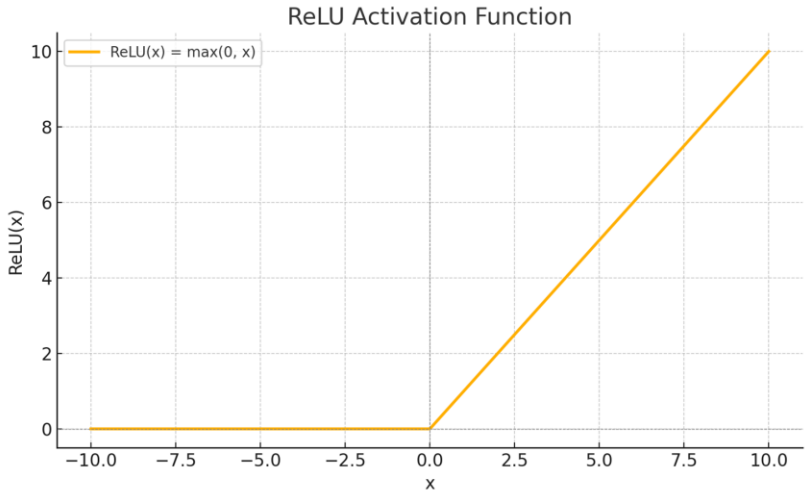


Activation Formulations

GAMS
MODEL – SOLVE – DEPLOY

Activation functions

ReLU	$\max(0, x)$
tanh	$\tanh(x)$
softmax	$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{i} e^{z_j}}$
Log softmax	$\text{LogSoftmax}(z_i) = \log\left(\frac{e^{z_i}}{\sum_{j=1}^{i} e^{z_j}}\right)$
Sigmoid	$y = \frac{1}{1 + \exp(-x)}$



Activation Formulations

GAMS
MODEL – SOLVE – DEPLOY

Activation functions

ReLU	$\max(0, x)$
tanh	$\tanh(x)$
softmax	$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{i} e^{z_j}}$
Log softmax	$\text{LogSoftmax}(z_i) = \log\left(\frac{e^{z_i}}{\sum_{j=1}^{i} e^{z_j}}\right)$
Sigmoid	$y = \frac{1}{1+\exp(-x)}$

$$y = ReLU(x) = \max(0, x)$$
$$y \geq 0$$
$$y \geq x$$
$$y \leq x - (1-\sigma)x_{lb}$$
$$y \leq \sigma x_{ub}$$
$$\sigma \in 0, 1$$

(MIP, new binary vars)

$$y = ReLU(x) = \max(0, x)$$
$$y \geq 0$$
$$y(y-x) = 0$$
$$y - x \geq 0$$
$$\text{i.e. } (0 \leq y \perp (y-x) \geq 0)$$

Complementarity
(MPEC->NLP transform)

Activation Formulations

GAMS
MODEL – SOLVE – DEPLOY

Activation functions

ReLU	$\max(0, x)$
tanh	$\tanh(x)$
softmax	$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{i} e^{z_j}}$
Log softmax	$\text{LogSoftmax}(z_i) = \log\left(\frac{e^{z_i}}{\sum_{j=1}^{i} e^{z_j}}\right)$
Sigmoid	$y = \frac{1}{1 + \exp(-x)}$

```
from gamspy import Container, Set, Variable
from gamspy.math.activation import
relu_with_binary_var

m = Container()
i = Set(m, "i", records=range(3))
x = Variable(m, "x", domain=[i])


y, b, eqs = relu_with_binary_var(x,
return_binary_var=True)

Out[1]: <Container (0x1056c8190) with 9 symbols>
```

Formulation Generator

Activation Formulations

GAMS
MODEL – SOLVE – DEPLOY

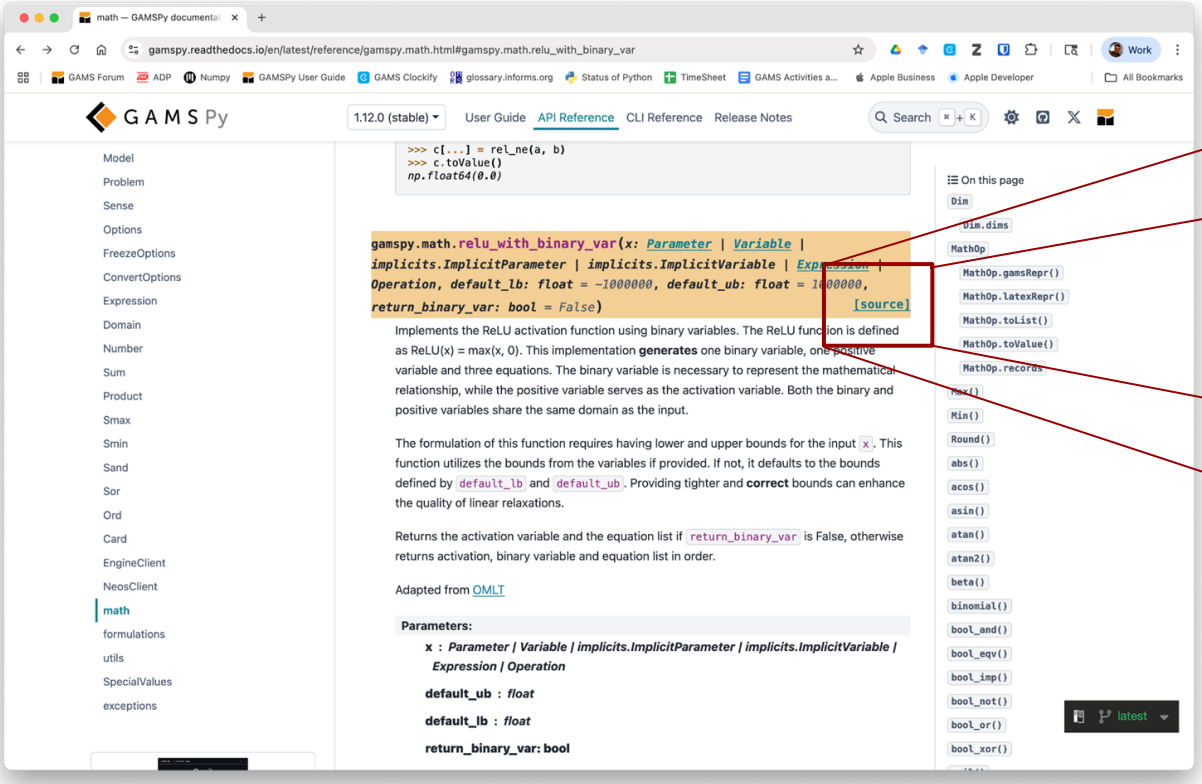
Activation functions	
ReLU	$\max(0, x)$
tanh	$\tanh(x)$
softmax	$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{i} e^{z_j}}$
Log softmax	$\text{LogSoftmax}(z_i) = \log\left(\frac{e^{z_i}}{\sum_{j=1}^{i} e^{z_j}}\right)$
Sigmoid	$y = \frac{1}{1 + \exp(-x)}$

```
y, b, eqs = relu_with_binary_var(x,
return_binary_var=True)

>>>

eq[1][...] = y <= x - (1 - sigma) * _get_lb(x,
default_lb)
eq[2][...] = y <= sigma * _get_ub(x, default_ub)
y.lo[...] = gamspy.math.Max(0, x.lo[...])
y.up[...] = gamspy.math.Max(0, x.up[...])

<<<
```

THE FORMULATION

Activation Formulations

GAMS
MODEL – SOLVE – DEPLOY



Expand source for the details

Formulations

GAMS
MODEL – SOLVE – DEPLOY

Layers

Linear
Conv1D, Conv2D
Min/Max/Ave Pooling
Flattening (combine domains)

```python
import gamspy as gp
import numpy as np
from gamspy.math import dim

# weights & bias
w = np.random.rand(64, 128)
b = np.random.rand(64)

m = gp.Container()

lin_1 = gp.formulations.Linear(m, 128, 64)

lin_1.load_weights(w, b)
x = gp.Variable(m, "x", domain=dim([10, 128]))

y, set_y = lin_1(x)
```

Formulation Generator

Formulations

GAMS
MODEL – SOLVE – DEPLOY

Layers

Linear
Conv1D, Conv2D
Min/Max/Ave Pooling
Flattening (combine domains)

```python
import gamspy as gp
import numpy as np
from gamspy.math import dim

# weights & bias
w = np.random.rand(64, 128)
b = np.random.rand(64)

m = gp.Container()

lin_1 = gp.formulations.Linear(m, 128, 64)

lin_1.load_weights(w, b)
x = gp.Variable(m, "x", domain=dim([10, 128]))

y, set_y = lin_1(x)
```

Formulation Generator

Now, call it

Formulations

GAMS
MODEL – SOLVE – DEPLOY

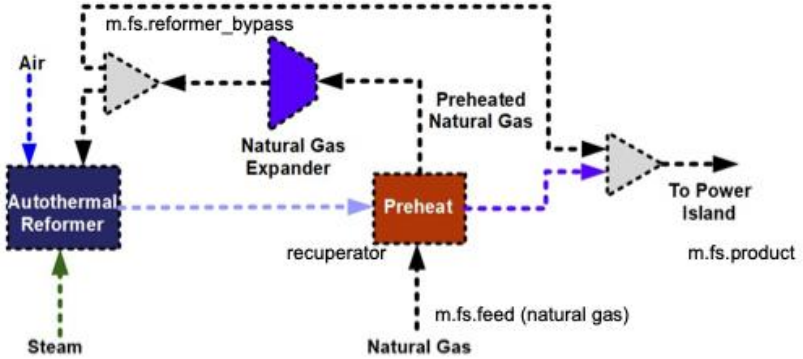
Layers

Linear
Conv1D, Conv2D
Min/Max/Ave Pooling
Flattening (combine domains)

```
import gamspy as gp
import numpy as np
from gamspy.math import dim

y, set_y = lin_1(x)

print(m)

>>>

<Container (0x115df6510) with 11 symbols>

<<<
```

Now, call it

Example - Reformer Process



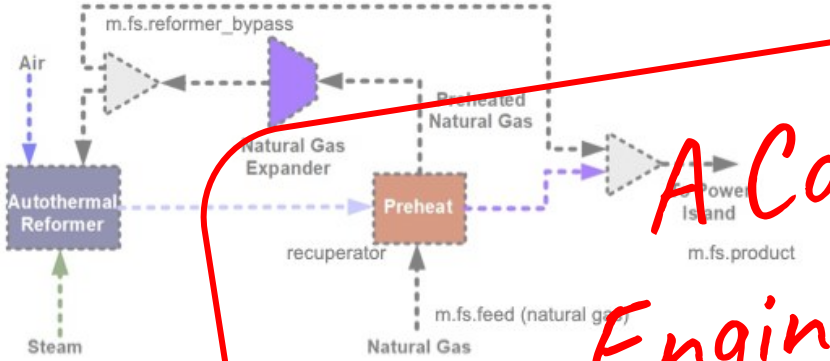
An auto-thermal reformer (ATR) used in hydrogen and syngas production

1. Partial Oxidation: Where natural gas reacts with oxygen
2. Steam Reforming: Where natural gas reacts with steam

These combined reactions produce a mixture of hydrogen, carbon monoxide, and other gases.

Reformer process as modeled in IDAES

Example - Reformer Process



A Complex Engineering** System

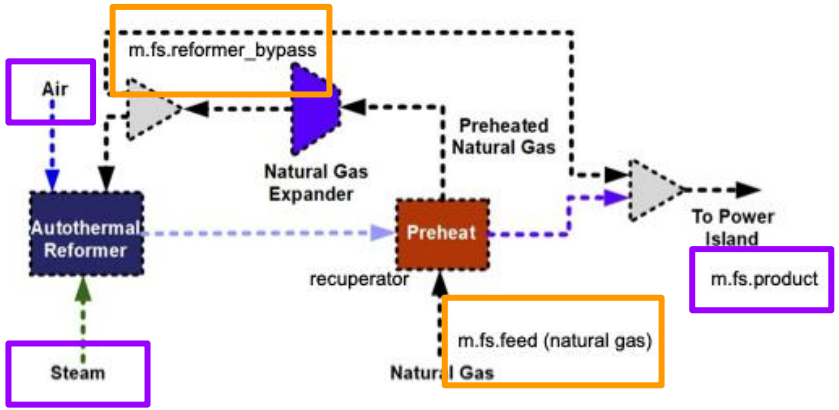
An auto-thermal reformer (ATR) used in hydrogen and syngas production

1. Partial Oxidation: Where natural gas reacts with oxygen
2. Steam Reforming: Where natural gas reacts with steam

These combined reactions produce a mixture of hydrogen, carbon monoxide, and other gases.

Reformer process as modeled in IDAES

Example - Reformer Process



maximize hydrogen production

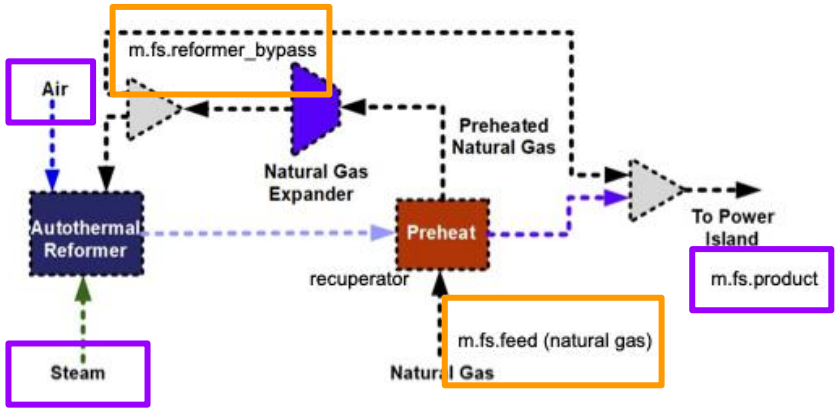
s.t. constraints on nitrogen concentration

Input Variables:
- Bypass Fraction
- Nat Gas / Steam Ratio

Output Variables:
- Steam flow
- Reformer duty
- Steam Composition
  - H2 (hydrogen)
  - Argon
  - C2H6 (ethane)
  - C3H8 (propane)
  - C4H10 (butane)
  - …

Reformer process as modeled in IDAES

Example - Reformer Process



Input Variables:
- Bypass Fraction
- Nat Gas / Steam Ratio

Output Variables:
- Steam flow
- Reformer duty
- Steam Composition
  - H2 (hydrogen)
  - Argon
  - C2H6 (ethane)
  - C3H8 (propane)
  - C4H10 (butane)
  - …

No existing first principles model, but lots of data!

Reformer process as modeled in IDAES

Train the NN with PyTorch

GAMS
MODEL – SOLVE – DEPLOY

```python
class NeuralNetwork(nn.Module):
    def __init__(self):
        super().__init__()
        self.l1 = nn.Linear(2, 10)  # Input layer: 2 -> 10
        self.l2 = nn.Linear(10, 10)  # Hidden layer 1
        self.l3 = nn.Linear(10, 10)  # Hidden layer 2
        self.l4 = nn.Linear(10, 10)  # Hidden layer 3
        self.l5 = nn.Linear(10, 12)  # Output layer: 10 -> 12

    def forward(self, x):
        # Define how data flows through the network
        relu = nn.ReLU()
        x = self.l1(x)  # First linear transformation
        x = relu(x)  # Apply ReLU activation
        x = self.l2(x)  # Second linear transformation
        x = relu(x)  # Apply ReLU activation
        x = self.l3(x)  # Third linear transformation
        x = relu(x)  # Apply ReLU activation
        x = self.l4(x)  # Fourth linear transformation
        x = relu(x)  # Apply ReLU activation
        x = self.l5(x)  # Final linear transformation
        return x
```

```python
# training function
def train(model, train_loader, optimizer, epoch):
    model.train()  # Set model to training mode
    for batch_idx, (data, target) in enumerate(train_loader):
        # Zero the parameter gradients
        optimizer.zero_grad()

        # Forward pass
        output = model(data)

        # Calculate loss (mean squared error)
        loss = F.mse_loss(output, target)

        # Backward pass and optimize
        loss.backward()
        optimizer.step()
```

Lots of experimentation

Begin adding Neural Net

```
# Create a GAMSPy container

m = gp.Container()

with torch.no_grad():
    lin1 = gp.formulations.Linear(m, in_features=2, out_features=10)
    lin1.load_weights(model.l1.weight.numpy(), model.l1.bias.numpy())

    lin2 = gp.formulations.Linear(m, in_features=10, out_features=10)
    lin2.load_weights(model.l2.weight.numpy(), model.l2.bias.numpy())

    lin3 = gp.formulations.Linear(m, in_features=10, out_features=10)
    lin3.load_weights(model.l3.weight.numpy(), model.l3.bias.numpy())

…
```

Numpy Arrays

Extract Data from NN

```
# Define variables for the original (unnormalized) inputs, these are what we'll actually
optimize
a0 = gp.Variable(m, name="a0", domain=gp.math.dim([2]))

# Define variables for the normalized inputs, these are what the neural network will use
a1 = gp.Variable(m, name="a1", domain=gp.math.dim([2]))

[. . . Data Normalization Steps . . .]

# Implement NN -> each layer is a linear transformation + ReLU
z2, _ = lin1(a1)  # First linear layer
a2, _ = relu(z2)  # First ReLU activation

z3, _ = lin2(a2)  # Second linear layer
a3, _ = relu(z3)  # Second ReLU activation
. . .
z6, _ = lin5(a5)  # Output layer (no ReLU)
```

Solve

GAMS
MODEL – SOLVE – DEPLOY

```
# Add constraint on nitrogen concentration -> keep N2 below 34%
eq1 = gp.Equation(m, name="n2_limit")
eq1[...] = z7[n2_idx] <= 0.34

# Create and solve the model
model = gp.Model(
    m,
    name="thermal_reformer",
    objective=z7[h2_idx],  # Maximize hydrogen concentration
    equations=m.getEquations(),
    sense="max",
    problem="mip",
)

model.solve(solver="cplex")
```

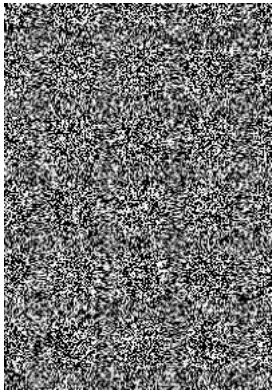
A constraint of interest

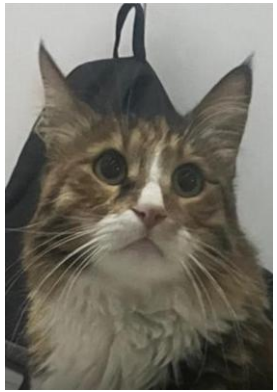
Objective

Optimization (Robustness Example)

$$min_{noise} NN(input)_{correct} - NN(input)_{incorrect}$$

$$\|noise\|_\infty \leq \epsilon$$
$$input = image + noise$$
$$noise \in \mathbb{R}^{3\times H\times W}$$
$$image \in \mathbb{R}^{3\times H\times W}$$
$$input \in \mathbb{R}^{3\times H\times W}$$
$$NN : \mathbb{R}^{3\times H\times W} \rightarrow \mathbb{R}^2$$





xx

yy

German Sign Recognition Benchmark

43 traffic signs

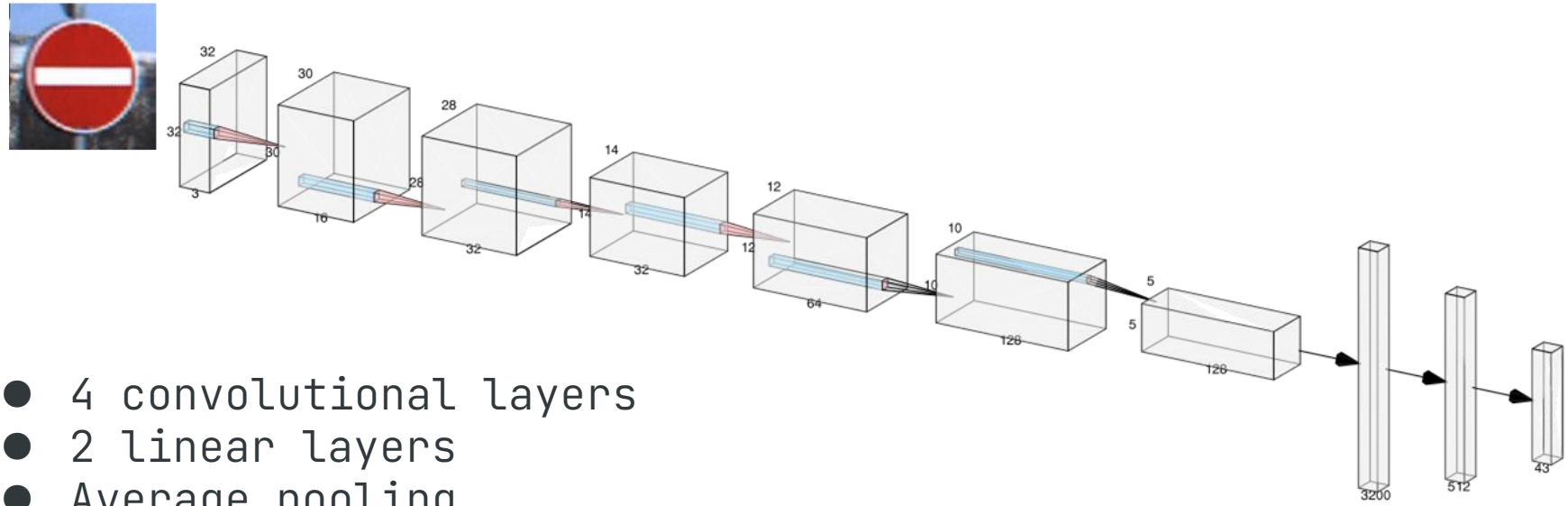
Real life example

Not trivially recognizable

A critical example

>50,000 images

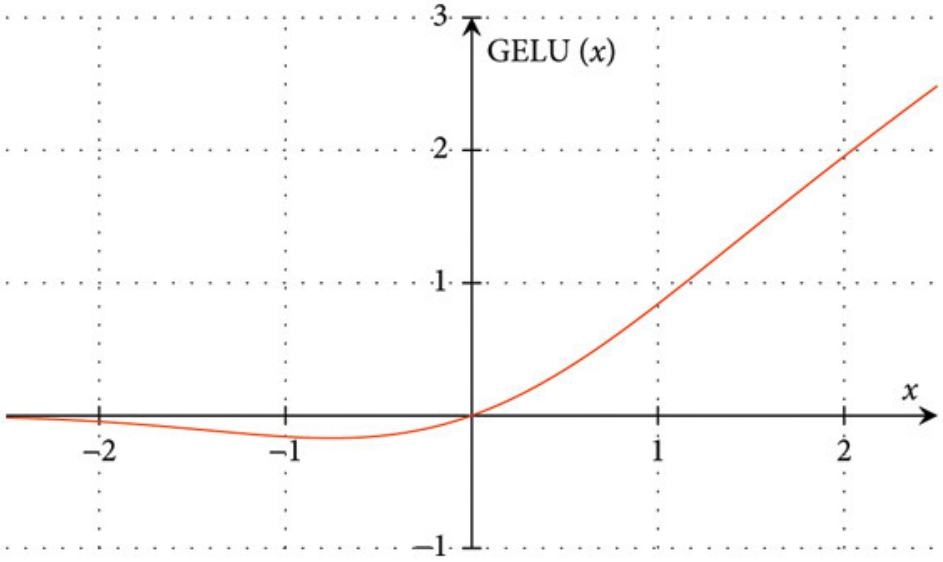
A sample neural network



- 4 convolutional layers
- 2 linear layers
- Average pooling
- GELU
- Batch normalization

1,758,731 trainable parameters
85% accuracy on test set

GELU activation function




```
Y = cdf(x) * x
```

Easy to model with GAMSPy

Convolutional layers -> **gp.formulations.Conv2d**

Linear layers -> **gp.formulations.Linear**

AvgPooling -> **gp.formulations.AvgPool2d**

Batch normalization and GELU missing:

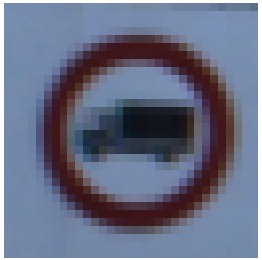
- But easy to implement with existing features!
- cdf(x) = errorf(x)

Applying Example

GAMS
MODEL – SOLVE – DEPLOY

model.py

```python
opt_model = gp.Model(
    m,
    equations=m.getEquations(),
    problem="nlp",
    objective=out[0, 16] - out[0, 7],
    sense="min",
)

opt_model.solve(output=sys.stdout, solver="conopt")
```



Class 16

Class 7

Trick nn in this direction

Solve statistics

158,508 constraints

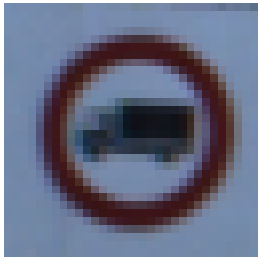
155,436 variables

15,969,198 Jacobian elements

62,016 of which are nonlinear

Local optimal solution found in ~1 min w/ CONOPT

No feasible solution found in 9 hours by a global solver



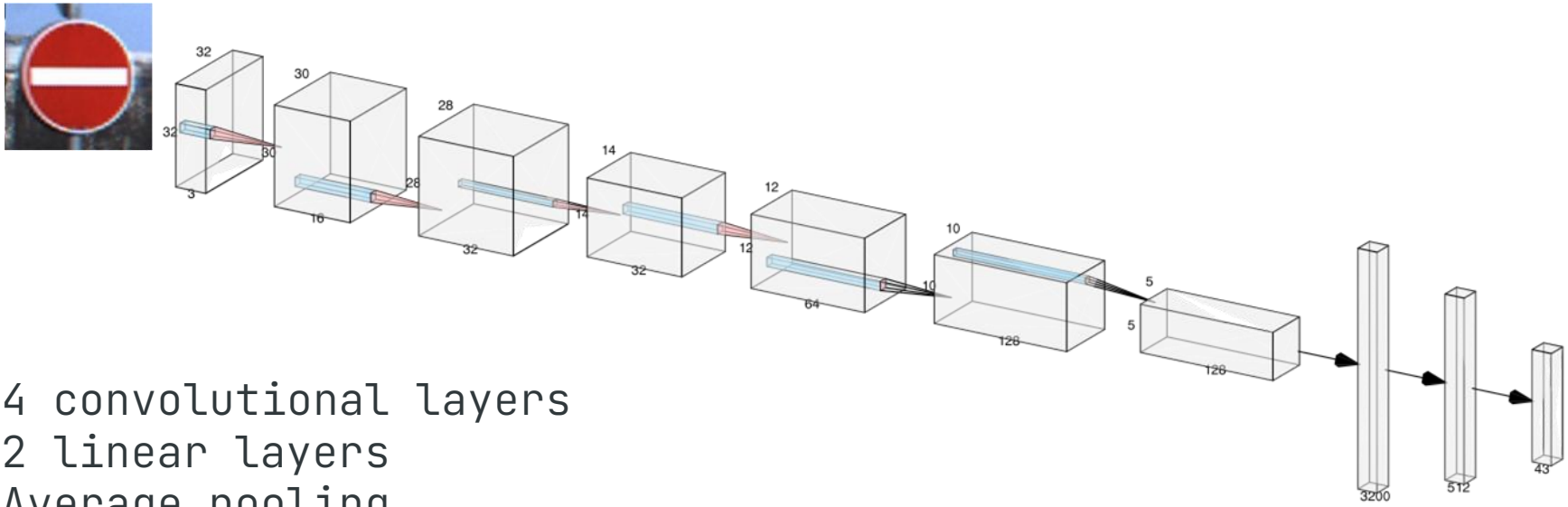
99.8%

No vehicles over 3.5 t permitted

99%

Speed limit 100 km/h

A sample neural network

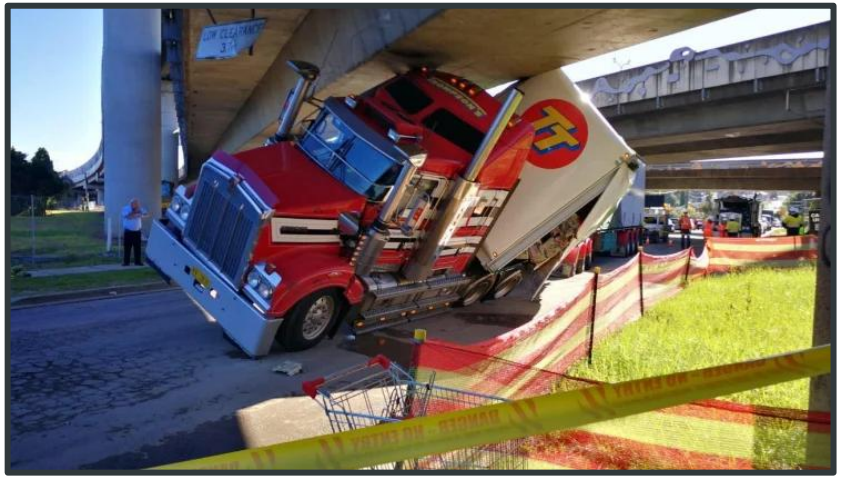


4 convolutional layers
2 linear layers
Average pooling
GELU → ReLU
Batch normalization

1,758,731 trainable parameters
88% accuracy on test set

Solve statistics

217,451 constraints
282,539 variables
16,279,275 Non-zeros
62,016 binary variables



No primal solution found in 1 hour

Summary

GAMS
MODEL – SOLVE – DEPLOY

- GAMSPy leverages GAMS backend to generate and solve models
- State-of-the-art optimization solvers
- Unique and streamlined way to complete pre/post-processing and visualization – all in a single environment
- Seamless integration with GAMS MIRO, GAMS Engine, and NEOS (local machines vs. cloud/AWS machines)
- Installed with one line:

```
pip install gamspy
```


GAMS
MODEL – SOLVE – DEPLOY