GRIFFINLab

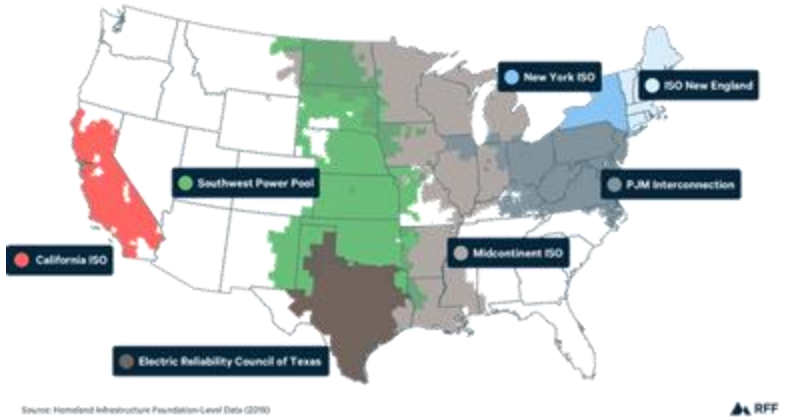
Learning-Enhanced Optimization for Power Systems

Dr. Kyri Baker

University of Colorado Boulder

Optimization governs the operation of many grids

- The “Optimal Power Flow” problem is solved to match supply and demand in adherence with many complex constraints
  - This problem is so challenging that it is often broken up into stages: solve for integer variables day-ahead, use linear approximations of grid physics…
- But it is so important that the U.S. government spent nearly $10M running a competition for people to solve the full-blown mixed integer nonlinear problem (the winner of said competition is among us and works for Gurobi).



Source: Resources for the Future

Why do we want to solve OPF better?

- By avoiding approximations, we can **lower carbon emissions**, **cost**, and **improve reliability**

Metric	AC OPF	DC OPF w AC
Emissions (Tons of CO2e)	11,490.61	16,005.74
Cost	$21,196,158.05	$21,487,150.71
Voltage Violations (#)	0	181
Computation Time (s)	2.25	0.25

24 hour simulation w/15-min timesteps, 118-bus system

→ **28% decrease in emissions** when it is assumed natural gas generators are the slack generators (the power plant which adjusts its output to account for AC-related violations).

→ **1.4% decrease in cost** (consistent with previous literature)

→ **No voltage violations** (improving grid reliability)

C. Winner, J. Garland, C. Crozier, K. Baker, "Carbon Emissions Resulting from Different Power Flow Models for Dispatch," IEEE PESGM 2023.
T. Overbye, X. Cheng, Y. Sun, "A comparison of the AC and DC Power Flow Models for LMP calculations," HICSS, 2004.

But true AC Optimal Power Flow is *hard.*

(NP-hard in fact)

- Even if you can solve it, it might not be fast enough for real-time operation
- You may have issues finding a feasible solution (so it is not uncommon for grid operators to transform the constraints into soft constraints in the objective)

AI to the rescue! ...maybe?

- Zamzam and Baker, “Learning Optimal Solutions for Extremely Fast AC Optimal Power Flow”, 2019
- Fioretto, Mak, and Van Hentenryck, “Predicting AC Optimal Power Flows: Combining Deep Learning and Lagrangian Dual Methods,” 2019
- Donti, Rolnick, and Kolter, “DC3: A learning method for optimization with hard constraints,” 2021
- Piloto et. al (Google DeepMind), “CANOS: A Fast and Scalable Neural AC-OPF Solver Robust To N-1 Perturbations,” 2024

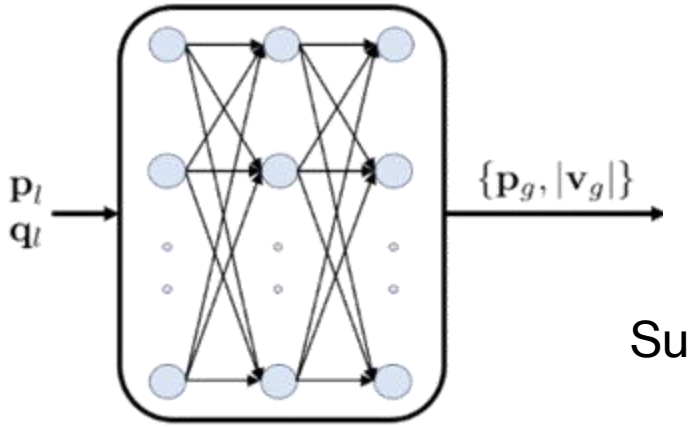
Solving this problem quickly, using machine learning, has inspired many papers

(Brief background on learning-for-OPF)

Regression-based models

Inputs

(e.g. current loads)



Outputs

(optimal generation)

Super fast and generally close to optimal, but no guarantee on constraint satisfaction!

Models with Implicit Layers (e.g. Donti et. al, 2021) – these neural network layers enforce input/output relationships

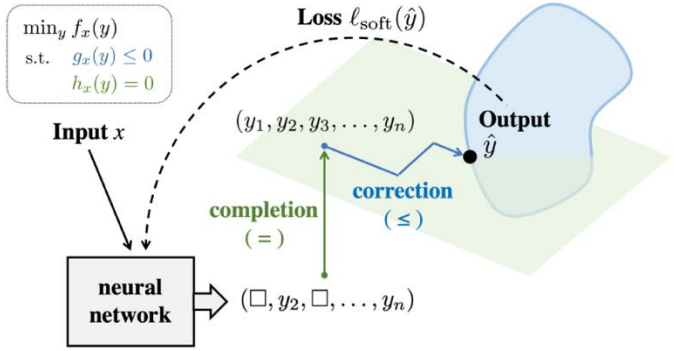


Figure 1: A schematic of the DC3 framework.

Better constraint satisfaction guarantees, but no optimality guarantees, and will still produce some kind of solution even with infeasible inputs

What's wrong with training a neural net to learn to produce solutions to OPF?

- Nothing…if you can do it right and if you can convince grid operators it's trustworthy.

Challenge #1: Grid operators trust the optimization tools they have been using for years.

If there was a way we could integrate learning in with these tools and improve the process of finding AC OPF solutions, but still have guarantees on constraint satisfaction and optimality (local), that would be ideal

Challenge #2: These models *will* produce an output for you, *even if the inputs are infeasible*

OPF infeasibilities **do** happen in real systems, and operators must take action (like enacting blackouts, for example)

We need to make sure these models aren't going to spit out a solution when there actually *isn't* one

Challenge 1: Integrating AI in with existing solvers

Challenge #1: Grid operators trust the optimization tools they have been using for years.

Potential solutions:

- *Learning* better warm starts/initial points then using a conventional solver
  - AC OPF is very sensitive to initial points
- *Learning* better algorithms for fundamental optimization operations and changing algorithms in the solver
  - Google DeepMind's AlphaEvolve discovered a new matrix multiplication algorithm that requires fewer multiplications than the current state of the art

More? (Discussion!)

Better Warm-Start Points in AC OPF

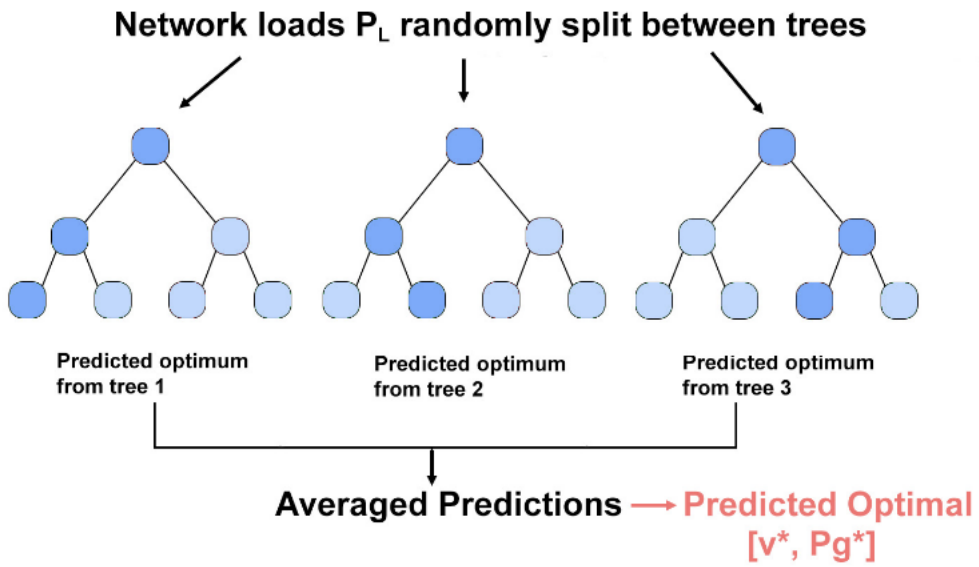


Table 4. Total time to solve ACOPF on the testing dataset

Network	Tot. Time (s) Learned	Tot. Time (s) DC	Tot. Time (s) Flat
14-bus	10.72	17.29	14.06
57-bus	15.18	24.76	20.58
118-bus	25.46	33.49	32.51
300-bus	56.68	67.82	69.02

Even without deep learning (using a random forest), I was able to get decent speedups by learning better warm-starts points for an AC OPF solver

Sloppy intermediate iterations

Newton-Raphson based solvers are prevalent when solving AC OPF

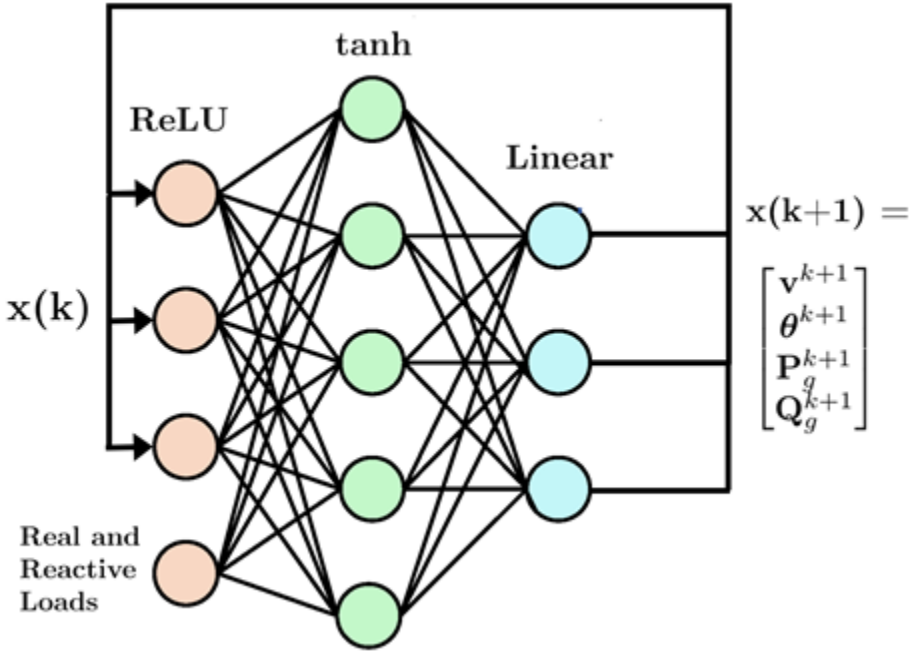
But you don't need exact Jacobian matrices to get close to the optimal solution – you often just need to move in the right direction for a while until you get close (quasi-Newton methods where you keep the Jacobian constant, for example!)

A NN can learn intermediate iterations and perform them quickly

Generate training data by storing intermediate iterations

Input: Current candidate iteration x(k)

Output: Next candidate iteration x(k+1)

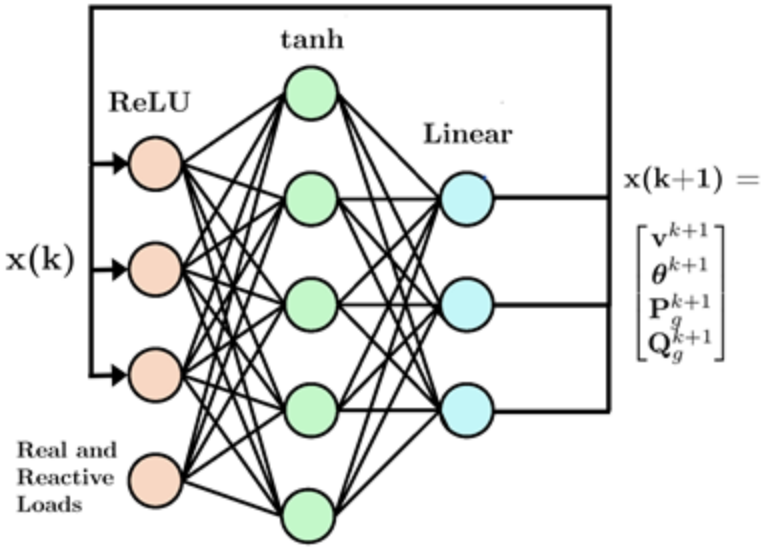


Sloppy intermediate iterations

Both Newton-Raphson and Learning-Boosted Newton Raphson are fixed point iterations:

$$\mathbf{x}^{k+1} = F(\mathbf{x}^k)$$

But the learning-boosted method has an easier to evaluate *F()*



It may be easier to learn what direction to move in than to directly learn an optimal solution from an initial point

K. Baker, "A Learning-boosted Quasi-Newton Method for AC Optimal Power Flow," https://arxiv.org/abs/2007.06074, Workshop on machine learning for engineering modeling, simulation and design @ NeurIPS 2020.

How much faster is it?

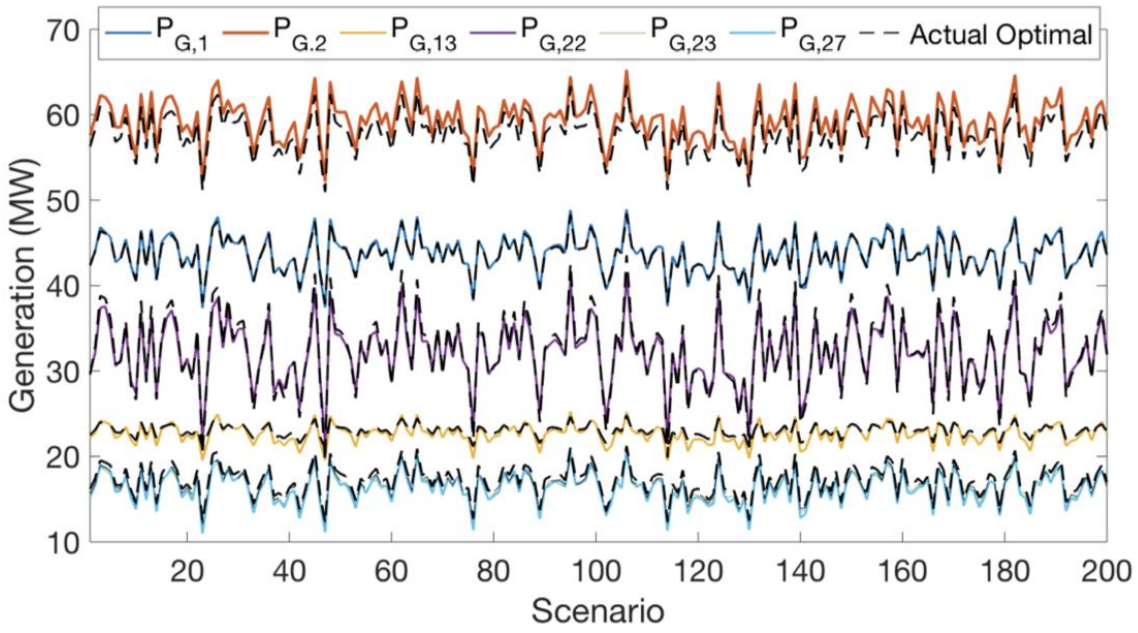


Fig. 2. Predicted generation values for 200 test scenarios (colors) and actual optimal values (black dashed line) for the IEEE 30-bus system.

Granted, this is for the MATPOWER Interior Point Solver, not Gurobi (which is much faster than MIPS).

MEAN ABSOLUTE ERROR ACROSS TESTING DATASET

Case	MAE: Voltage Magnitude (pu)	MAE: Active Power (MW)	MAPE: Cost (%)
30-bus	0.004 pu	0.64 MW	0.29%
300-bus	0.009 pu	10.47 MW	0.65%
500-bus	0.099 pu	0.62 MW	0.66%
1,354-bus	0.019 pu	7.55 MW	1.16%

TABLE IV
TIME TO CONVERGENCE FOR EACH NETWORK.

Case	Mean Time (s)	Max Time (s)	Variance in Time (s)	Mean Speedup
30-bus MIPS	0.04	0.41	4.52e-04	
30-bus NN	0.06	0.42	3.53e-04	**-0.66x**
300-bus MIPS	1.09	10.87	2.09	
300-bus NN	0.03	0.42	0.001	**36.3x**
500-bus MIPS	1.46	2.96	0.49	
500-bus NN	0.08	0.45	3.25e-4	**18.3x**
1,354-bus MIPS	7.64	19.89	15.55	
1,354-bus NN	0.34	0.69	0.0016	**22.5x**

What if we take this all the way?

And keep iterating with the neural net until convergence?

Takes more iterations to converge because each iteration is using an approximate direction, but each iteration is faster

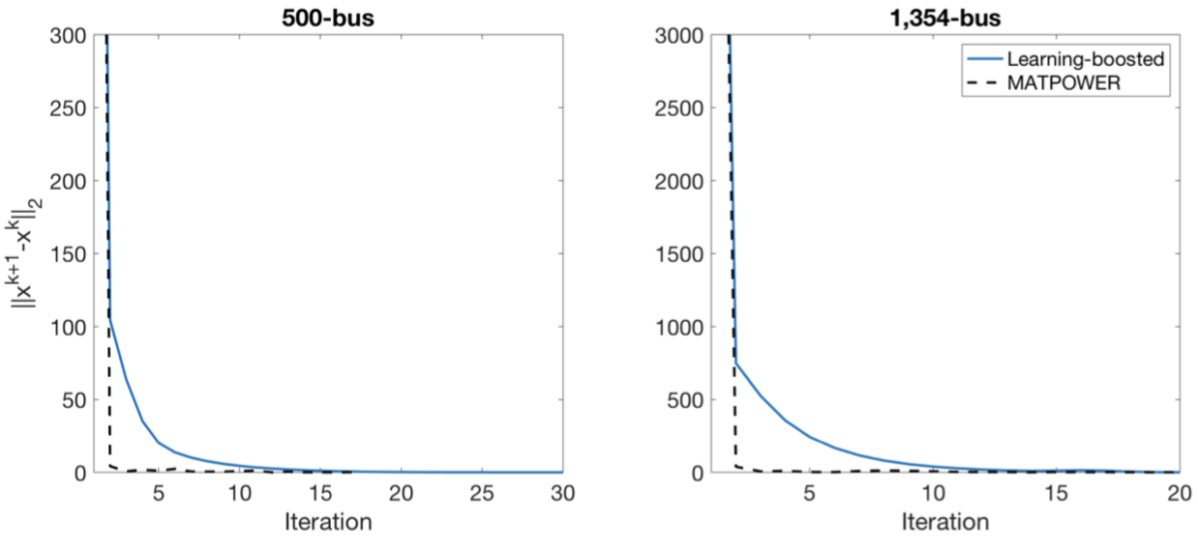
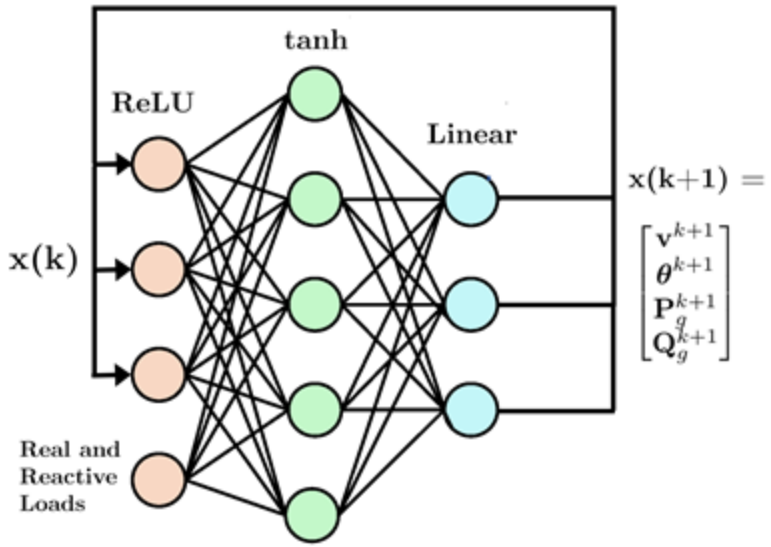


Fig. 3. Norm of $\mathbf{x}^{k+1} - \mathbf{x}^k$ for two scenarios in the 500-bus (left) and 1,354-bus (right) systems.

TABLE IV
TIME TO CONVERGENCE FOR EACH NETWORK.

Case	Mean Time (s)	Max Time (s)	Variance in Time (s)	Mean Speedup
30-bus MIPS	0.04	0.41	4.52e-04	
30-bus NN	0.06	0.42	3.53e-04	**-0.66x**
300-bus MIPS	1.09	10.87	2.09	
300-bus NN	0.03	0.42	0.001	**36.3x**
500-bus MIPS	1.46	2.96	0.49	
500-bus NN	0.08	0.45	3.25e-4	**18.3x**
1,354-bus MIPS	7.64	19.89	15.55	
1,354-bus NN	0.34	0.69	0.0016	**22.5x**

What about constraint satisfaction?



Subset of predicted variables

Power flow solver

Feasible and (generally) close to optimal solution

We can use a traditional equation solver at the end (not an optimization problem!) to ensure equality constraints are satisfied

Essentially we've learned a warm-start in a sense...

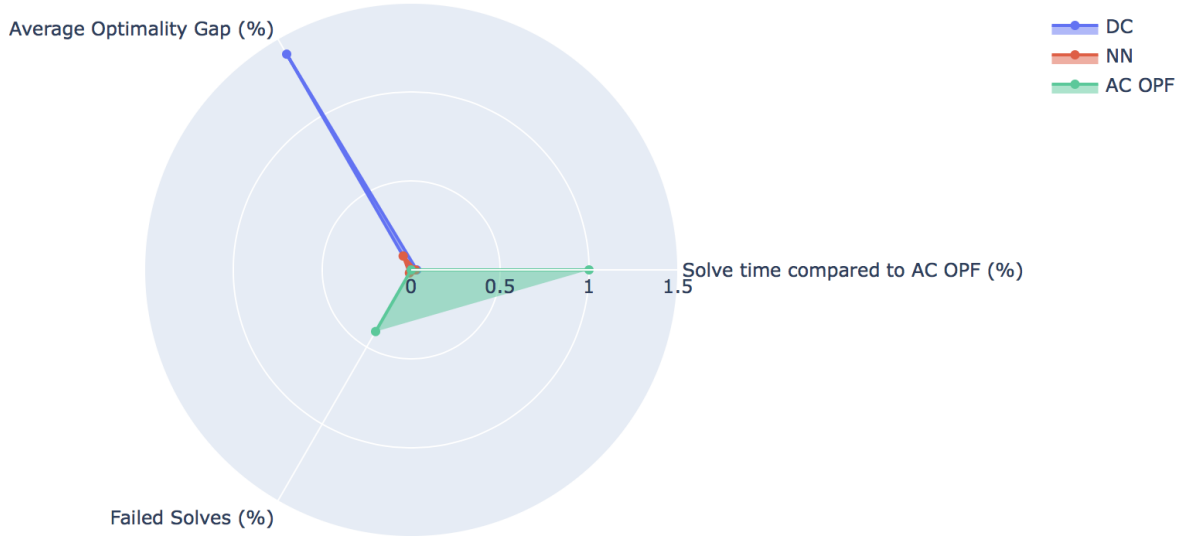
What about optimality gap?

Network	Average Gap: NN	Average Gap: DC	Worst Gap: NN	Worst Gap: DC
30-bus	0.20%	1.46%	0.31%	1.82%
500-bus	2.70%	5.30%	10.12%	16.22%
1354-bus	0.09%	1.40%	0.20%	1.73%

(across 500 test loading scenarios)

DC OPF is often used to determine market clearing prices and LMPs. This is an “acceptable” optimality gap.

We get speed and better optimality than currently used linear approximations



Challenge 2: Feasibility

Challenge #2: These models *will* produce an output for you, *even if the inputs are infeasible*

OPF infeasibilities **do** happen in real systems, and operators must take action (like enacting blackouts, for example)

We need to make sure these models aren't going to spit out a solution when there actually *isn't* one

From talking to grid operators, they do have challenges with infeasibility, and thus often include constraints in the objective function (as soft constraints)

The ARPA-E Grid Optimization Competition mentioned before also had soft constraints for many constraints due to feasibility challenges

Use slack variables?

If we don't know where the infeasibility is coming from, we have to add a variable for every constraint in the problem, and more constraints

Gurobi's feasrelax and other feasibility "fixing" techniques often use slack variables.

Even limiting the number of these variables can add significant model complexity

$$\min_{\boldsymbol{x}} \; f(\boldsymbol{x}) \qquad h(\boldsymbol{x}) \leq \mathbf{0} \quad \Rightarrow \quad \min_{\boldsymbol{x},\boldsymbol{s}} \; f(\boldsymbol{x}) + g(\boldsymbol{s}) \qquad h(\boldsymbol{x}) \leq \boldsymbol{s} \qquad \mathrm{s} \geq \mathbf{0}$$

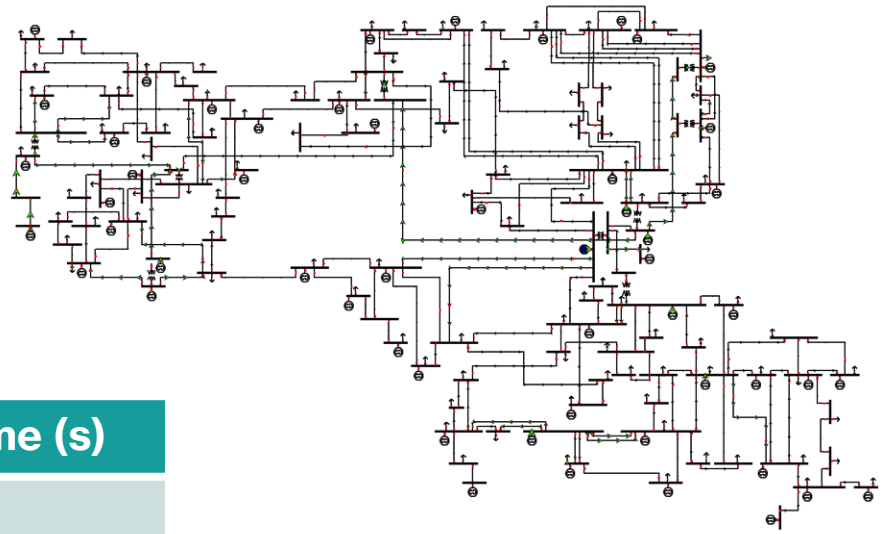
Also, these techniques usually only yield **one** set of slack variables – or one option for making the system feasible
Sometimes this can result in equity issues in power systems (the same customers getting power cuts!)

Can we learn which constraints to add slack variables to?

→ Our current work in progress

→ Sometimes adding slack variables can actually *speed up* solution time, as the feasible region is increased

For example, some networks (the 118-bus system),
can actually solve *faster* with *certain* slack variables added
- even if the problem is feasible!



	Mean Solve Time (s)	Min Solve Time (s)	Max Solve Time (s)
No Slack	1.1129	0.1031	1.9234
Slack	0.989	0.1774	4.4593

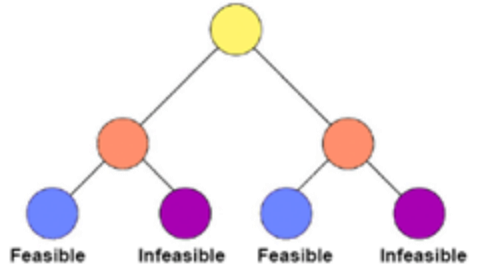
Can we *learn* when a problem is infeasible ahead of time without attempting to solve it?

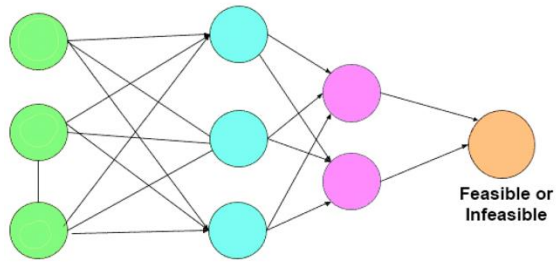
→ Depending on the solver, sometimes it can take a very long time to iterate to try to find a solution – even if the problem is actually infeasible

→ AI can help us learn to detect infeasible problems ahead of time

→ Outside of obvious cases of course, like the total load exceeds total generation capacity

Feasibility Detection (Binary Classification)
Decision Tree or FFNN

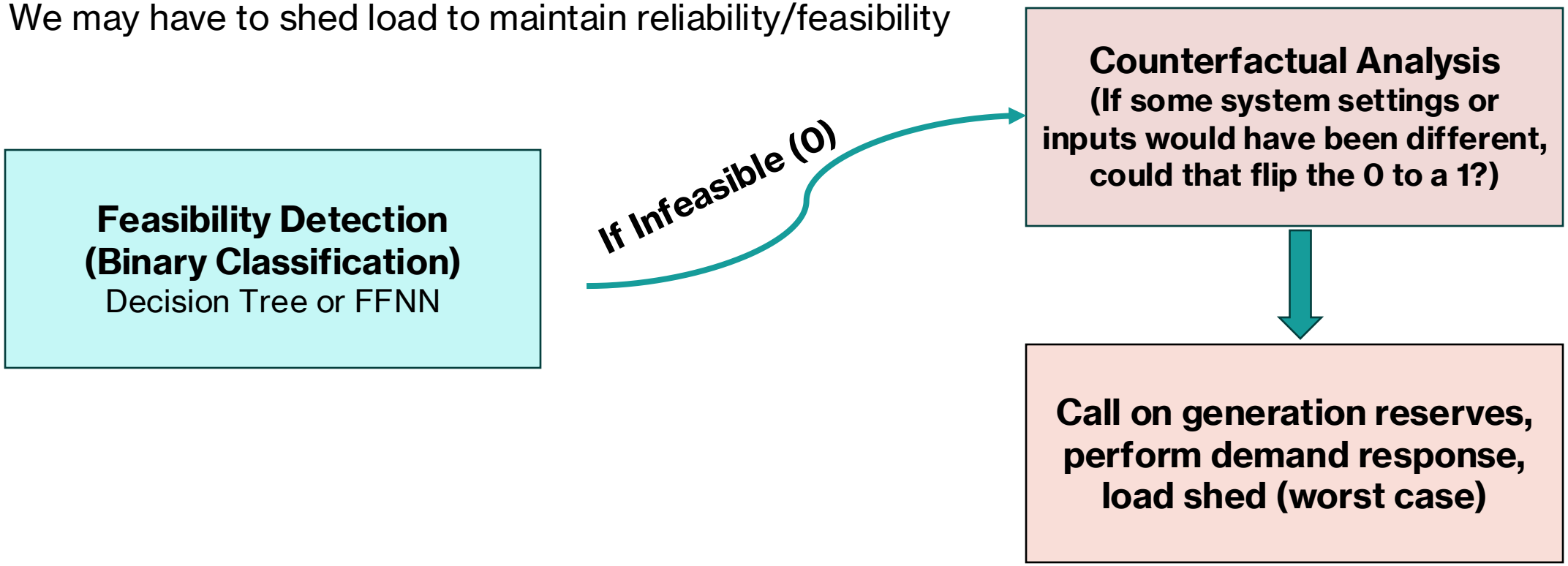




If we know it's infeasible, can we have options for how to fix it?

→ Sometimes in power systems, infeasibility actually means we don't have enough generation capacity
→ Or lines or transformers are overloaded

→ We may have to shed load to maintain reliability/feasibility



A suite of options that can push us to feasibility

Isn’t there an “optimal”?

$$\min_{\boldsymbol{x},\boldsymbol{s}} \; f(\boldsymbol{x}) + g(\boldsymbol{s})$$
$$h(\boldsymbol{x}) \leq \boldsymbol{s}$$
$$\mathrm{s} \geq \mathbf{0}$$

Isn’t the ‘optimal’ decision the one which minimizes the amount of slack added to the original problem?

Not necessarily.

Isn’t there an “optimal”?

$$\min_{\boldsymbol{x},\boldsymbol{s}} \; f(\boldsymbol{x}) + g(\boldsymbol{s})$$

$$h(\boldsymbol{x}) \leq \boldsymbol{s}$$

$$\boldsymbol{s} \geq \boldsymbol{0}$$

Isn’t the ‘optimal’ decision the one which minimizes the amount of slack added to the original problem?

Not necessarily.

The “optimal” way to load shed, for example, may hurt more communities more than others

Mathematical optimality ≠ Social optimality

NATIONAL

California power outages highlight economic disparity

by: CATHY BUSSEWITZ, Associated Press
Posted: Oct 13, 2019 / 12:19 PM EDT
Updated: Oct 13, 2019 / 12:19 PM EDT

This Sunday, Oct. 6, 2
brother Colton Awalt
Read More

SHARE

This is an archived article and the information in the article may be outdated. Please look at the time stamp on the st
to see when it was last updated.

When the nation’s largest utility warned customers that it would cut power to nearly 2 million people across Northern California, many rushed out to buy portable generators, knowing the investment could help sustain them during blackouts.

(Also, for large systems, adding hundreds or thousands of new variables/constraints is not ideal!)

First step: DC OPF – what most operators currently use

Infeasibilities here are due to **line congestion**
Or power lines reaching their thermal limits

Can also have infeasibilities due to too much load in the system – but this is trivial to predict (just check if $\sum load > \sum gen$)

Cost function: Minimize cost of power generation

$$\min_{p_g} \sum_{j \in \mathcal{G}} a_j p_{g,j}^2 + b_j p_{g,j} + c_j$$

Set of buses

Power balance at each bus

$$\text{s.t}: p_{l,i} - \sum_{k \in \mathcal{G}_i} p_{g,k} = \sum_{m \in \mathcal{N}} B_{im}\theta_{im}, \quad \forall i \in \mathcal{N}$$

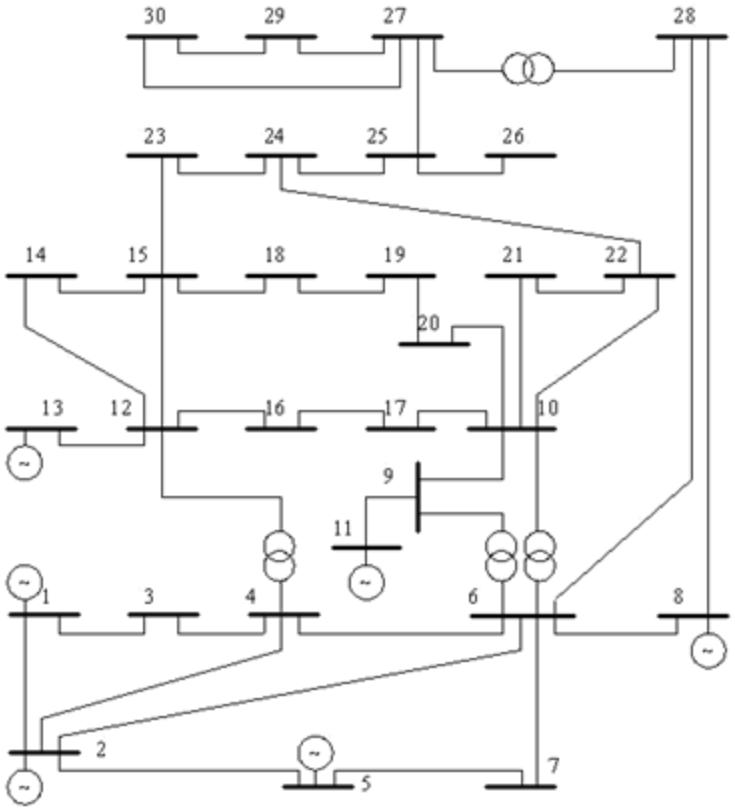
$$-F_{im} \leq B_{im}\theta_{im} \leq F_{im}, \quad \forall im \in \mathcal{L}$$

Line flow limits

Set of lines

+ generator limits
+ angle constraints

Dataset Creation



30 and 300-bus systems

10,000 load scenarios

Input Data: {load, feasible_flag}

50/50 Class Balance

80/20 Train/Test split

This data is used to train the classifiers

The counterfactual model can be modeled as an unconstrained optimization or another ML model

The Counterfactual

What modifications to the load vector will result in the classifier prediction changing from a 0 to a 1?

Set of *k* counterfactuals (load/gen modification vectors)

$$C(\boldsymbol{x}) = \arg \min_{\boldsymbol{c}_1,\ldots,\boldsymbol{c}_k} \Big(\frac{1}{k} \sum_{i=1}^{k} \text{yloss}(f(\boldsymbol{c}_i), y) + \frac{\lambda_1}{k} \sum_{i=1}^{k} \text{dist}(\boldsymbol{c}_i, \boldsymbol{x}) - \lambda_2 \, \text{dpp\_diversity}(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k) \Big)$$

We can generate a *set* of possible counterfactuals that an operator can choose from

(should I shed load at bus 34? Should I bring online additional generation at bus 12?)

The Counterfactual

What modifications to the load vector will result in the classifier prediction changing from a 0 to a 1?

Set of *k* counterfactuals (load/gen modification vectors)

$$C(\boldsymbol{x}) = \arg \min_{\boldsymbol{c}_1,\ldots,\boldsymbol{c}_k} \Big(\frac{1}{k} \sum_{i=1}^{k} \text{yloss}(f(\boldsymbol{c}_i), y) + \frac{\lambda_1}{k} \sum_{i=1}^{k} \text{dist}(\boldsymbol{c}_i, \boldsymbol{x}) - \lambda_2 \, \text{dpp\_diversity}(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k) \Big)$$

The loss function here describes how effectively the counterfactuals change the classification from a 0 to a 1

f is a differentiable model (e.g. our classifier)

The Counterfactual

What modifications to the load vector will result in the classifier prediction changing from a 0 to a 1?

Set of *k* counterfactuals (load/gen modification vectors)

$$C(\boldsymbol{x}) = \arg \min_{\boldsymbol{c}_1,\ldots,\boldsymbol{c}_k} \Big(\frac{1}{k} \sum_{i=1}^{k} \text{yloss}(f(\boldsymbol{c}_i), y) + \frac{\lambda_1}{k} \sum_{i=1}^{k} \text{dist}(\boldsymbol{c}_i, \boldsymbol{x}) - \lambda_2 \, \text{dpp\_diversity}(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k) \Big)$$

A distance metric that penalizes too much change from our original feature vector ***x***

We don't want to change the system more than we have to!

The Counterfactual

What modifications to the load vector will result in the classifier prediction changing from a 0 to a 1?

Set of *k* counterfactuals (load/gen modification vectors)

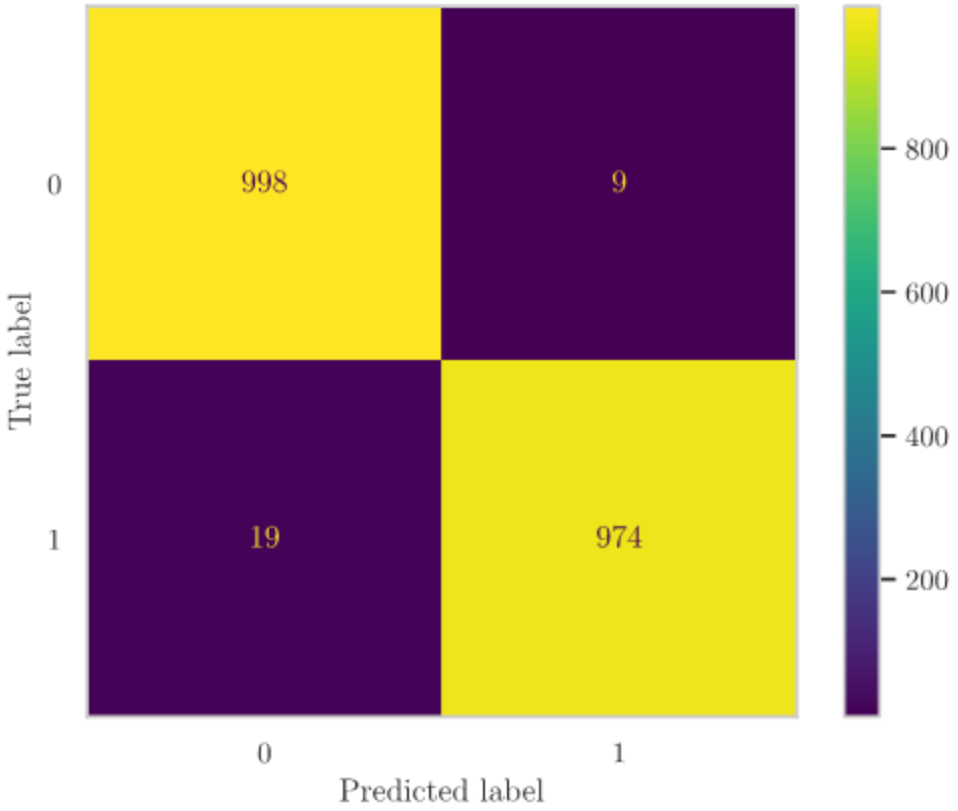
$$C(\boldsymbol{x}) = \arg\min_{\boldsymbol{c}_1,\ldots,\boldsymbol{c}_k} \Big(\frac{1}{k}\sum_{i=1}^{k} \text{yloss}(f(\boldsymbol{c}_i), y) + \frac{\lambda_1}{k}\sum_{i=1}^{k} \text{dist}(\boldsymbol{c}_i, \boldsymbol{x}) - \lambda_2 \,\text{dpp\_diversity}(\boldsymbol{c}_1, \ldots, \boldsymbol{c}_k) \Big)$$

A diversity metric that promotes a wider *variety* of counterfactuals to yield diverse options (rather than a bunch of counterfactuals that are slight perturbations of each other)

$$\text{dpp\_diversity} = \det(\boldsymbol{K}) \quad \text{where} \quad \boldsymbol{K}_{i,j} = \frac{1}{1 + \text{dist}(\boldsymbol{c}_i, \boldsymbol{c}_j)}$$

Classification Accuracy

Confusion matrix for (30-bus) decision tree:



0 = Infeasible, 1 = Feasible

Method	IEEE 30-bus	IEEE 300-bus
FFNN	99.15%	88.05%
DT	99.0%	84.0%

Decision tree is actually pretty good!

Accuracy decreases as system complexity increases

Accuracy could be improved even further by more rigorous hyperparameter tuning

Feasibility restoration

After a problem is classified as infeasible, how well can we change power injections to restore feasibility?

Accuracy at Restoring Feasibility

Method	IEEE 30-bus	IEEE 300-bus
CF_{FFNN}	99%	100%
CF_{DT}	98%	100%
Gurobi	100%	100%

"Ground-truth" solution which uses slack variables to guarantee feasibility

Speed to Restore Feasibility

Method	IEEE 30-bus	IEEE 300-bus
CF_{FFNN}	4.14 ± 1.55	41.0 ± 7.23
CF_{DT}	3.54 ± 1.86	5.33 ± 2.41
Gurobi	41.4 ± 19.0	234 ± 15.1

(in milliseconds; for 1 CF generation).

ML-based approach is significantly faster at restoring feasibility

80 – 98% speedups

A suite of fixes

Adding slack variables to the problem directly *can* yield different fixes by changing the objective function to penalize the slack variables accordingly:

$$||x||_1 = \sum_{i=1}^{n} |x_i| ,$$

L-1 norm promotes sparsity
(give me a counterfactual that modifies as few loads as possible)

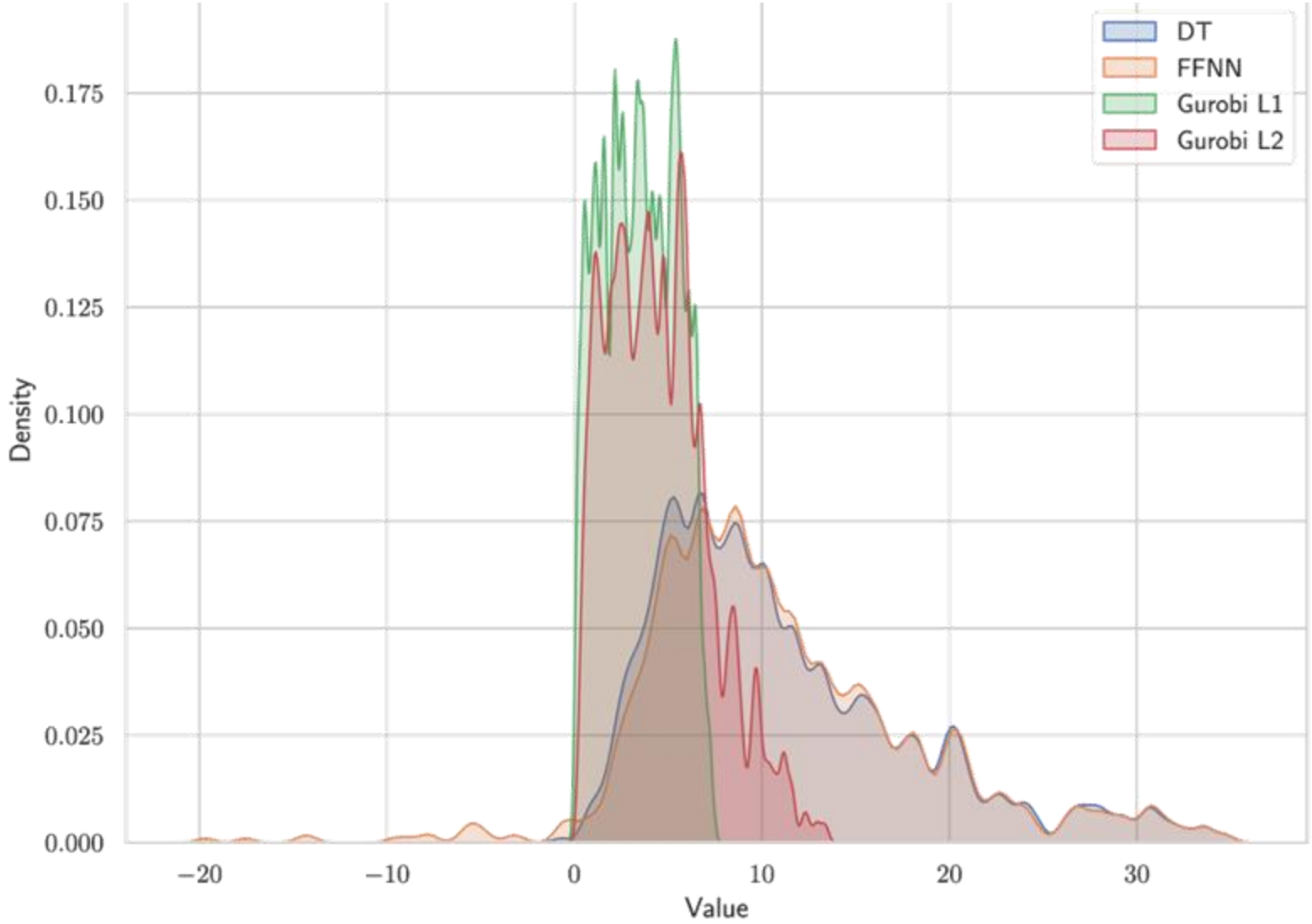
We'd ideally want an L-0 norm (not a real norm) which measures the number of nonzero components in a vector, but this is not continuous or differentiable

$$||x||_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$$

L-2 norm promotes smallest change
(give me a counterfactual that is as close to the original solution as possible in terms of Euclidean distance)

A suite of fixes

Distribution of bus-level load adjustments (positive = load decrease; negative = increase)



Example scenario

	0	1	2	3	4	5	6	7	8	9	Total load reduction or gen increase
CF Option 1	-	-	-	-	27.630195	-	-	3.7318735	-	-	**31.36 MW**
CF Option 2	-	-	-	-	41.550417	-	-	-	-	-	**41.55 MW**
CF Option 3	-	-	-	-	17.093741	-	-	-	9.58467	-	**26.67 MW**

Hmm...option 3 results in the lowest change, but customers at bus 8 have experienced a lot of outages lately.

I could call on a lot of demand response for the industrial customer at bus 4, or use CF Option 1 and call on a smaller amount of DR for customer 4 and use some storage reserves at bus 7.

→ Any of these options will help keep the lights on!

Conclusions

- There are many ways that AI can be used to help solve optimization problems
- Warm starts, learning better algorithms, learning how to adjust input problems….
- It does *not* have to be a black box from start to finish

http://www.kyrib.com