DTU Summer School 2022

Advanced Optimization, Learning, and Game-Theoretic Models in Energy Systems

Trustworthy AI for Power Systems

Spyros Chatzivasileiadis
Associate Professor
Head of Section Power Systems

This work would not have been possible without the hard work of several people! Many thanks to…

Andreas Venzke

Lejla Halilbasic

Elea Prat

Jochen Stiasny

Sam Chevalier

Florian Thams

Georgios Misyris

Rahul Nellikkath

Ilgiz Murzakhanov

And to our collaborators:

Dan Molzahn, GeorgiaTech

Steven Low, Caltech
Guannan Qu, Caltech (now at CMU)

Machine learning: Why shall we apply it in power systems?

Machine learning: Why shall we apply it in power systems?

1. ML methods can handle well extremely complex systems
   - Sometimes impossible or very difficult to develop models based on first principles, e.g. weather forecasting, image processing, etc.

2. ML methods can infer from incomplete data
   - Given a limited set of training data, ML methods are known to generalize well, e.g. written text recognition

3. ML methods can be extremely fast
   - Milliseconds instead of seconds, minutes, or hours: Require a limited set of linear and simple non-linear computations instead of e.g. computationally intensive numerical integration methods

Machine learning: Why shall we apply it in power systems?

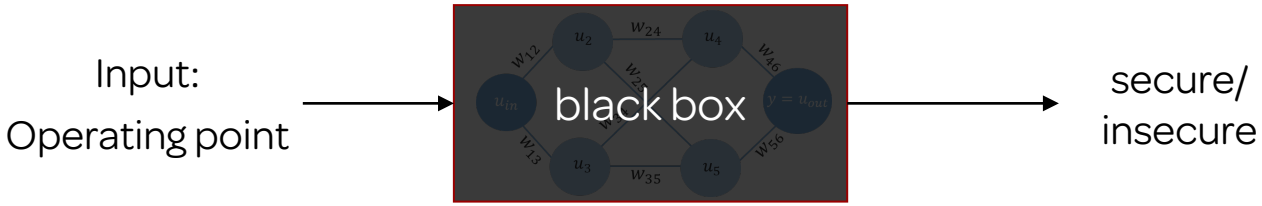
Machine Learning

1. ML methods can handle well extremely complex systems
2. ML methods can infer from incomplete data
3. ML methods can be extremely fast

Power Systems

1. Real-life power systems are described by thousands of variables, parameters, and differential-algebraic equations
2. It is computationally impossible (intractable) to check for all possible operating conditions
3. Build proxies (=surrogate models) → get an estimate 100-1'000 times faster than conventional models; assess 100-1'000 more scenarios in the same time

ML Barriers for Power systems



BUT:

1. Why would we use a "**black box**" to decide about **a safety-critical application**?

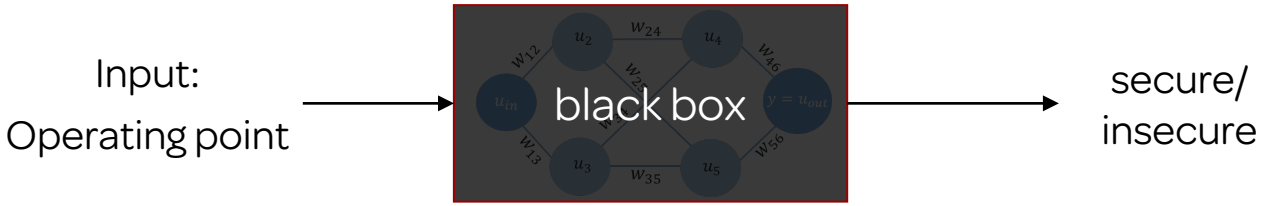
2. **Accuracy is** a **purely statistical** performance metric. Who guarantees that the Neural Network can handle well previously unseen operating points?

3. Why would we depend on **discrete and incomplete data**, when we have developed **detailed physical models** over the past 100 years?

Takeaway #1

Solid motivation is key: If you wish to apply machine learning (including deep learning) methods on any problem, develop **solid arguments** why this is the only or the best way to do it

ML Barriers for Power systems



ML Interpretability → understand the decisions of the ML Algorithm

1. Why would we use a "**black box**" to decide about **a safety-critical application**?

2. **Accuracy is** a **purely statistical** performance metric. Who guarantees that the Neural Network can handle well previously unseen operating points?

3. Why would we depend on **discrete and incomplete data**, when we have developed **detailed physical models** over the past 100 years?

→ **Neural Network verification**: guarantees for the NN performance!
→ *Remove dependence on the test database*

→ Physics-Informed Neural Networks
→ *Prior knowledge!*

Goals of this lecture

1. The importance of high-quality data
2. Sampling beyond statistics
3. Neural network verification
4. Physics-informed neural networks

---

5. If time permits (appendix):
   - capturing previously intractable constraints with Decision Trees and Neural Networks

Article without any equations ☺

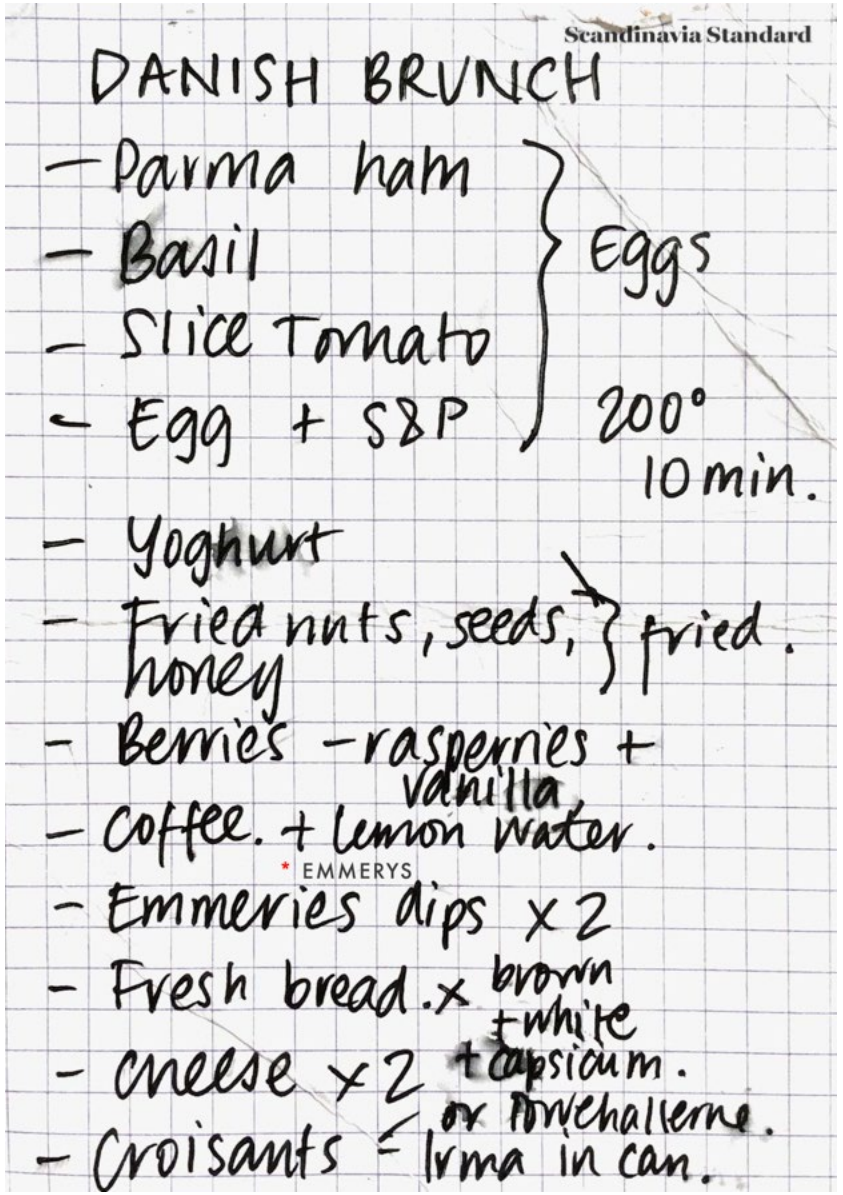
S. Chatzivasileiadis, A. Venzke, J. Stiasny and G. Misyris, **"Machine Learning in Power Systems: Is It Time to Trust It?,"** in *IEEE Power and Energy Magazine*, vol. 20, no. 3, pp. 32-41, May-June 2022 [.pdf]

Machine learning applications
(for power system security assessment)
A very short overview

The ingredients

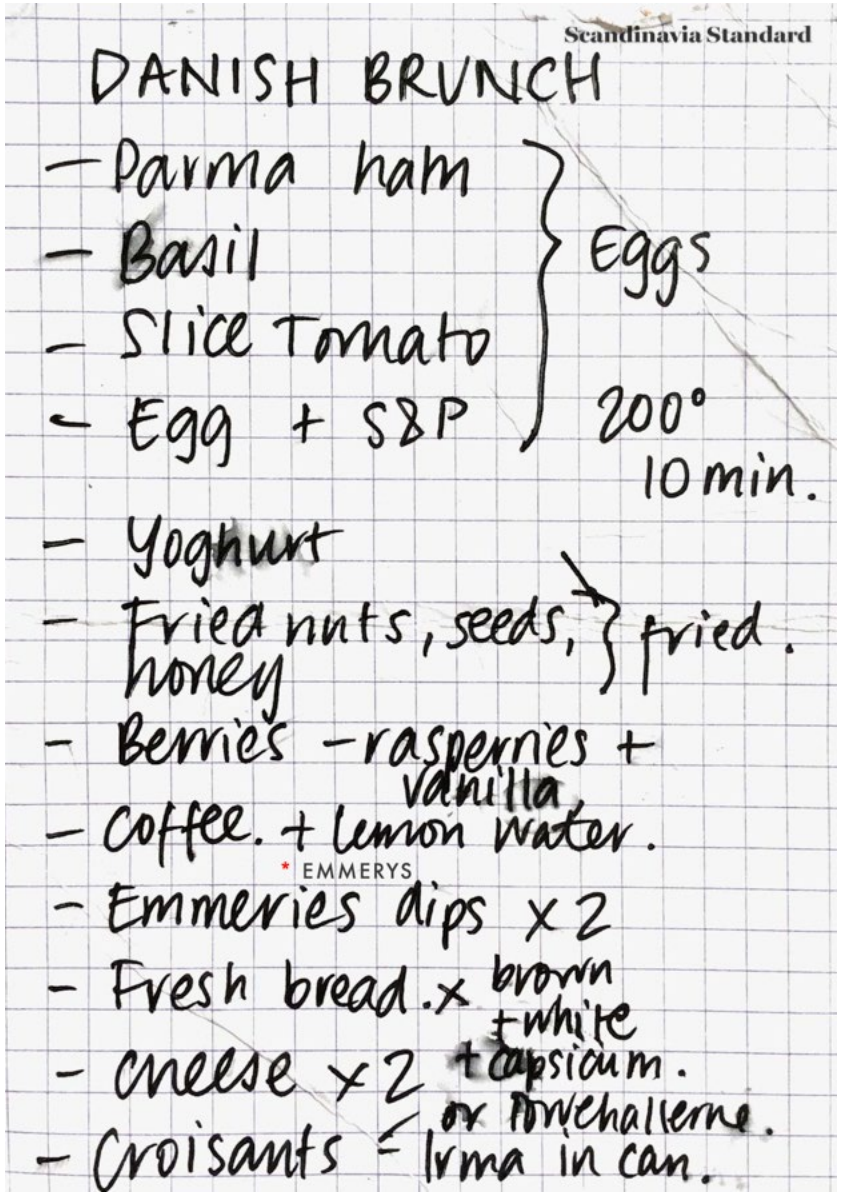
What do we need?

If we want to apply machine learning approaches for power system security



The ingredients

- A training database
- A training algorithm
- A test database
  - To test accuracy of the approach



Test Database

Test Database

Traditionally:

- Split training database to e.g. 80% training samples and 20% test samples
- Train with the 80%
- Test with the 20%

Modern toolboxes have this integrated and automatized → only need to provide a training database

Point to remember:

The test database determines the performance of your method. If the test data come from the same simulations as your training data, the accuracy can be deceivingly high. Would it be equally high in reality?

Ideally → use a different real-life dataset

(Unfortunately, not always possible)

Takeaway #2

The quality of your test database is crucial: the test database determines the performance of your method; for a valid assessment, it needs to include a wide range of operating conditions with the same frequency of occurrence as in real-life

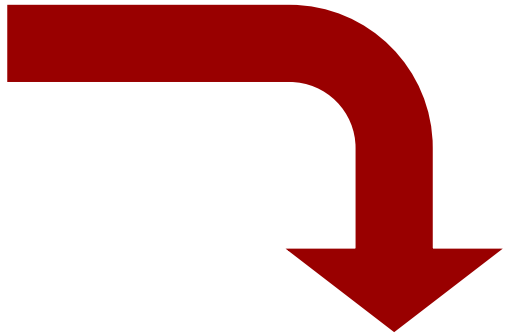
Evaluating the performance of a neural network

Evaluating the performance: Confusion matrix

		Target class (actual values)	
		1	0
Output class (predicted values)	1	True positive (TP)	False positive (FP)
	0	False negative (FN)	True Negative (TN)

Evaluating the performance: Confusion matrix

		Target class (actual values)	
		1	0
Output class (predicted values)	1	True positive (TP)	False positive (FP)
	0	False negative (FN)	True Negative (TN)



Classification for power system security

	Actually Safe	Actually Unsafe
Predicted safe	True positive (TP)	False positive (FP)
Predicted Unsafe	False negative (FN)	True Negative (TN)

Performance Metrics: Accuracy

	Actually Safe	Actually Unsafe
Predicted safe	True positive (TP)	False positive (FP)
Predicted Unsafe	False negative (FN)	True Negative (TN)

- Accuracy: The proportion of correct classifications in the whole dataset

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+TN+FN}$$

- **Example:** Assume 1000 datapoints
  - Actually safe: 500 TP=480 FP=30
  - Actually unsafe: 500 FN=20 TN=470

Accuracy = ?

Performance Metrics: Accuracy

	Actually Safe	Actually Unsafe
Predicted safe	True positive (TP)	False positive (FP)
Predicted Unsafe	False negative (FN)	True Negative (TN)

- Accuracy: The proportion of correct classifications in the whole dataset

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+TN+FN}$$

- **Example:** Assume 1000 datapoints
  - Actually safe: 500 TP=480 FP=30
  - Actually unsafe: 500 FN=20 TN=470

$$\text{Accuracy} = \frac{480+470}{480+20+470+30} = 95\%$$

Evaluating performance by measuring only accuracy is often not enough

Why?

Performance Metrics: Accuracy

	Actually Safe	Actually Unsafe
Predicted safe	True positive (TP)	False positive (FP)
Predicted Unsafe	False negative (FN)	True Negative (TN)

- Accuracy: The proportion of correct classifications in the whole dataset

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+TN+FN}$$

- **Example:** Assume 1000 datapoints
  - Actually safe: 500 TP=480 FP=30
  - Actually unsafe: 500 FN=20 TN=470

$$\text{Accuracy} = \frac{480+470}{480+20+470+30} = 95\%$$

Evaluating performance by measuring only accuracy is often not enough

Why?

- **Example:** Assume 1000 datapoints
  - Actually safe: 20 TP= 1 FP=30
  - Actually unsafe: 980 FN= 19 TN=950

Accuracy = ?

Performance Metrics: Accuracy

	Actually Safe	Actually Unsafe
Predicted safe	True positive (TP)	False positive (FP)
Predicted Unsafe	False negative (FN)	True Negative (TN)

- Accuracy: The proportion of correct classifications in the whole dataset

$$\text{Accuracy} = \frac{TP+TN}{TP+FP+TN+FN}$$

- **Example:** Assume 1000 datapoints
  - Actually safe: 500 TP=480 FP=30
  - Actually unsafe: 500 FN=20 TN=470

$$\text{Accuracy} = \frac{480+470}{480+20+470+30} = 95\%$$

Evaluating performance by measuring only accuracy is often not enough

Why?

- **Example:** Assume 1000 datapoints
  - Actually safe: 20 TP= 1 FP=30
  - Actually unsafe: 980 FN= 19 TN=950

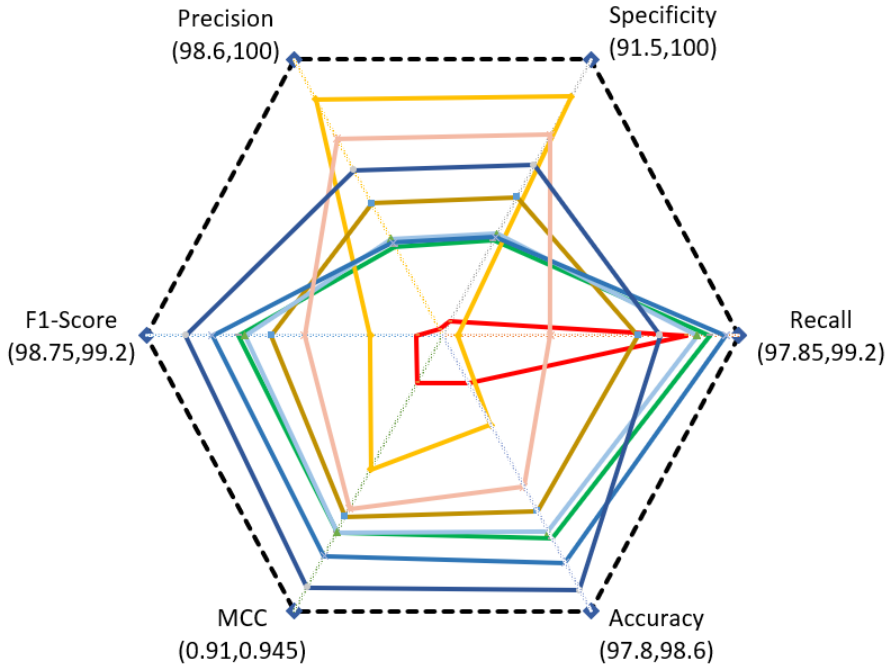
$$\text{Accuracy} = \frac{1+950}{1+19+950+30} = 95\%$$

- 95% accurate but we have misclassified almost all truly safe points!
- For heavily unbalanced data, accuracy is not sufficient!

Performance metrics

- Accuracy
- Recall: True Positive Rate $\frac{TP}{TP+FN}$
- Specificity: True Negative Rate $\frac{TN}{TN+FP}$
- Precision: Positive Predictive Value $\frac{TP}{TP+FP}$
- F1: harmonic mean of Precision and Recall $\mathrm{F1} = \frac{Precision \cdot Recall}{Precision+Recall}$
- MCC (Matthews correlation coefficient)

  (only for binary classification – 2 classes only)
  - MCC=1 → perfect prediction
  - MCC=0 → random (like flipping a coin)
  - MCC= –1→ Completely mistaken



Hidalgo-Arteaga, Hancharou, Thams, Chatzivasileiadis, Powertech 2019

$$\mathrm{MCC} = \frac{(TP \cdot TN) - (FP \cdot FN)}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$$

Key hints for your implementation

- Regularization: Training of neural networks work better if you normalize your inputs
  - Try to normalize your active power setpoints (e.g. if PG1 =30 MW and PG1max = 100 MW, then PG1=0.3)

- 1-hot encoding: Neural networks work better if you use *one vector for each class*

Instead of:

	Safe=1 Unsafe=0
x_1	0
x_2	1
x_3	0

	Safe	Unsafe
x_1	0	1
x_2	1	0
x_3	0	1

Takeaway #3:
Neural networks (or Decision Trees) for classification: you need a balanced training database → similar number of safe and unsafe points

Takeaway #4:
Accuracy is not sufficient to assess the NN/DT performance. We need additional metrics

Takeaway #5:
Neural Network training requires additional "tricks" to boost its performance (e.g. 1-hot encoding/regularization)

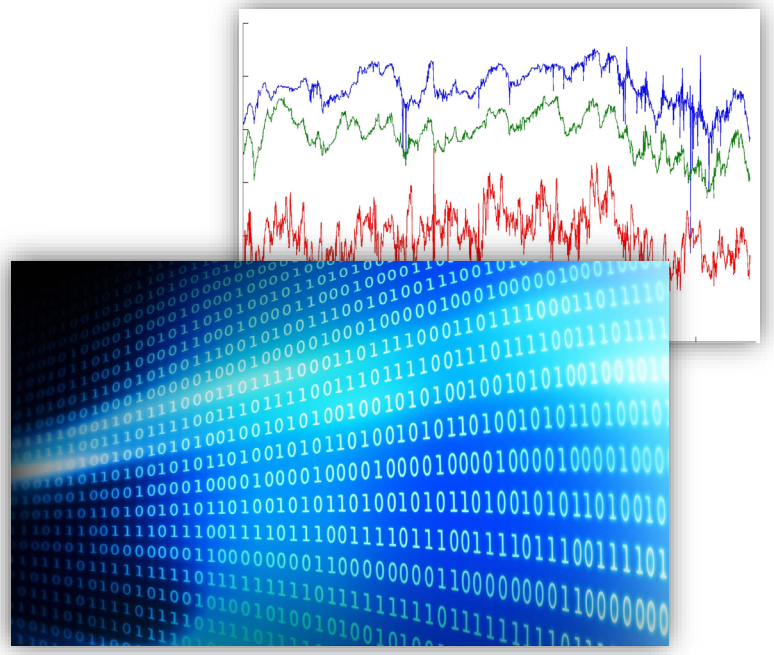
Training database

We need data!

1. Historical data are often insufficient

2. We often need to generate our own data to test the performance of our ML algorithm before deployment ("emulate")

3. **Need to generate simulation data**



Here: generate data for power system security assessment

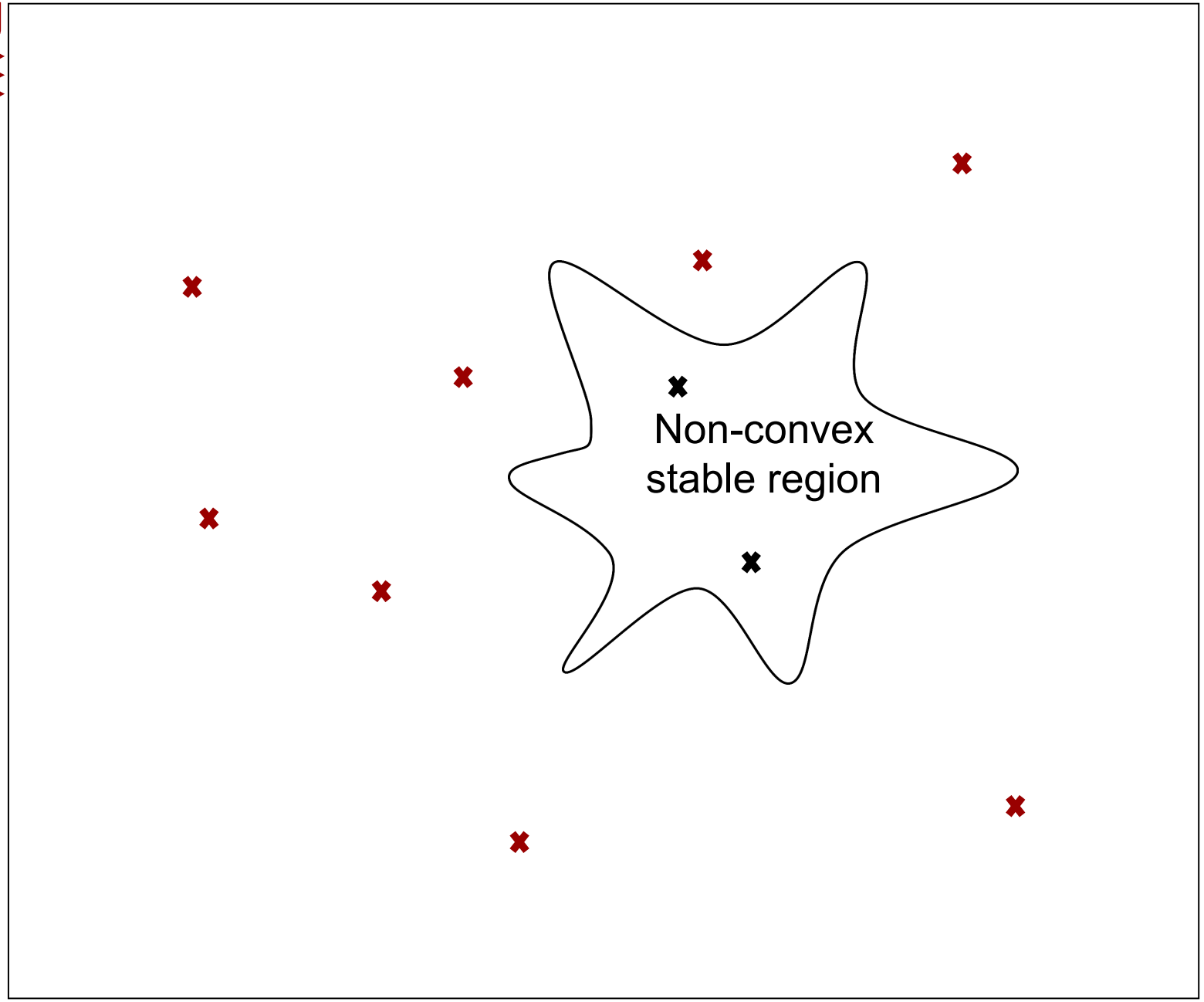
- Assessing the stability of 100'000s of operating points is an extremely demanding task
  - Immense search space
  - How can I do it efficiently?

Sampling beyond Statistics: Efficient Database Generation

- Modular and highly efficient algorithm
- Can accommodate numerous definitions of power system security (e.g. N-1, N-k, small-signal stability, voltage stability, transient stability, **or a combination** of them)
- **10-20 times faster** than existing state-of-the-art approaches
- Our use case: N-1 security + small-signal stability
- Generated Databases for IEEE 14-bus and NESTA 162-bus system available!

  http://www.chatziva.com/downloads.html#databases

F. Thams, A. Venzke, R. Eriksson, and S. Chatzivasileiadis, "Efficient database generation for data-driven security assessment of power systems". ". IEEE Trans. Power Systems, vol. 35, no. 1, pp. 30-41, Jan. 2020. https://www.arxiv.org/abs/1806.0107.pdf



Conventional Database Generation

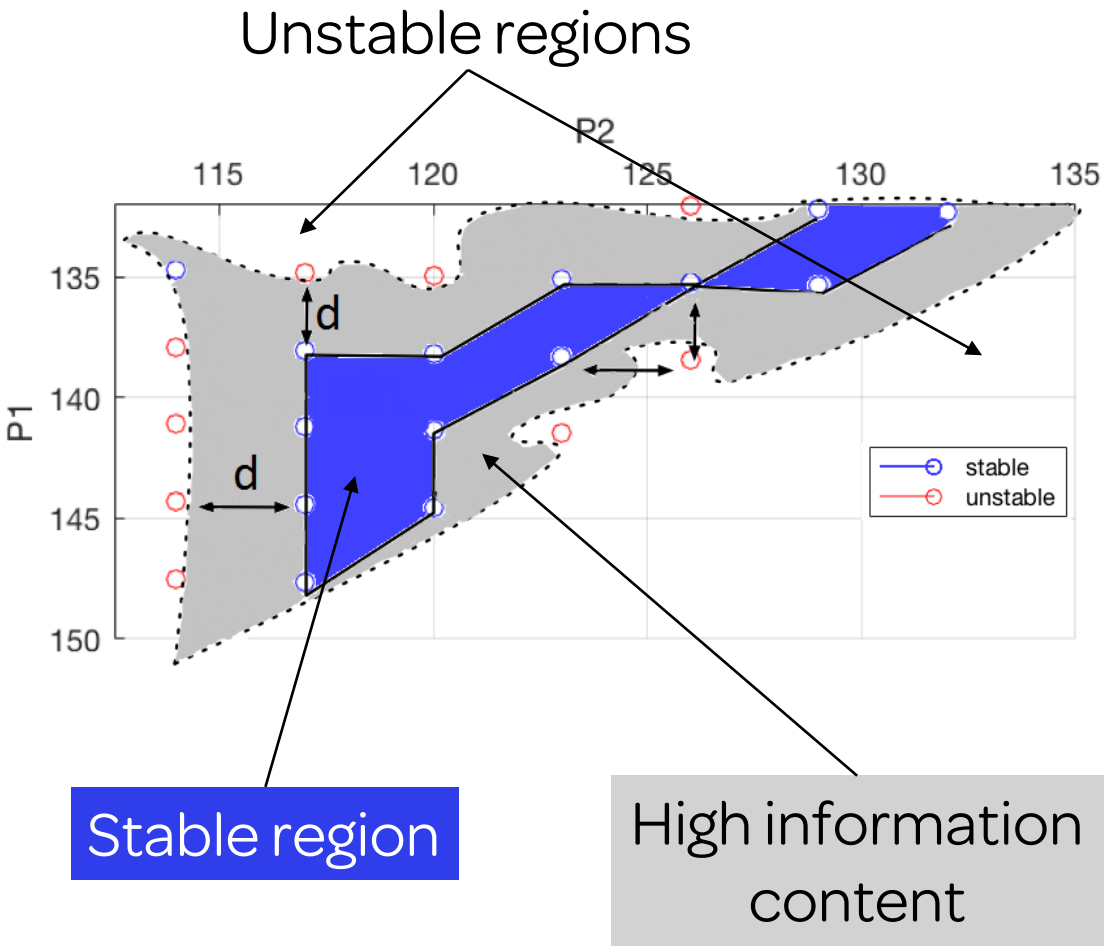
1. Statistical sampling across the input space

2. Often results to highly unbalanced database
   - the stable/safe region is often1%-2% of the total region

Alternative: use our prior knowledge

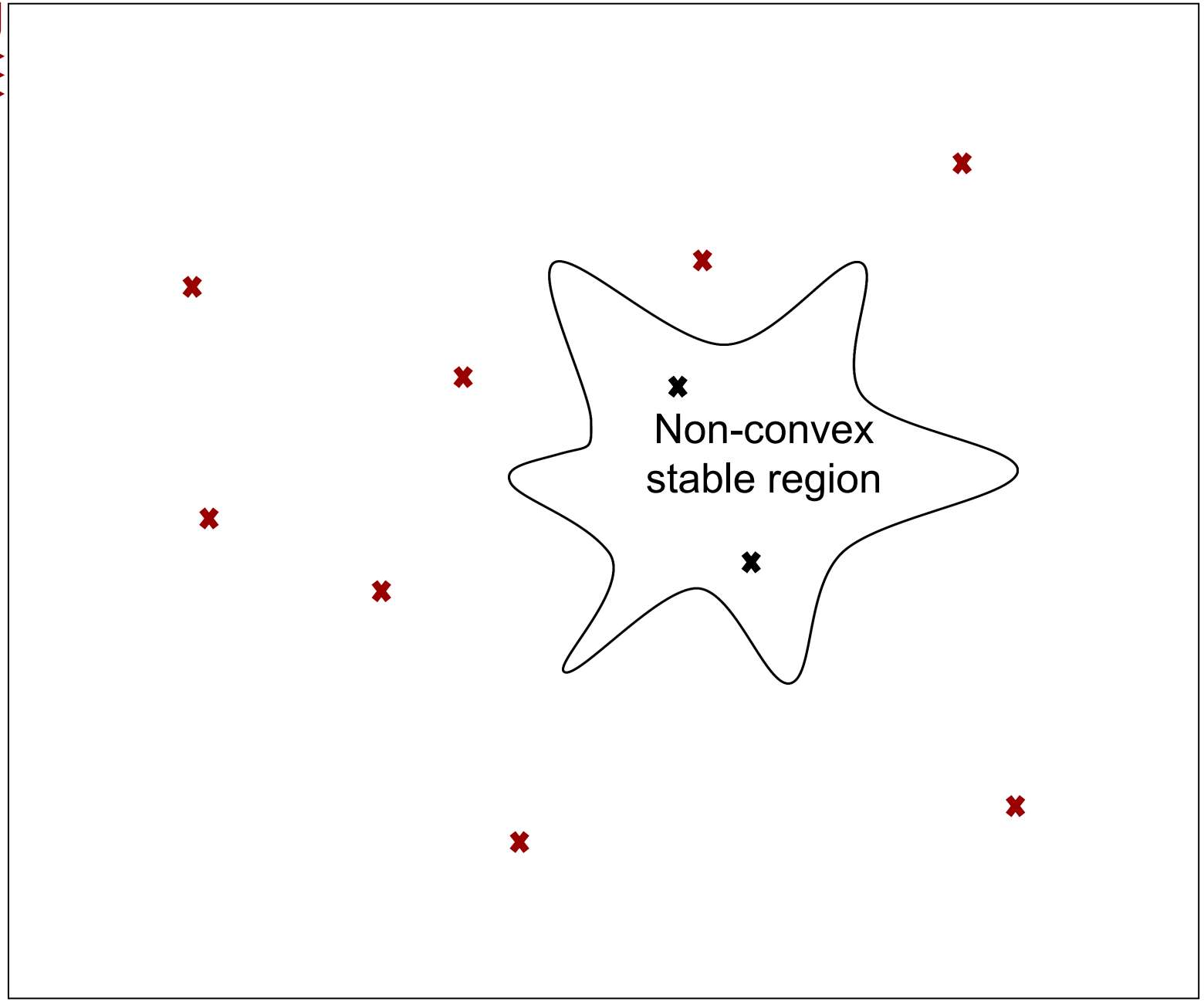
Sampling beyond Statistics: Efficient Database Generation

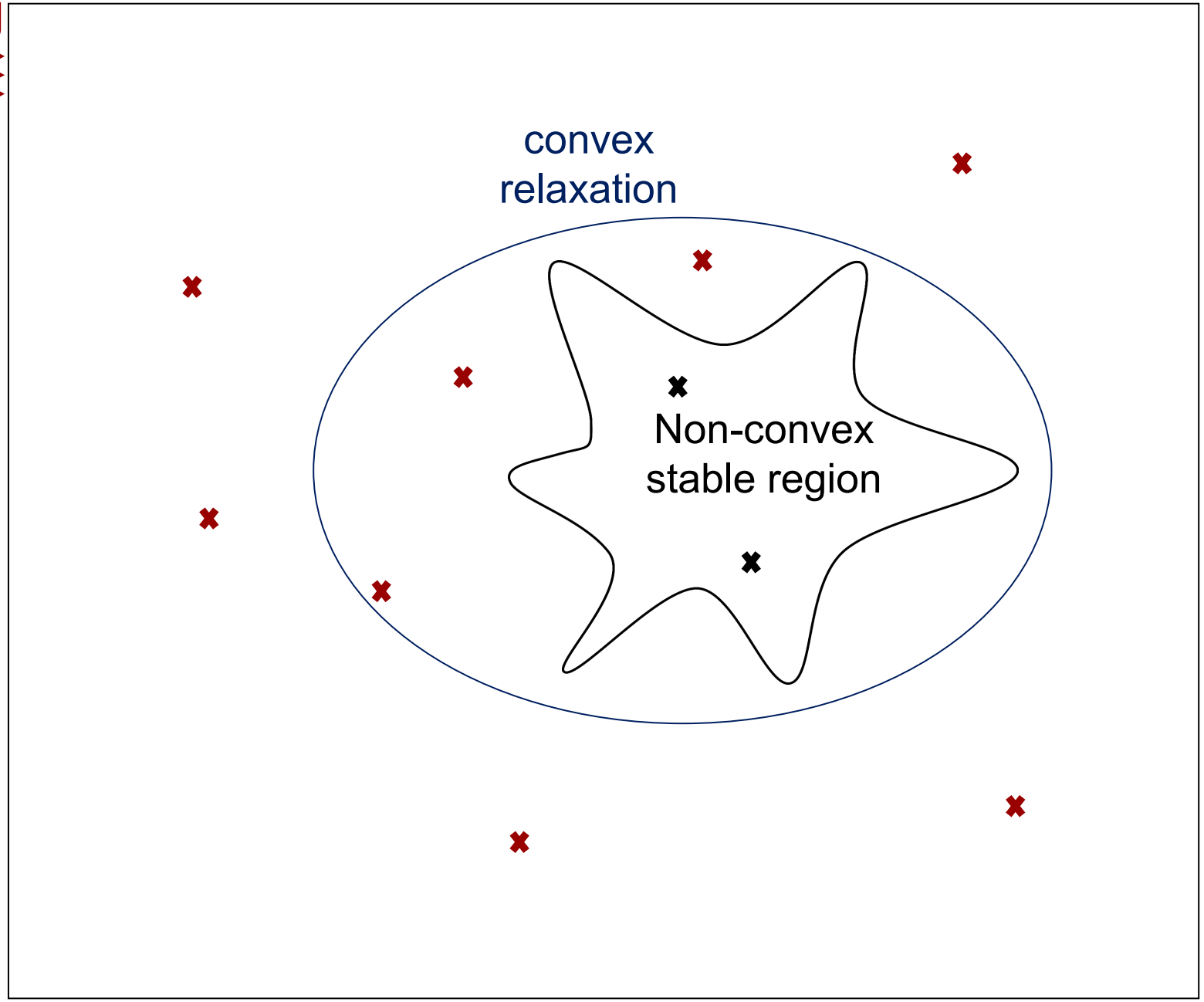
- The goal
  - Focus on the boundary between stability and instability
  - We call it: "high information content" region
- How?
  1. Using convex relaxations
  2. And "Directed Walks"



Real data for the IEEE 14-bus system
N-1 security and small-signal stability

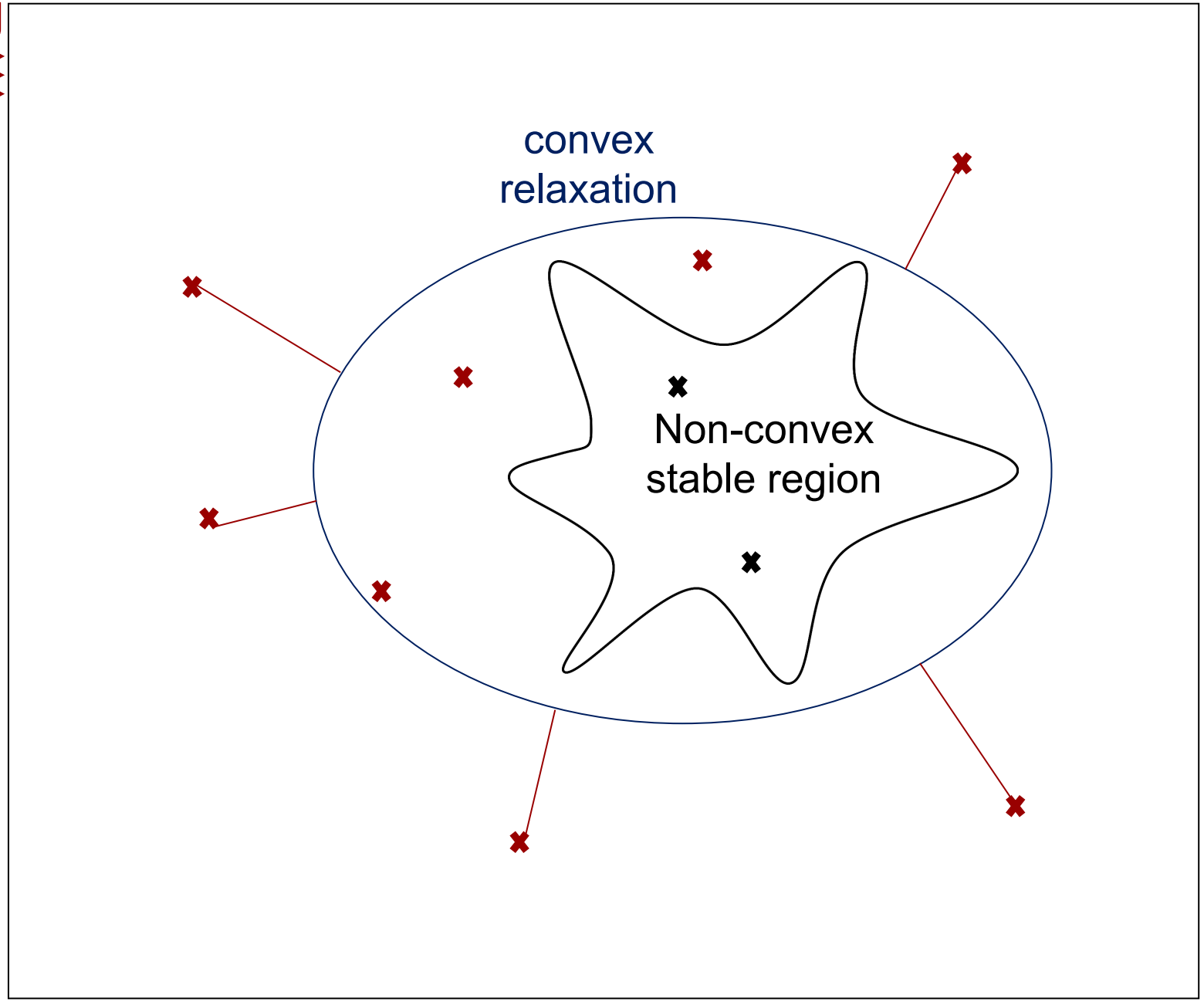
Convex relaxations to discard infeasible regions





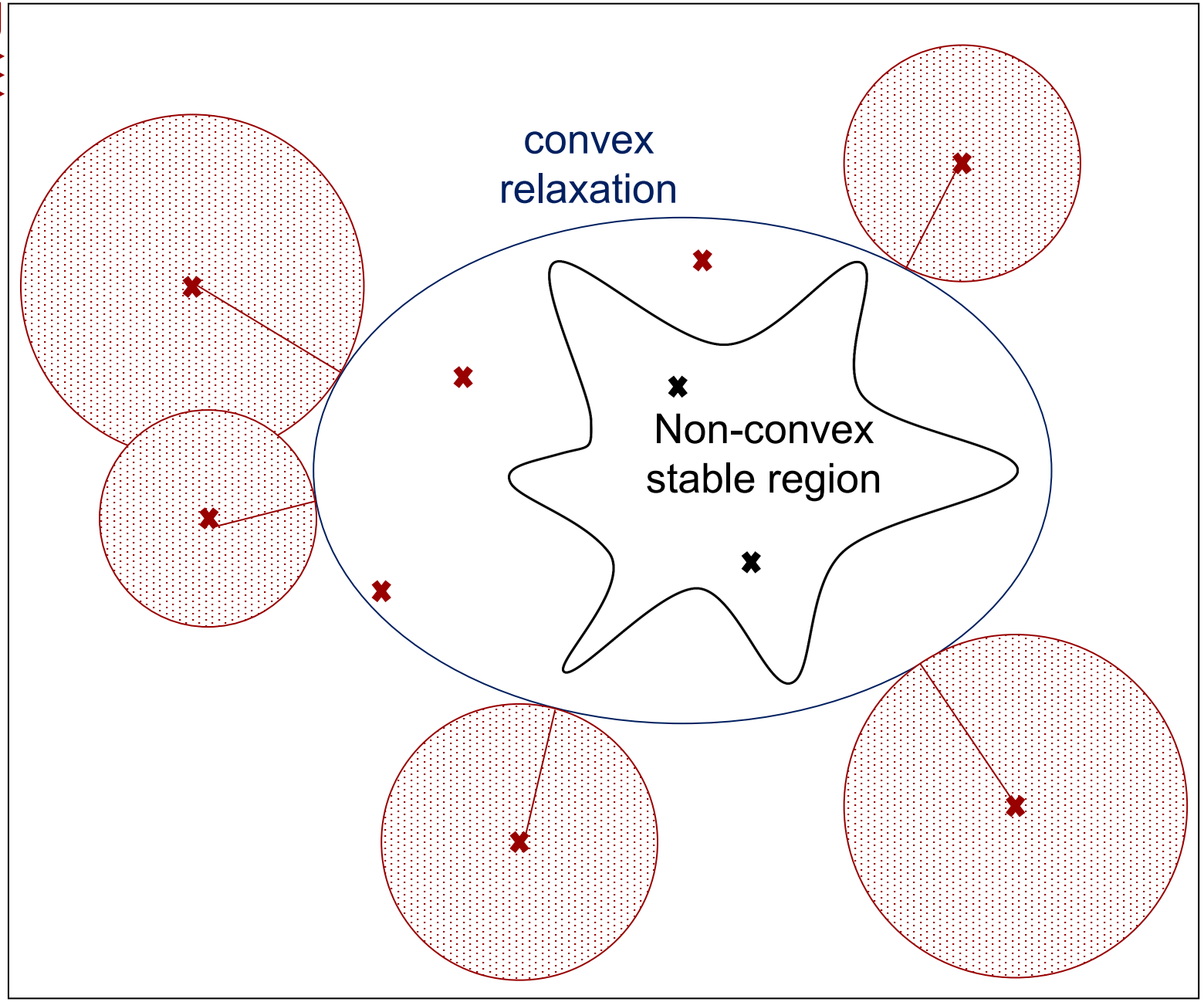
Convex relaxations to discard infeasible regions

- Certificate: if point infeasible for semidefinite relaxation → infeasible for the original problem



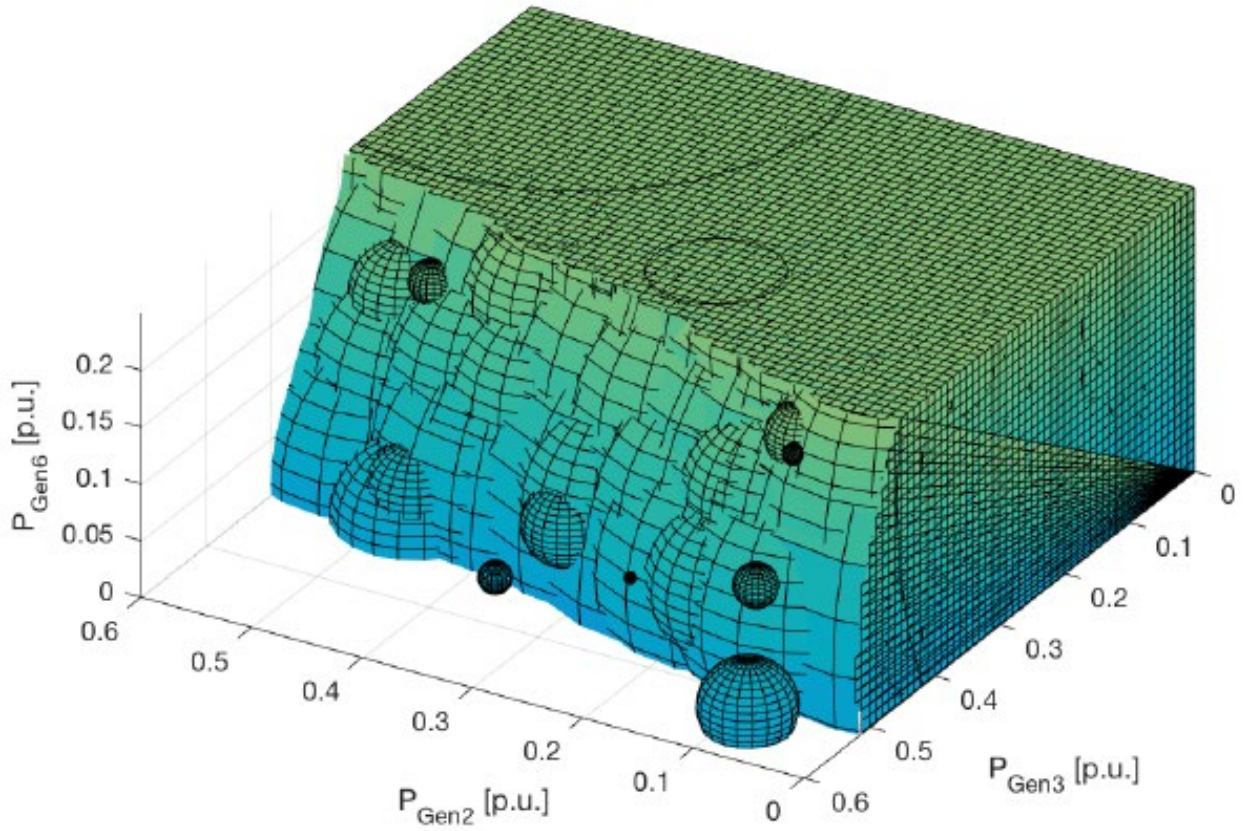
Convex relaxations to discard infeasible regions

- Certificate: if point infeasible for semidefinite relaxation → infeasible for the original problem
- If infeasible point: find minimum radius to feasibility



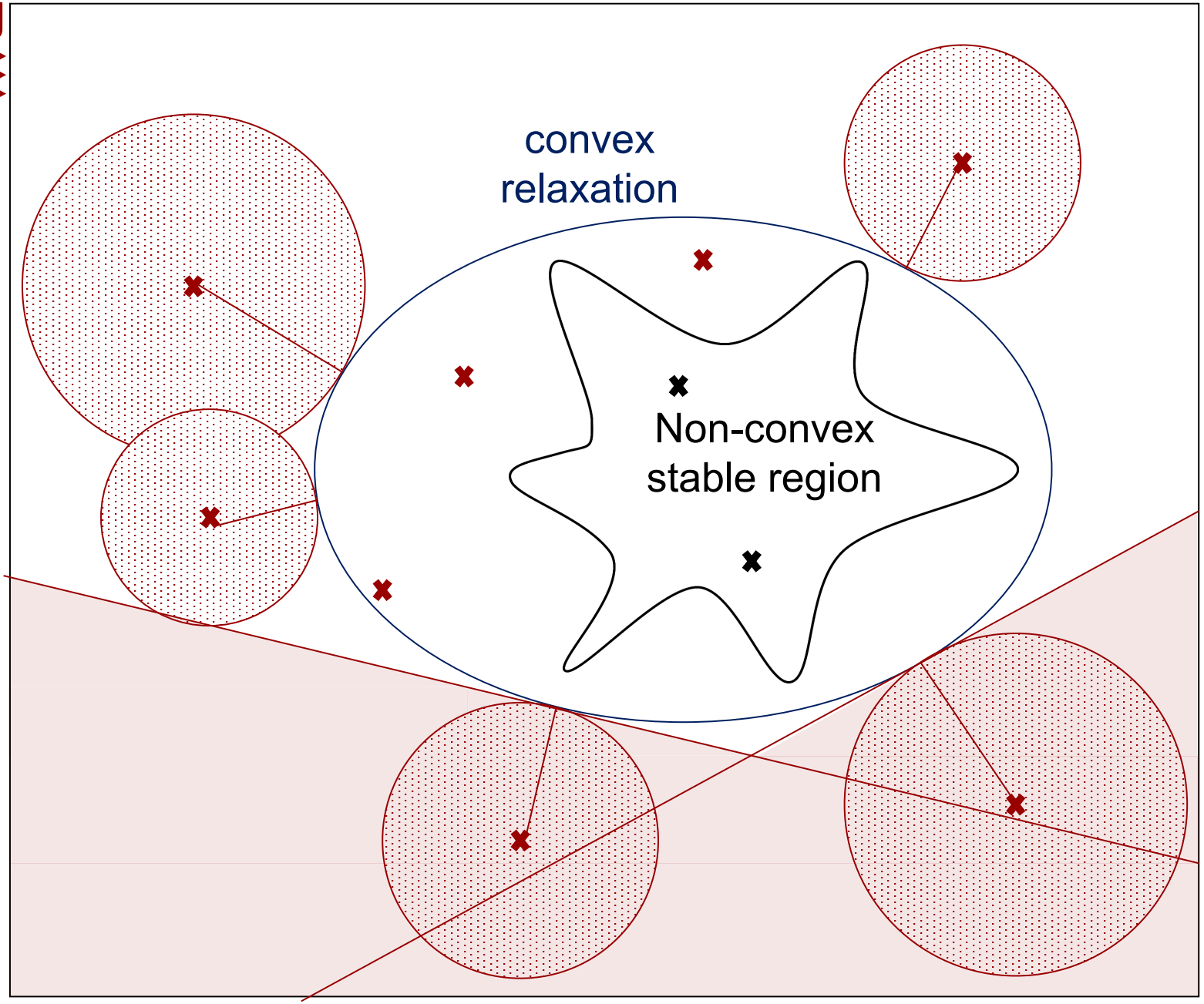
Convex relaxations to discard infeasible regions

- Certificate: if point infeasible for semidefinite relaxation → infeasible for the original problem
- If infeasible point: find minimum radius to feasibility
- Discard all points inside the (hyper)sphere



- 3D projection of hyperspheres
- IEEE 14-bus system
- Rapidly discarding (=classifying) large chunks of the search space as infeasible to focus on the boundary

F. Thams, A. Venzke, R. Eriksson, and S. Chatzivasileiadis, "Efficient database generation for data-driven security assessment of power systems". IEEE Trans. Power Systems, vol. 35, no. 1, pp. 30-41, Jan. 2020.. https://www.arxiv.org/abs/1806.0107.pdf



Convex relaxations to discard infeasible regions

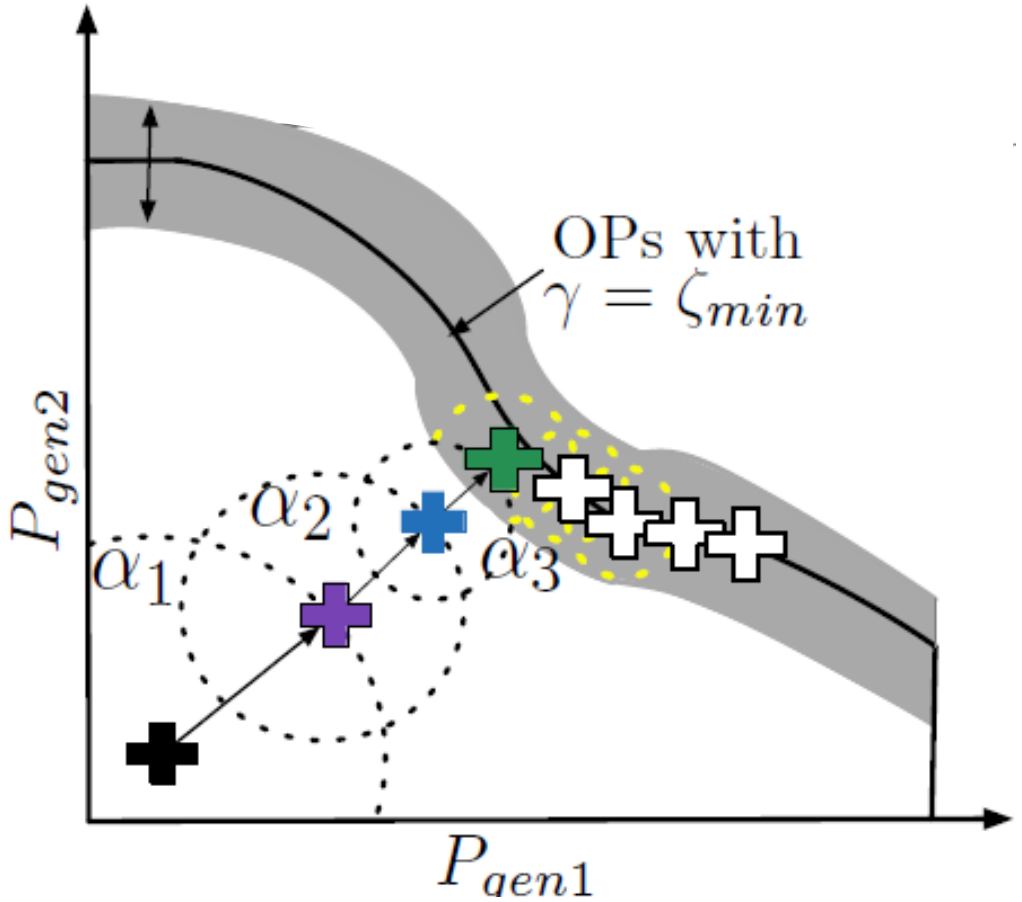
- Extension of this work to hyperplanes

- A. Venzke, D.K. Molzahn, S. Chatzivasileiadis, **Efficient Creation of Datasets for Data-Driven Power System Applications**. PSCC 2020. https://arxiv.org/pdf/1910.01794.pdf

Directed Walks

- "Directed walks": steepest-descent based algorithm to explore the remaining search space, focusing on the area around the security boundary

1. Variable step-size
2. Parallel computation
3. Full N-1 contingency check



Results

	Points close to the security boundary (within distance γ)	
	IEEE 14-bus	NESTA 162-bus
Brute Force	100% of points in **556.0 min**	*intractable*
Importance Sampling	100% of points in **37.0 min**	**901 points** in 35.7 hours
Proposed Method	100% of points in **3.8 min**	**183'295 points** in 37.1 hours

- We tested these databases with decision trees. Further benefits for the decision trees:
  - Higher accuracy
  - Better classification quality (Matthews correlation coefficient)

Takeaway #6

Creating high-quality training databases is extremely complex and an open research topic. We need to go beyond purely statistical methods and exploit the underlying physics during sampling.

Neural Network Verification

for classification NNs in Power Systems

A. Venzke, S. Chatzivasileiadis. Verification of Neural Network Behaviour: Formal Guarantees for Power System Applications. In *IEEE Transactions on Smart Grid*, vol. 12, no. 1, pp. 383-397, Jan. 2021, https://arxiv.org/pdf/1910.01624.pdf

V. Tjeng, K. Y. Xiao, and R. Tedrake, "Evaluating robustness of neural networks with mixed integer programming," in International Conference on Learning Representations (ICLR 2019), 2019

Why is neural network verification important?

- Until recently, the only way to assess the output of the neural networks was to **individually test** each input of interest and pass it through the neural network
  - Accuracy was assessed on discrete samples

Challenge #1: No way to guarantee what the output is for a continuous range of inputs

Why is neural network verification important?

- Until recently, the only way to assess the output of the neural networks was to **individually test** each input of interest and pass it through the neural network
  - Accuracy was assessed on discrete samples

Challenge #1: No way to guarantee what the output is for a continuous range of inputs

---

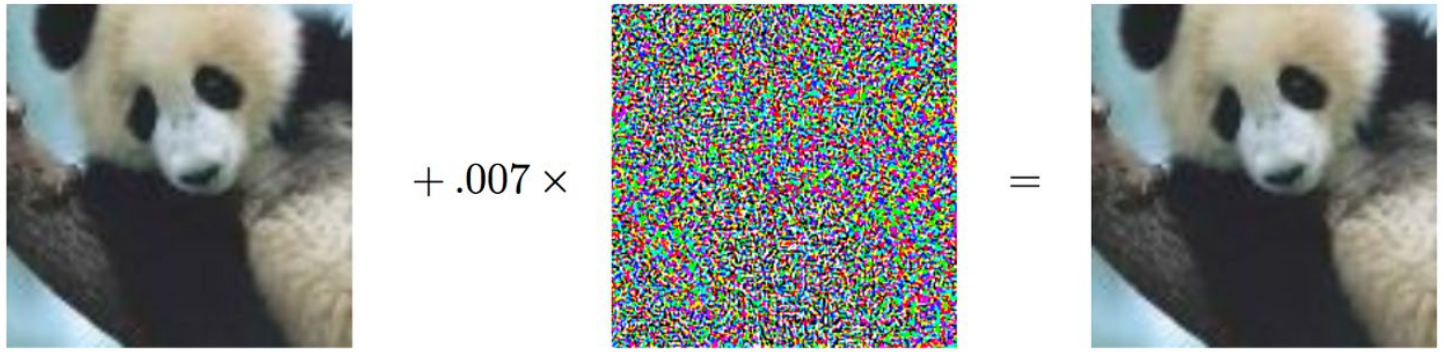
Challenge #2:

The test database determines the performance of the neural network.

Neural network verification does not depend on a test database

Adversarial examples

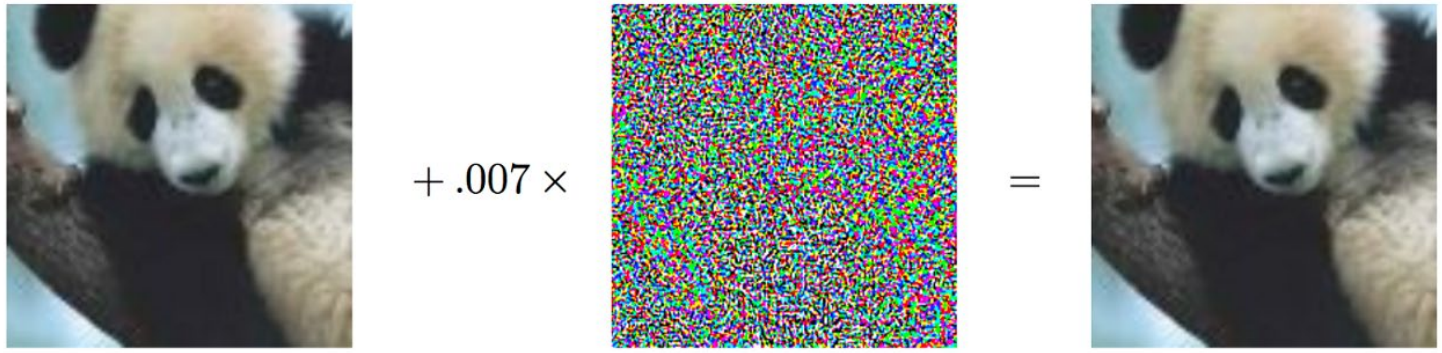
- Adversarial example: small perturbations lead to a false prediction



"panda" noise "gibbon"

Adversarial examples

- Adversarial example: small perturbations lead to a false prediction



- **Challenge #3: No way to systematically identify adversarial examples**

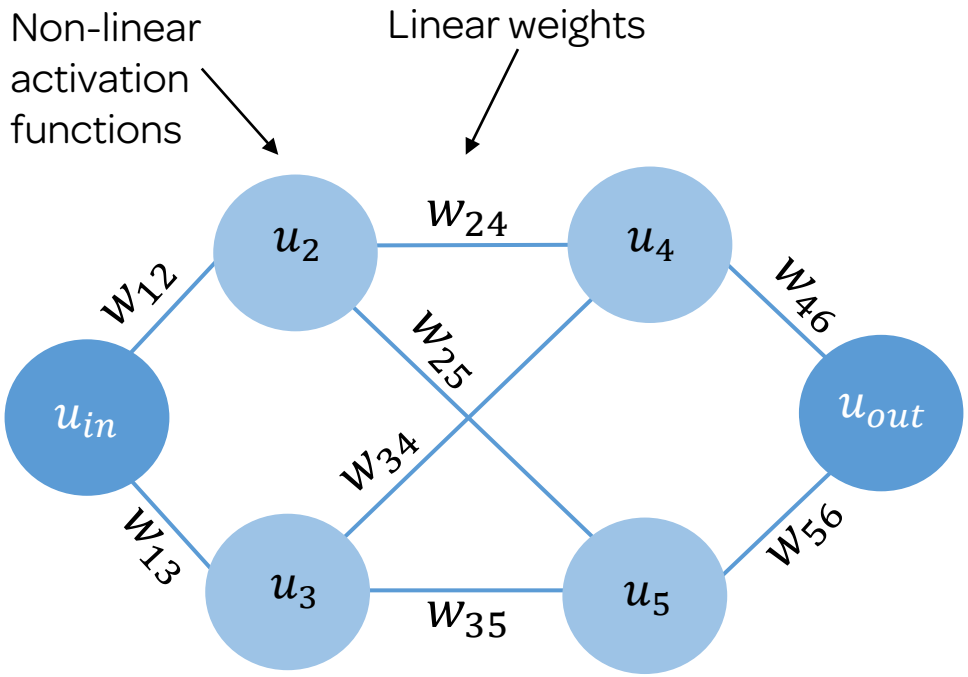
Neural Network Verification: HOW?

1. **Exact transformation:** Convert the neural network to a **set of linear equations with binaries**
   - The Neural Network can be included in a mixed-integer linear program

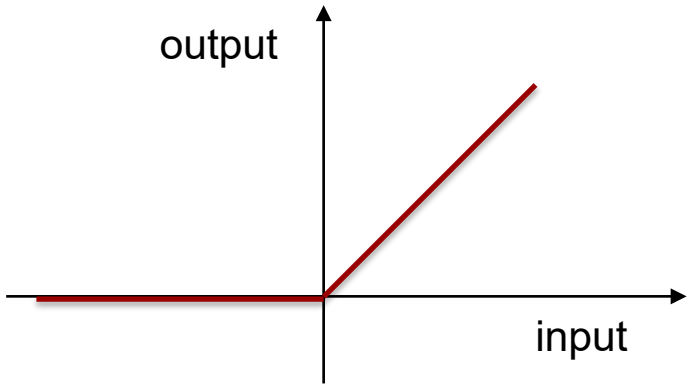
2. Formulate an **optimization** problem (MILP) and solve it → certificate for NN behavior

3. Assess if the neural network output complies with the ground truth

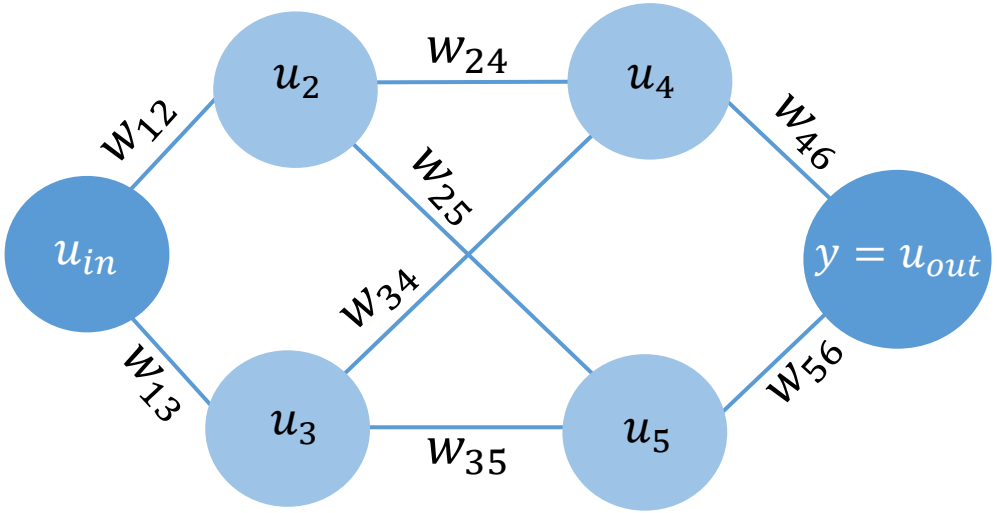
From Neural Networks to Mixed-Integer Linear Programming



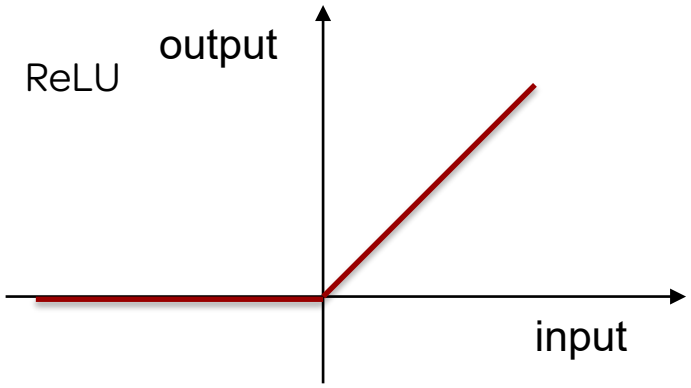
- Most usual activation function: ReLU
- ReLU: Rectifier Linear Unit



From Neural Networks to Mixed-Integer Linear Programming

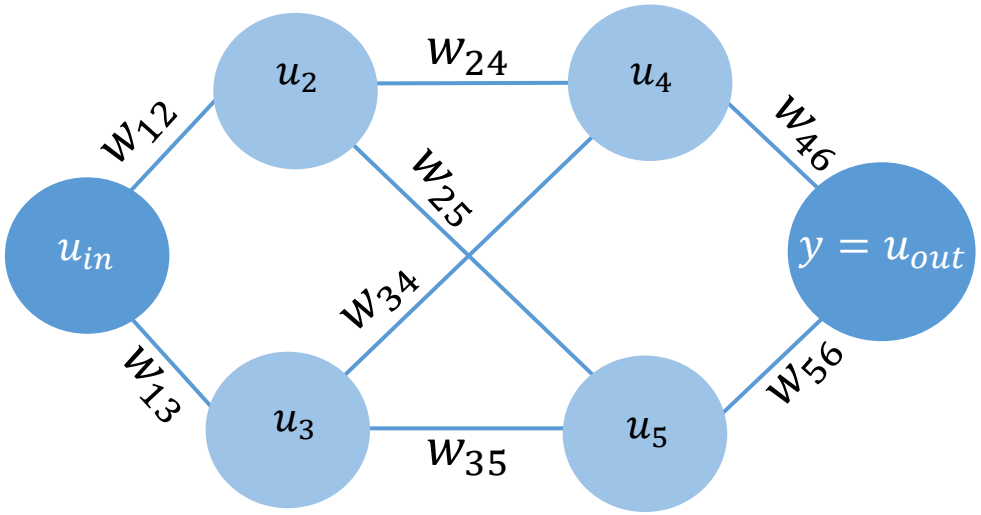


- Linear weights
- On every node: a non-linear activation function
  - ReLU: $\boldsymbol{u_j = \max\left(0, w_{ij} u_i + b_i\right)}$
- But ReLU can be transformed to a piecewise linear function with binaries



MILP

From Neural Networks to Mixed-Integer Linear Programming



- Input: Active power gen. setpoints

$$\mathbf{x} = [p_{g1}, p_{gi}, \ldots, p_{gN}]^T$$

- Output
  - Binary classification: safe/unsafe
  - Output vector y with two elements:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

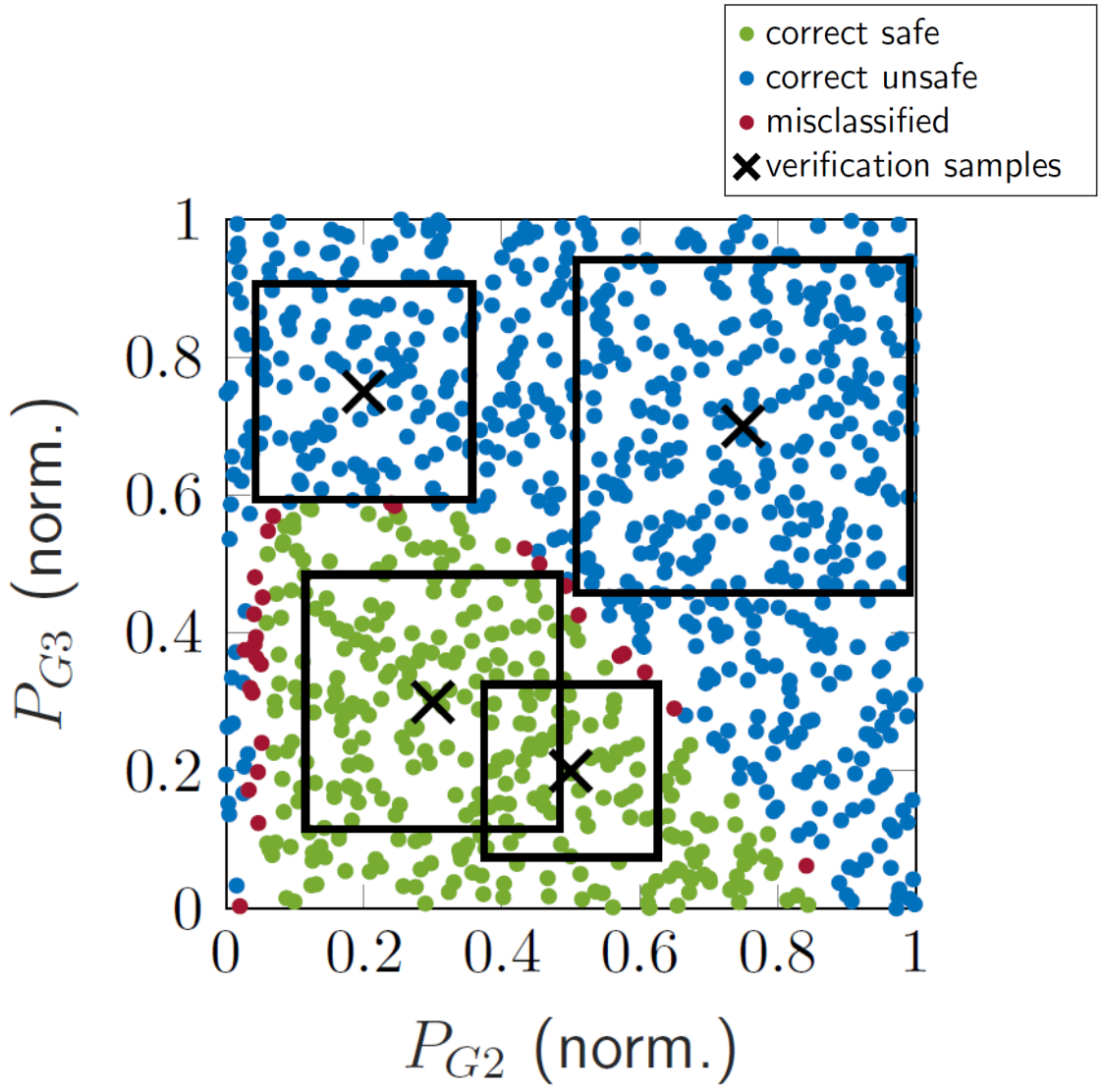
- $y_1 \geq y_2$: safe
- $y_1 < y_2$: unsafe

Certify the output for a continuous range of inputs

- We assume a given input x_{ref} with classification y: $y_1 > y_2$

1. For distance ϵ evaluate if input x exists with different classification y_2

$$\max_{x,y} \quad \mathbf{y_2} - \mathbf{y_1}$$
$$\text{s.t.} \quad y = NN(x)$$
$$|\mathbf{x} - \mathbf{x_{ref}}|_\infty \leq \epsilon$$



A. Venzke, S. Chatzivasileiadis. Verification of Neural Network Behaviour: Formal Guarantees for Power System Applications. *IEEE Transactions on Smart Grid*, Jan. 2021. https://arxiv.org/pdf/1910.01624.pdf

Adversarial examples in safety-critical systems

source: Wu et al. A game-based approximate verification of deep neural networks with provable guarantees. arXiv:1807.03571.

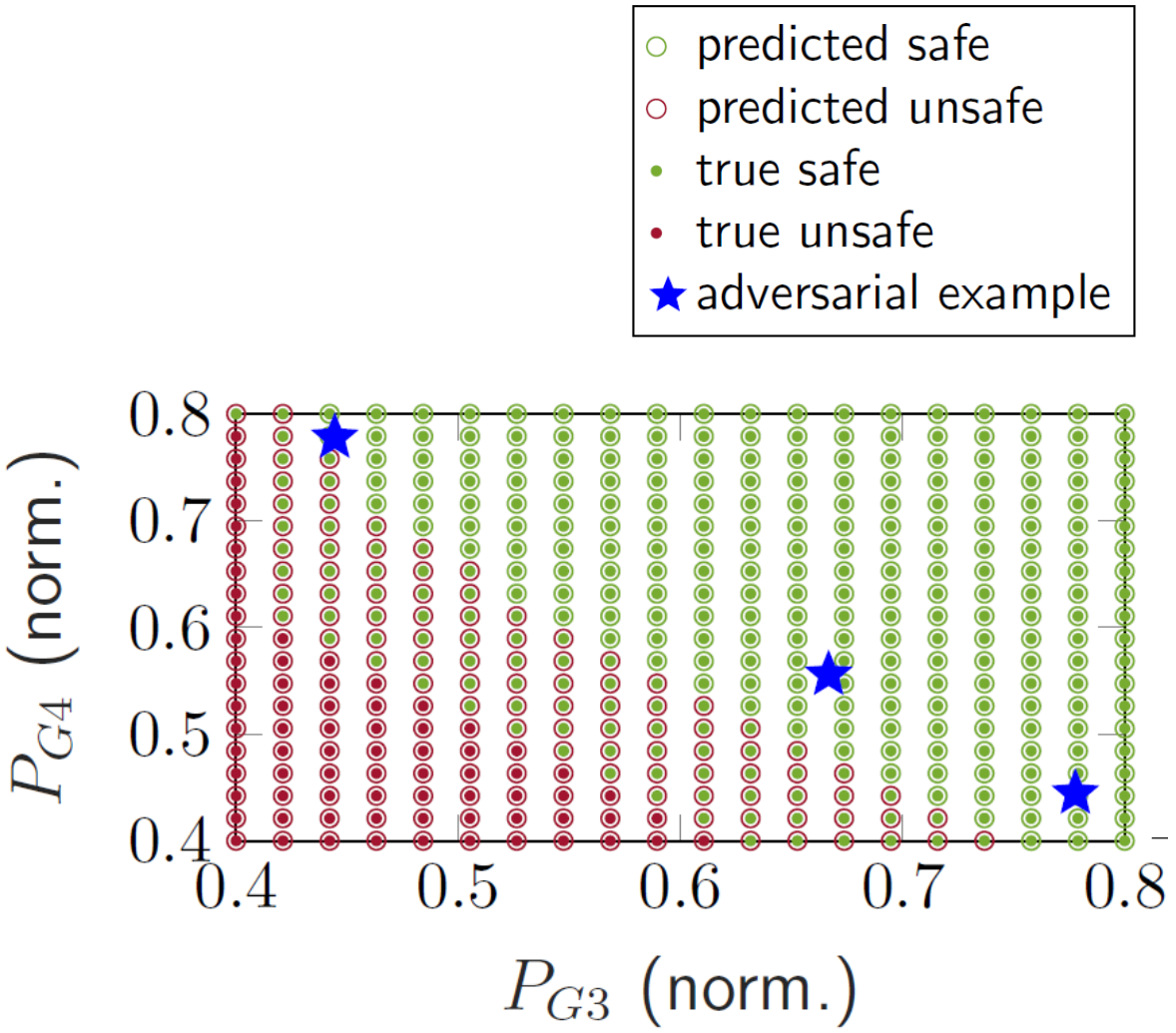
- Adversarial examples exist in many (deep) learning applications
- Major barrier for adoption of machine learning techniques in safety-critical systems!

Systematically identify adversarial examples

- We assume a given input x_{ref} with classification y: $y_1 > y_2$

2. Minimize distance ϵ from $\mathbf{x}_{\text{ref}}$ to input $\mathbf{x}$ with classification y_2

$$\begin{aligned} \min_{\mathbf{x},\mathbf{y},\epsilon} \quad & \epsilon \\ \text{s.t.} \quad & \mathbf{y} = NN(\mathbf{x}) \\ & |\mathbf{x} - \mathbf{x}_{\text{ref}}|_\infty \leq \epsilon \\ & y_2 \geq y_1 \end{aligned}$$



Challenges

- **Tractability** for large neural networks
  - Up to now, we have verified NNs with 5 layers and 50 nodes at each layer (NN used for the 162-bus system)
  - We require weight sparsification, bound tightening, and ReLU pruning (remove binary variables) to maintain tractability

- **Connect verification with ground truth assessment**
  - Currently, we can first certify the neural network output, and we should then assess if this output is correct (i.e. that the NN can be trusted in real operation)
  - **For specific classes of problems, we have integrated this into a single procedure → next section! NNs with worst-case guarantees**

- **Retraining** is necessary to avoid adversarial examples
  - The **quality of the training database is crucial** for good performance!
  - See last slide: Closing the loop!

Takeaway #7

Neural network verification removes barriers for neural network applications in safety-critical operations. High potential for power system applications.

Learning OPF:
Worst-case Guarantees for Regression Neural Networks

A. Venzke, G. Qu, S. Low, S. Chatzivasileiadis, Learning Optimal Power Flow: Worst-case Guarantees for Neural Networks. **Best Student Paper Award** at IEEE SmartGridComm 2020. https://arxiv.org/pdf/2006.11029.pdf

R. Nellikkath, S. Chatzivasileiadis, Physics-Informed Neural Networks for AC Optimal Power Flow
https://arxiv.org/abs/2110.02672 [code]

Quick Reminder: DC Optimal Power Flow

- **Objective**: find the minimum cost generation dispatch
- **Input**: Varying load demand at different nodes

- Considered constant: generator costs; system topology

$$
\begin{aligned}
\min_{\mathbf{p}_g, \boldsymbol{\theta}} \quad & \mathbf{c}^T \mathbf{p}_g && \text{Minimizes generation cost} \\
\text{s.t.} \quad & \mathbf{M}_g \mathbf{p}_g - \mathbf{M}_d \mathbf{p}_d = \mathbf{B}_{bus} \boldsymbol{\theta} && \text{Nodal power balance} \\
& -\mathbf{p}_{line}^{max} \leq \mathbf{B}_{line} \boldsymbol{\theta} \leq \mathbf{p}_{line}^{max} && \text{Transmission line limits} \\
& \mathbf{p}_g^{min} \leq \mathbf{p}_g \leq \mathbf{p}_g^{max} && \text{Generator limits}
\end{aligned}
$$

Quick Reminder: DC Optimal Power Flow

- **Objective**: find the minimum cost generation dispatch
- **Input**: Varying load demand at different nodes

- Considered constant: generator costs; system topology

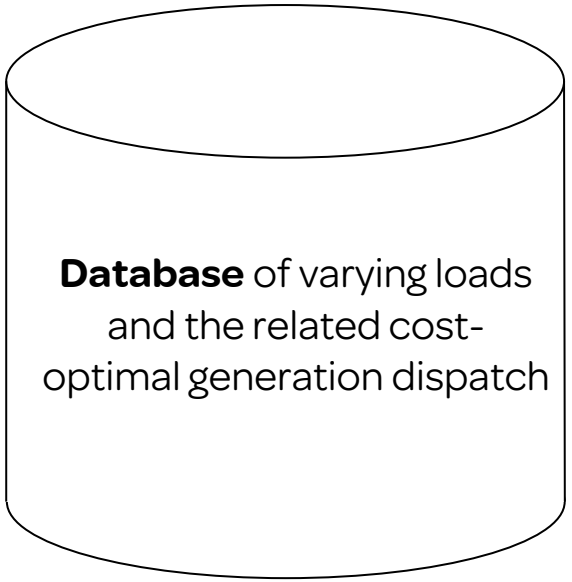
Several recent approaches in the literature that apply Neural Networks for solving the DC-OPF

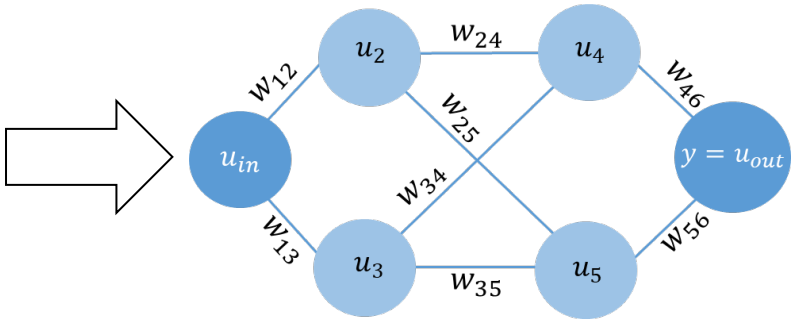
- Demonstrate up to **100x speedup**
- But **no performance guarantees**

$$
\begin{aligned}
\min_{\mathbf{p}_g, \boldsymbol{\theta}} \quad & \mathbf{c}^T \mathbf{p}_g && \text{Minimizes generation cost} \\
\text{s.t.} \quad & \mathbf{M}_g \mathbf{p}_g - \mathbf{M}_d \mathbf{p}_d = \mathbf{B}_{bus} \boldsymbol{\theta} && \text{Nodal power balance} \\
& -\mathbf{p}_{line}^{max} \leq \mathbf{B}_{line} \boldsymbol{\theta} \leq \mathbf{p}_{line}^{max} && \text{Transmission line limits} \\
& \mathbf{p}_g^{min} \leq \mathbf{p}_g \leq \mathbf{p}_g^{max} && \text{Generator limits}
\end{aligned}
$$

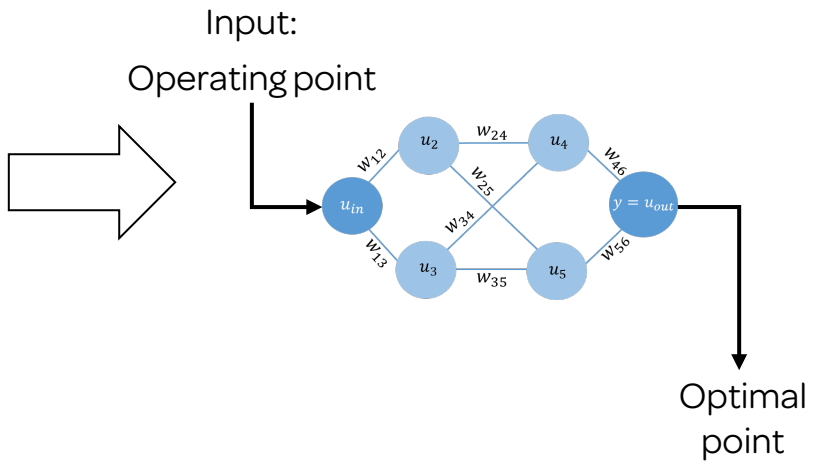
Guiding Application for Regression NN: Learning OPF

Approaches proposed up to now





5. Use the NN



1. Split the database in a training set and a test set

2. Train a neural network

3. Test the neural network

4. Is accuracy high enough?

NN Output:

Optimal generation dispatch

Extremely fast:

up to 100x faster

Part I: Maximum limit-violations

1. Maximum violation of generator limits

$$\nu_{\mathrm{g}} = \max(\hat{\mathbf{p}}_{\mathrm{g}} - \mathbf{p}_{\mathrm{g}}^{\max}, \mathbf{p}_{\mathrm{g}}^{\min} - \hat{\mathbf{p}}_{\mathrm{g}}, \mathbf{0})$$

$$\begin{aligned} \max \quad & \nu_{\mathrm{g}} \\ \text{s.t.} \quad & \mathbf{A}_{\mathrm{d}}\mathbf{p}_{\mathrm{d}} \leq \mathbf{b}_{\mathrm{d}} && \text{Convex polytope as input domain } \mathcal{D} \\ & \hat{\mathbf{p}}_{\mathrm{g}} = NN(\mathbf{p}_{\mathrm{d}}) && \text{Mixed-integer reformulation of trained NN} \end{aligned}$$

Example:

$$0.6\,\boldsymbol{p}_{\mathrm{d}}^{\max} \leq \boldsymbol{p}_{\mathrm{d}} \leq 1.0\,\boldsymbol{p}_{\mathrm{d}}^{\max}$$

Part I: Maximum limit-violations

1. Maximum violation of generator limits

$$\nu_{\mathrm{g}} = \max(\hat{\mathbf{p}}_{\mathrm{g}} - \mathbf{p}_{\mathrm{g}}^{\max}, \mathbf{p}_{\mathrm{g}}^{\min} - \hat{\mathbf{p}}_{\mathrm{g}}, \mathbf{0})$$

$$\begin{aligned} \max \quad & \nu_{\mathrm{g}} \\ \text{s.t.} \quad & \mathbf{A}_{\mathrm{d}}\mathbf{p}_{\mathrm{d}} \leq \mathbf{b}_{\mathrm{d}} && \text{Convex polytope as input domain } \mathcal{D} \\ & \hat{\mathbf{p}}_{\mathrm{g}} = NN(\mathbf{p}_{\mathrm{d}}) && \text{Mixed-integer reformulation of trained NN} \end{aligned}$$

Example:

$$0.6\,\boldsymbol{p}_{\mathrm{d}}^{\max} \leq \boldsymbol{p}_{\mathrm{d}} \leq 1.0\,\boldsymbol{p}_{\mathrm{d}}^{\max}$$

2. Maximum violation of line limits

$$\nu_{\mathrm{line}} = \max(\underbrace{|\mathbf{B}_{\mathrm{line}}\tilde{\mathbf{B}}_{\mathrm{bus}}^{-1}(\mathbf{M}_{\mathrm{g}}\hat{\mathbf{p}}_{\mathrm{g}} - \mathbf{M}_{\mathrm{d}}\mathbf{p}_{\mathrm{d}})^{\mathrm{nsb}}|}_{\text{Line flow equations for DC-OPF based on PTDFs}} - \mathbf{p}_{\mathrm{line}}^{\max}, \mathbf{0})$$

$$\begin{aligned} \max \quad & \nu_{\mathrm{line}} \\ \text{s.t.} \quad & \mathbf{A}_{\mathrm{d}}\mathbf{p}_{\mathrm{d}} \leq \mathbf{b}_{\mathrm{d}} && \text{Convex polytope as input domain } \mathcal{D} \\ & \hat{\mathbf{p}}_{\mathrm{g}} = NN(\mathbf{p}_{\mathrm{d}}) && \text{Mixed-integer reformulation of trained NN} \end{aligned}$$

Worst violation over the **whole training dataset** (training+test set)

Our algorithm: **provable** worst-case guarantee over the **whole input domain**

	Empirical lower bound		Exact worst-case guarantee	
Test cases	ν_{g} (MW)	ν_{line} (MW)	ν_{g} (MW)	ν_{line} (MW)
case9				
case30				
case39				
case57				
case118				
case162				
case300				

ν_{g} Maximum violation of generator limits

ν_{line} Maximum violation of line limits

Worst violation over the **whole training dataset** (training+test set)

Our algorithm: **provable** worst-case guarantee over the **whole input domain**

	Empirical lower bound		Exact worst-case guarantee	
Test cases	ν_{g} (MW)	ν_{line} (MW)	ν_{g} (MW)	ν_{line} (MW)
case9	2.5	1.8	2.8	1.9
case30	1.7	0.6	3.6	3.1
case39	51.9	37.2	270.6	120.0
case57	4.2	0.0	23.7	0.0
case118	149.4	15.6	997.8	510.8
case162	228.0	180.0	1563.3	974.1
case300	474.5	692.7	3658.5	3449.3

ν_{g} Maximum violation of generator limits

ν_{line} Maximum violation of line limits

Over the whole input domain **violations can be much larger** (here ~7x) compared to what has been estimated empirically on the dataset

Worst violation over the **whole training dataset** (training+test set)

Our algorithm: **provable** worst-case guarantee over the **whole input domain**

	Empirical lower bound		Exact worst-case guarantee	
Test cases	ν_{g} (MW)	ν_{line} (MW)	ν_{g} (MW)	ν_{line} (MW)
case9	2.5	1.8	2.8	1.9
case30	1.7	0.6	3.6	3.1
case39	51.9	37.2	270.6	120.0
case57	4.2	0.0	23.7	0.0
case118	149.4	15.6	997.8	510.8
case162	228.0	180.0	1563.3	974.1
case300	474.5	692.7	3658.5	3449.3

ν_{g} Maximum violation of generator limits

ν_{line} Maximum violation of line limits

Our method provides **guarantees that no NN output will violate the line limits** over the whole input domain

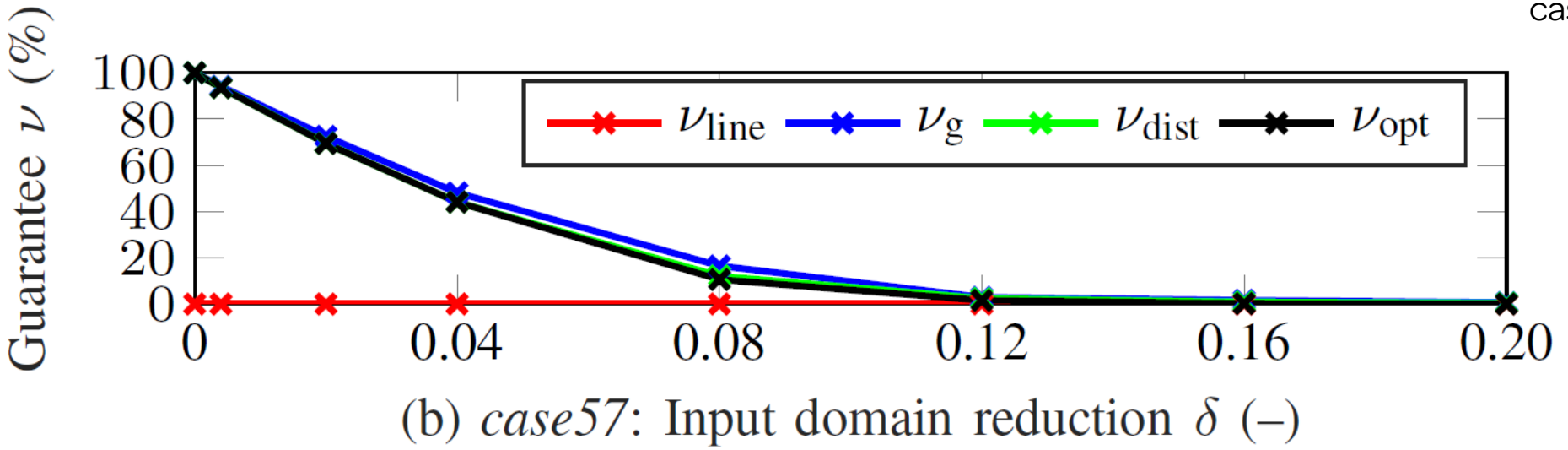
How can we reduce the worst-case violations?

- From our experiments with DC-OPF in 7 different test power systems, we observed that the **worst-case violations occur at the boundary of the input domain**

- Possible solution:
  1. Train on a larger input domain
  2. Use the NN on a subdomain of the original training input

Reducing the worst-case violations

100% on y-axis = Worst-case violation for $\delta = 0$



- Input domain used for training

$$0.6\,\boldsymbol{p}_{\text{d}}^{\text{max}} \leq \boldsymbol{p}_{\text{d}} \leq 1.0\,\boldsymbol{p}_{\text{d}}^{\text{max}}$$

- Input domain for using the NN (and where worst-case violations were evaluated)

$$(0.6+\delta)\mathbf{p}_{\text{d}}^{\text{max}} \leq \mathbf{p}_{\text{d}} \leq (1.0-\delta)\mathbf{p}_{\text{d}}^{\text{max}}$$

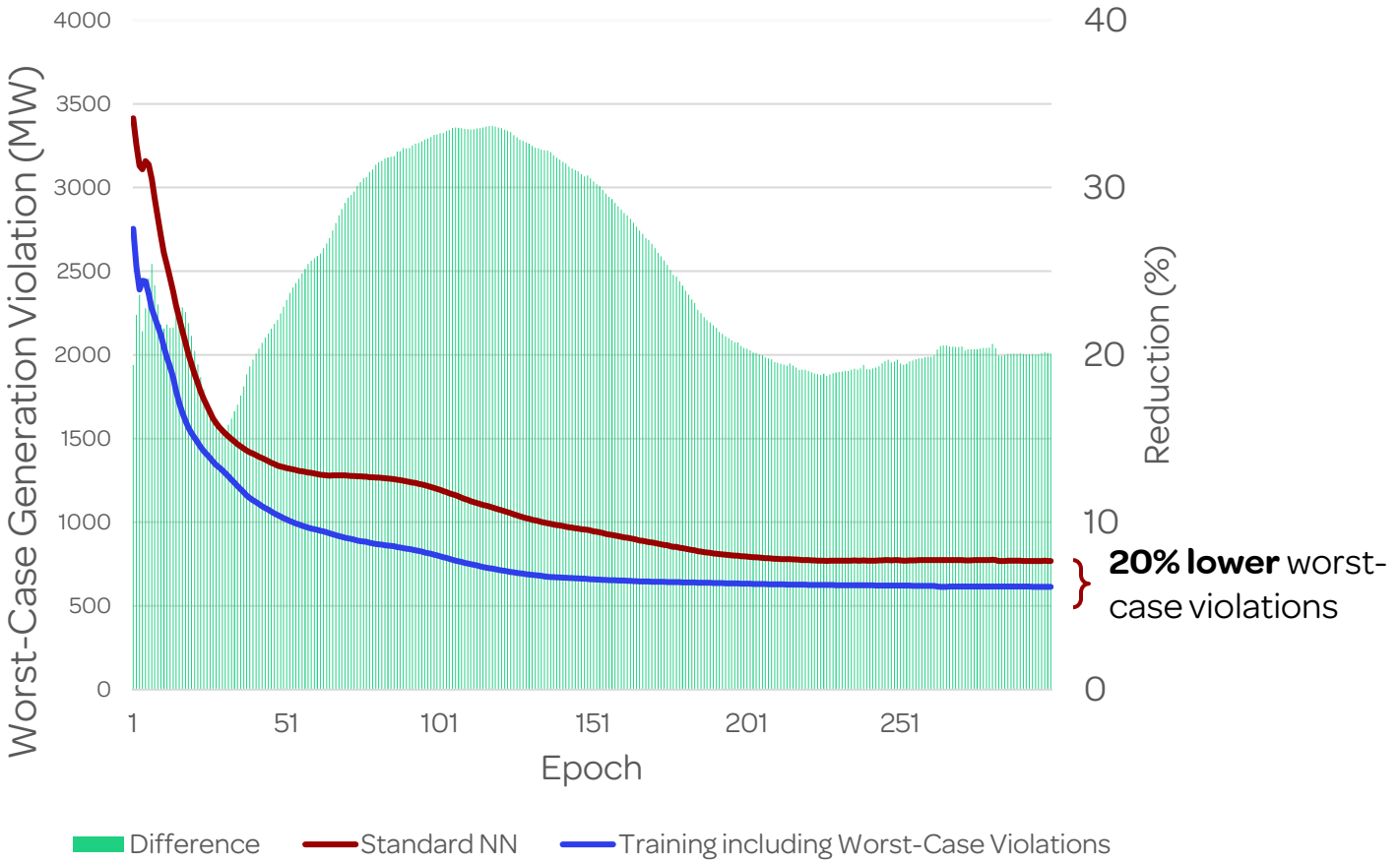
Example: If $\delta = 0.1$ then $0.7\,\boldsymbol{p}_{\text{d}}^{\text{max}} \leq \boldsymbol{p}_{\text{d}} \leq 0.9\,\boldsymbol{p}_{\text{d}}^{\text{max}}$

Next Step: Integrate the Worst-Case Guarantees inside the Neural Network Training

Preliminary Results

- Approach:
    1. Train for Epoch 1 (standard NN training, i.e. reduce mean absolute error)
    2. Determine the worst-case violations
    3. Optimization problem → I can **determine the gradients of the weights and biases to reduce the worst-case violations**
        - Steps 2 and 3 are completely independent of the training datasets. They consider the whole input region.
    4. **Update** the NN weights and biases based on the gradients of Step 3
    5. Train for Epoch 2 (standard NN training, i.e. reduce mean absolute error)
    6. Go to Step 2

Standard NNs vs Training with Worst-Case Violations

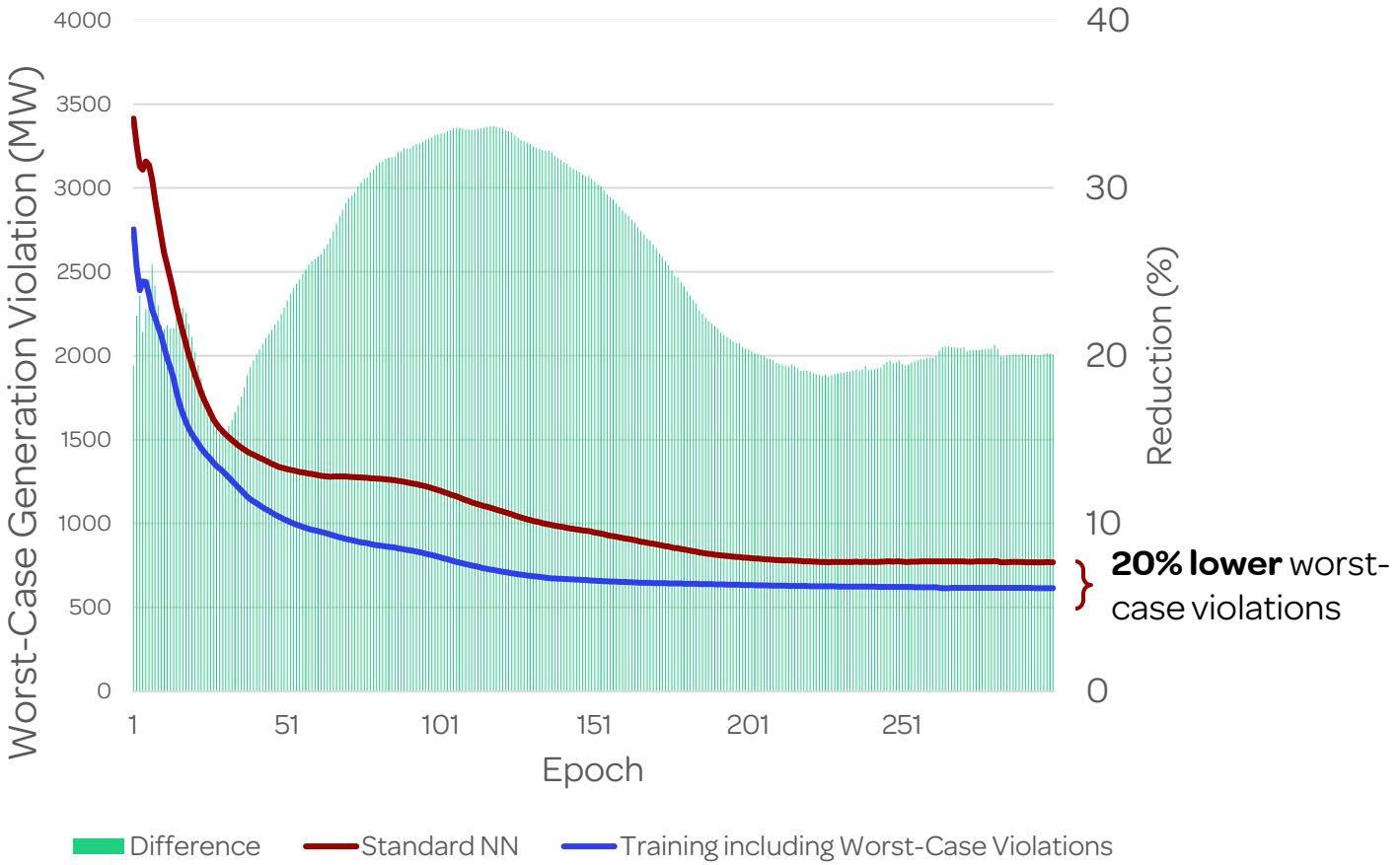


Preliminary Results

- NN training including worst-case violations consistently results to **~20% lower worst-case violations**

- Very low impact on the average performance: MAE is 0.04% higher

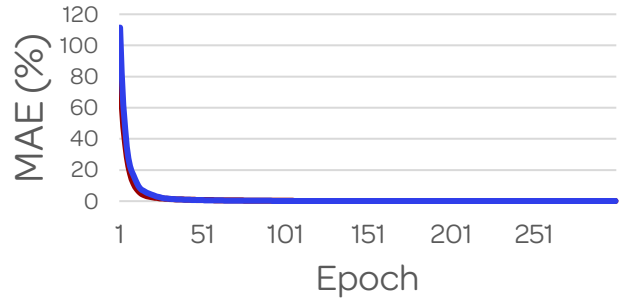
Standard NNs vs Training with Worst-Case Violations



Preliminary Results

- NN training including worst-case violations consistently results to **~20% lower worst-case violations**

- Very low impact on the average performance: MAE is 0.04% higher

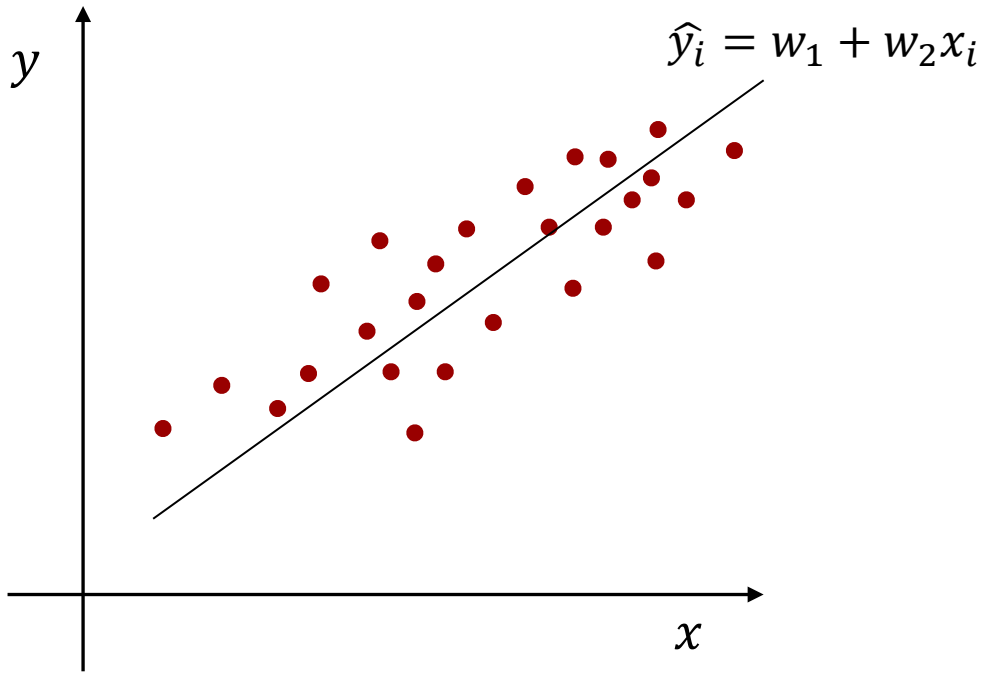


Physics-Informed Neural Networks for Power Systems

Neural Networks: An advanced form of non-linear regression

y_i: actual/correct value

$\hat{y}_i$: estimated value



Loss function: Estimate best w_1, w_2 to fit the training data

$$\min_{w_1, w_2} \quad \|y_i - \hat{y}_i\|$$

s.t.

$$\hat{y}_i = w_1 + w_2 x_i \qquad \forall i$$

Traditional training of neural networks required no information about the underlying physical model. Just data!

Physics Informed Neural Networks

- Automatic differentiation: derivatives of the neural network output with respect to the input can be computed during the training procedure
- A differential-algebraic model of a physical system can be included in the neural network training*
- Neural networks can now exploit knowledge of the actual physical system
- Machine learning platforms (e.g. Tensorflow) enable these capabilities

*M. Raissi, P. Perdikaris, and G. Karniadakis, Physics-Informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations", Journal of Computational Physics, vol.378, pp. 686-707, 2019

Physics-Informed Neural Networks for Power Systems

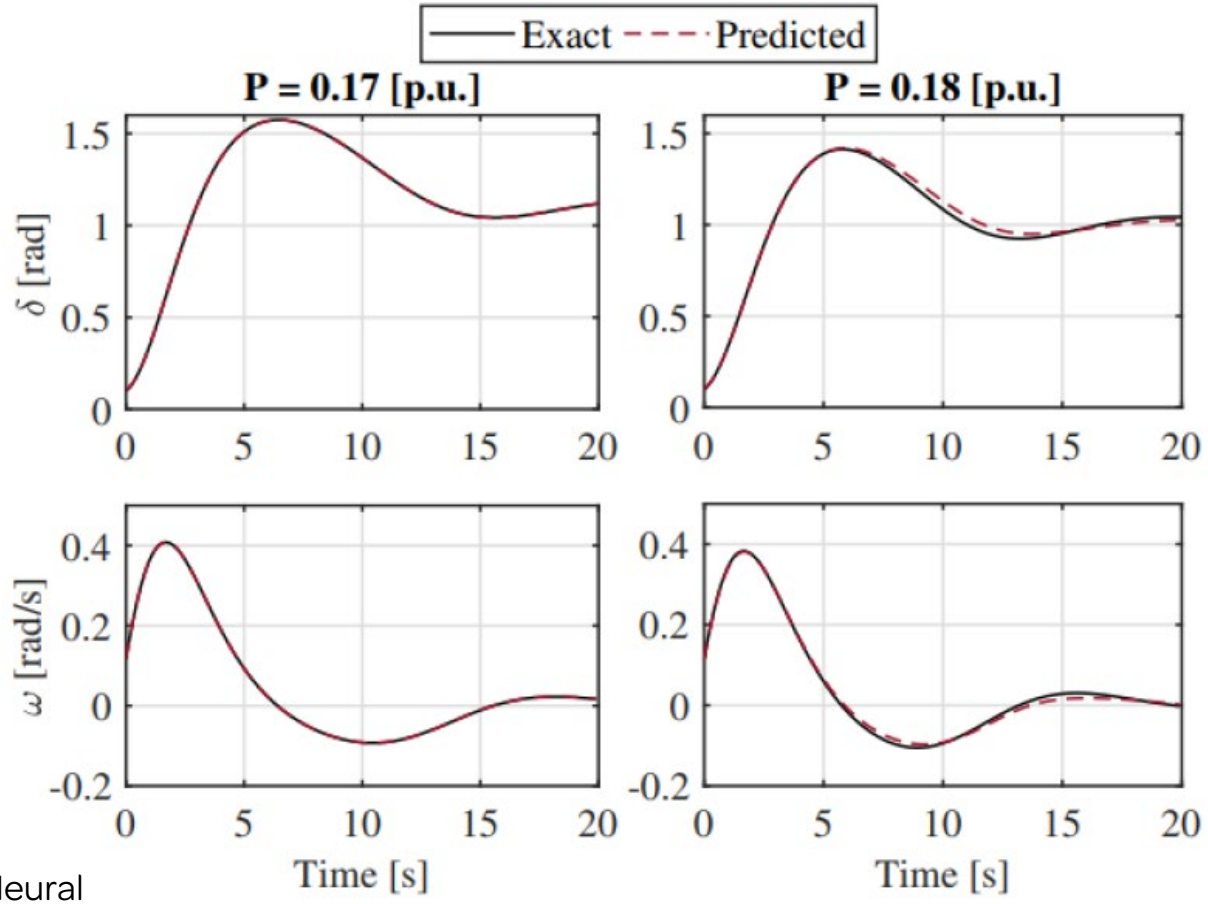
"Original" Loss function

$$\min_{\mathbf{W},\mathbf{b}} \quad \frac{1}{|N_\delta|}\sum_{i\in N_\delta}|\hat{\delta}-\delta^i|^2+\frac{1}{|N_f|}\sum_{i\in N_f}|f(\hat{\delta})|^2 \tag{6a}$$

$$s.t. \quad \hat{\delta}=NN(t,P_m,\mathbf{W},\mathbf{b}) \tag{6b}$$

$$\dot{\hat{\delta}}=\frac{\partial\hat{\delta}}{\partial t}, \qquad \ddot{\hat{\delta}}=\frac{\partial\dot{\hat{\delta}}}{\partial t} \tag{6c}$$

$$f(\hat{\delta})=M\ddot{\hat{\delta}}+D\dot{\hat{\delta}}+A\sin\hat{\delta}-P_m \tag{6d}$$



G. S. Misyris, A. Venzke, S. Chatzivasileiadis, Physics-Informed Neural Networks for Power Systems. Presented at the Best Paper Session of IEEE PES GM 2020. https://arxiv.org/pdf/1911.03737.pdf

Physics-Informed Neural Networks for Power Systems

"Original" Loss function

"Physics-Informed" term

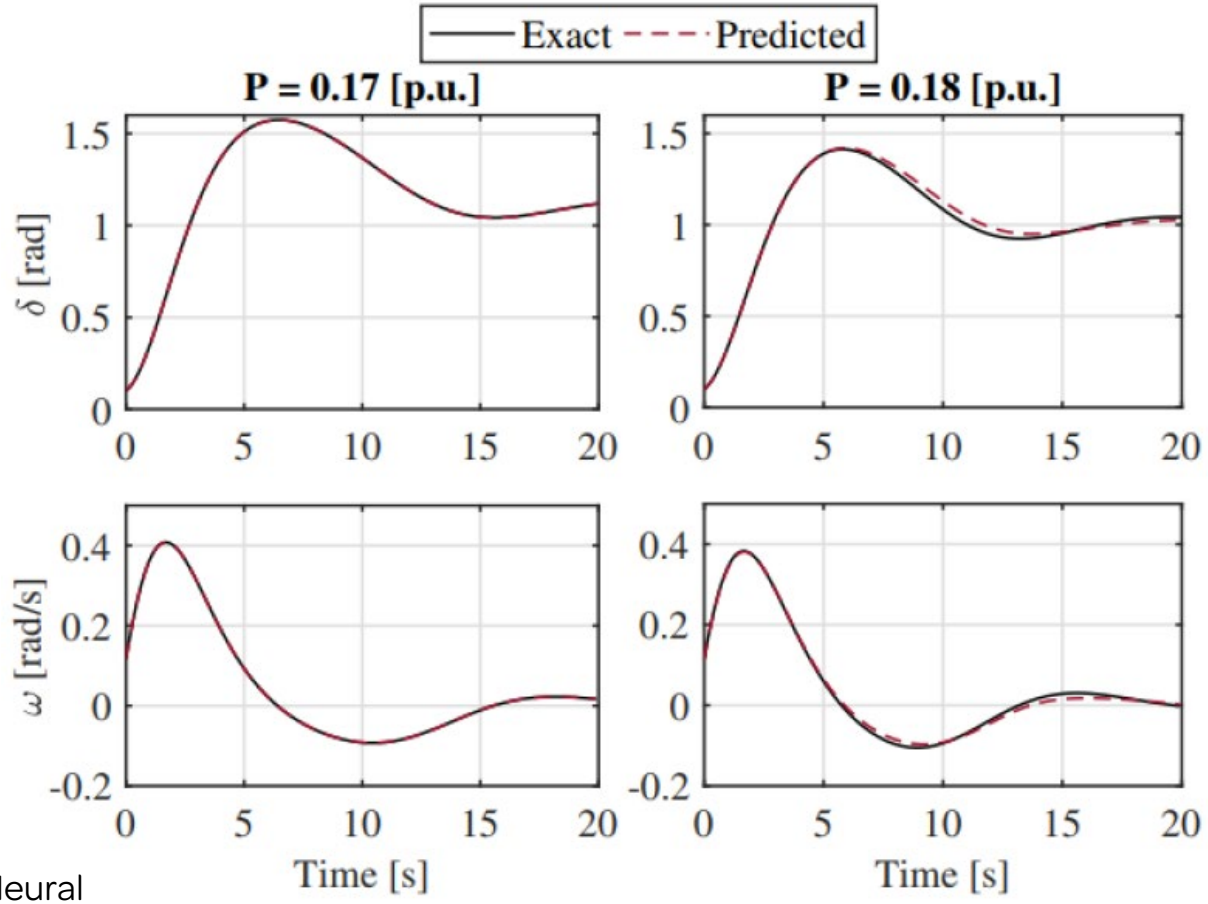
$$\min_{\mathbf{W},\mathbf{b}} \quad \frac{1}{|N_\delta|}\sum_{i\in N_\delta}|\hat{\delta}-\delta^i|^2+\frac{1}{|N_f|}\sum_{i\in N_f}|f(\hat{\delta})|^2 \tag{6a}$$

$$s.t. \quad \hat{\delta}=NN(t,P_m,\mathbf{W},\mathbf{b}) \tag{6b}$$

$$\dot{\hat{\delta}}=\frac{\partial\hat{\delta}}{\partial t}, \qquad \ddot{\hat{\delta}}=\frac{\partial\dot{\hat{\delta}}}{\partial t} \tag{6c}$$

$$f(\hat{\delta})=M\ddot{\hat{\delta}}+D\dot{\hat{\delta}}+A\sin\hat{\delta}-P_m \tag{6d}$$

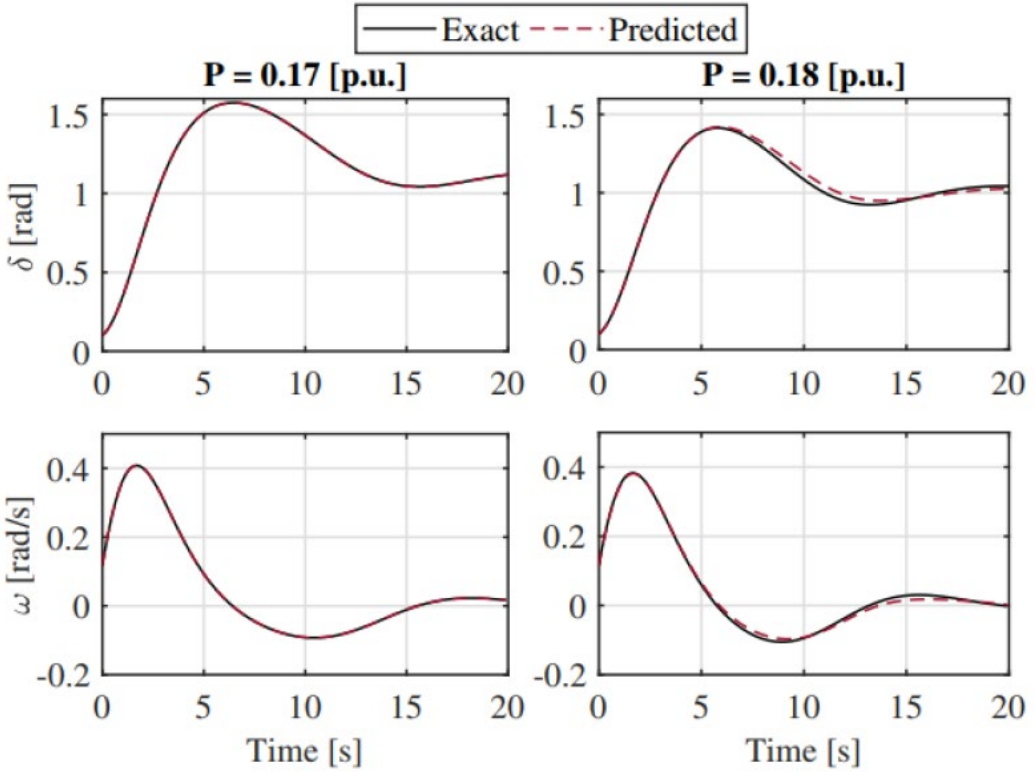
Swing equation



G. S. Misyris, A. Venzke, S. Chatzivasileiadis, Physics-Informed Neural Networks for Power Systems. Presented at the Best Paper Session of IEEE PES GM 2020. https://arxiv.org/pdf/1911.03737.pdf

Physics-Informed Neural Networks for Power Systems

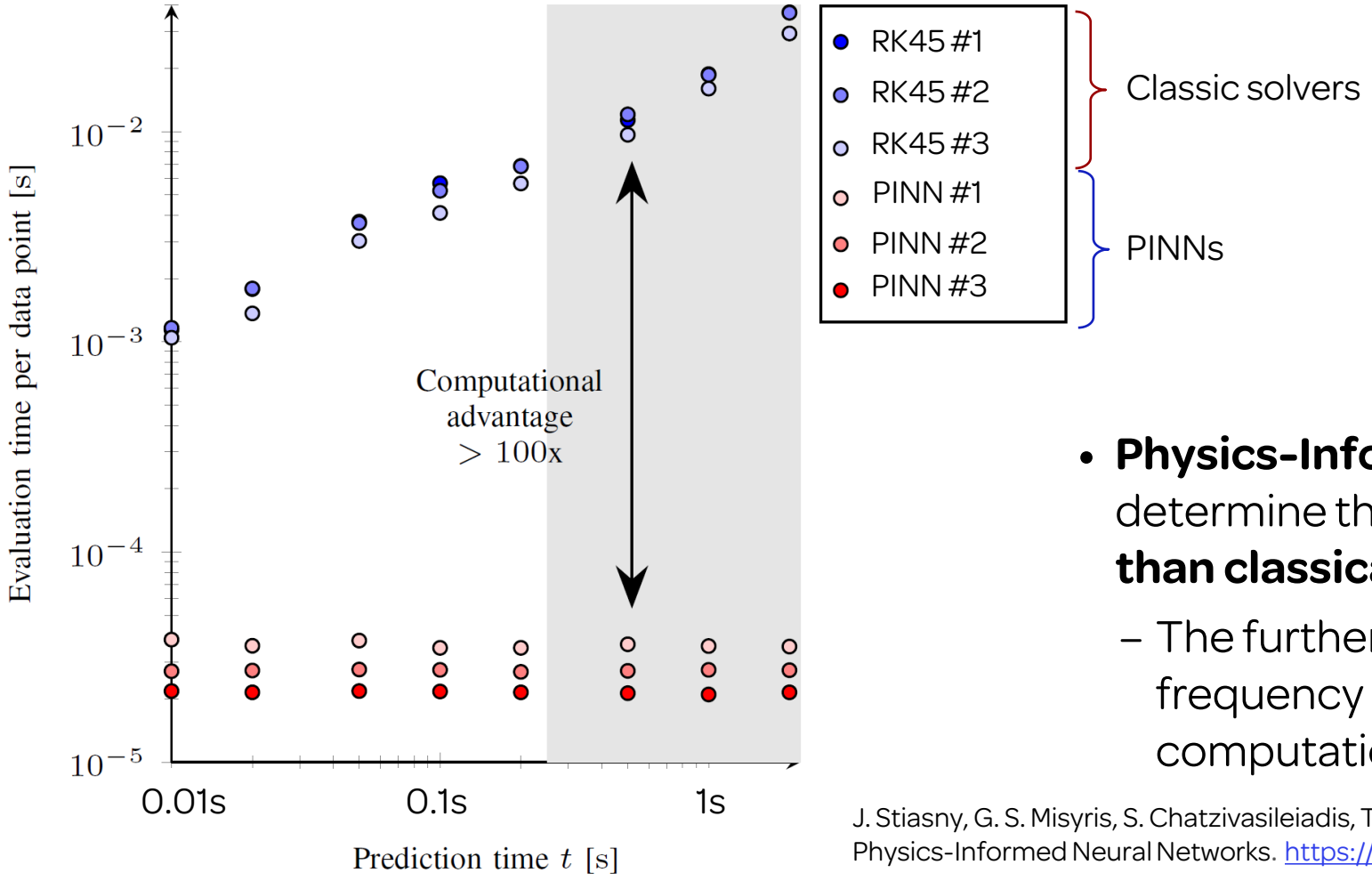
- Physics-Informed Neural Networks (PINN) could potentially replace solvers for systems of differential-algebraic equations in the long-term
  - **Probable power system application: Extremely fast screening of critical contingencies**
- In our example: PINN 87 times faster than ODE solver
- Can **directly estimate** the rotor angle at **any** time instant



Code is available on GitHub: https://github.com/jbesty

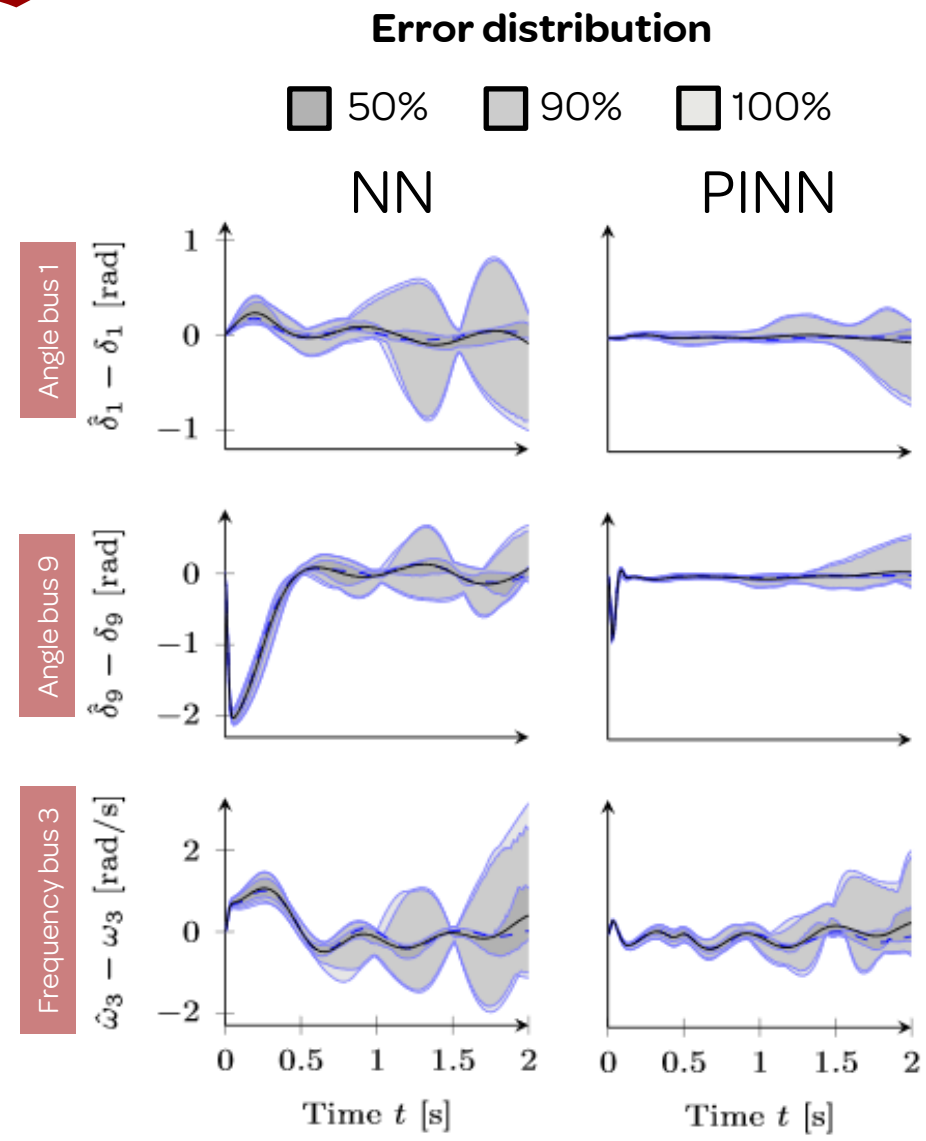
G. S. Misyris, A. Venzke, S. Chatzivasileiadis, Physics-Informed Neural Networks for Power Systems. Presented at the Best Paper Session of IEEE PES GM 2020. https://arxiv.org/pdf/1911.03737.pdf

Computation time: Classical numerical solvers vs. Physics-Informed NNs



- **Physics-Informed Neural Networks** can determine the outputs more than **100x faster than classical numerical solvers**
  - The further we look in time, e.g. what is the frequency at t=1s, the larger the computational advantage is

J. Stiasny, G. S. Misyris, S. Chatzivasileiadis, Transient Stability Analysis with Physics-Informed Neural Networks. https://arxiv.org/abs/2106.13638 [code]



Accuracy: Standard Neural Networks (NN) vs Physics-Informed NNs (PINN)

- **PINNs result in lower errors** than standard neural networks
- The error increases as we look further into the future
- PINNs can deliver an excellent screening tool, i.e. to very quickly assess if critical scenarios are secure or not.
  - To determine the exact numerical values, classic solvers are still very valuable

J. Stiasny, G. S. Misyris, S. Chatzivasileiadis, Transient Stability Analysis with Physics-Informed Neural Networks. https://arxiv.org/abs/2106.13638 [code]

Takeaway #8

Physics-informed neural networks exploit the underlying physics in the training procedure.

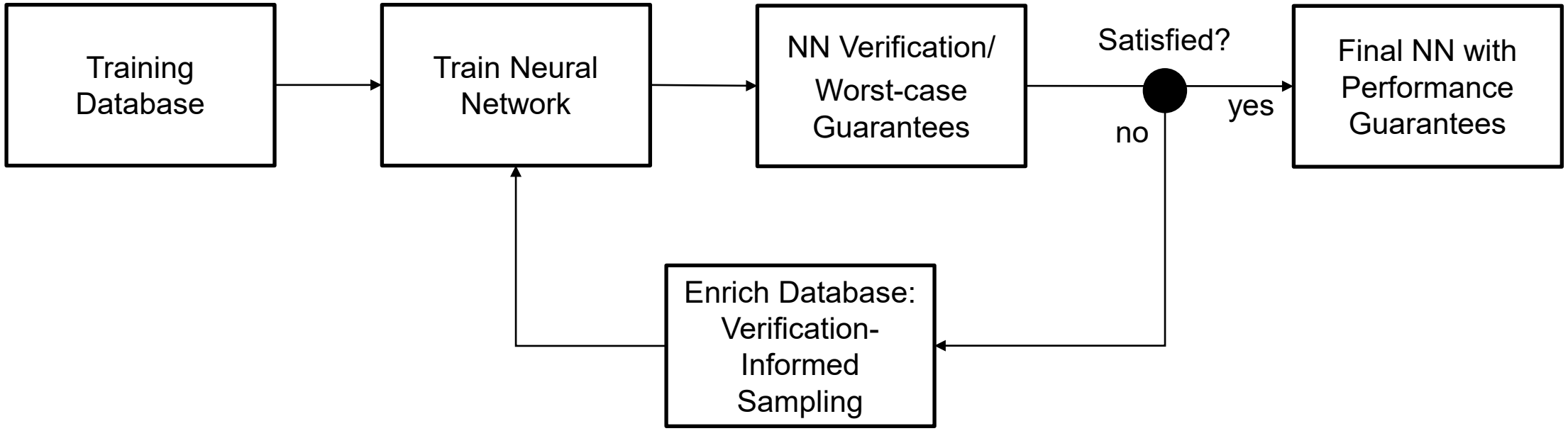
Sampling beyond statistics can yield high quality training databases with smaller amounts of data

"Data-centric AI movement"
(Andrew Ng, Stanford, and others)

"Small [data] is the new big" (IEEE Spectrum, Apr. 2022)

Exploit the prior knowledge

Closing the loop



J. Stiasny, S. Chevalier, R. Nellikkath, B. Sævarsson, S. Chatzivasileiadis. **Closing the Loop: A Framework for Trustworthy Machine Learning in Power Systems**. Accepted to *2022 iREP Symposium - Bulk Power System Dynamics and Control - XI (iREP)*. Banff, Canada. 2022. [paper | code]

Wrap-up

- Neural network verification (and worst-case guarantees) can **build the missing trust** for neural network applications in power systems

- Physics Informed Neural Networks can take advantage of the rich information about existing power system models inside neural network training

- This lecture:
  1. Sampling Beyond Statistics
  2. Verification of Classification Neural Networks for Power Systems
     - Application in Security Assessment
  3. Provable worst-case guarantees for Regression Neural Networks
     - Application in OPF (and probably in many other optimization programs)
  4. Physics-Informed Neural Networks for Power Systems

Ongoing work

Exploring a wide range of research directions

1. Contracting Neural-Newton Solver
   - Derive convergence guarantees for Neural Networks that can replace conventional Newton solvers [https://arxiv.org/pdf/2106.02543.pdf , L4DC 2022]

2. Physics-Informed Neural Networks for Fast Dynamic Security Assessment [https://arxiv.org/pdf/2106.13638.pdf, code: https://github.com/jbesty/PINNs_transient_stability_analysis]

3. Physics-Informed Neural Networks for OPF with worst-case guarantees [soon on ArXiV]

4. Using neural networks to capture previously intractable optimization constraints [https://arxiv.org/pdf/2103.17004.pdf , IREP 2022]

5. Accelerating MILPs: using Decision Trees to estimate the active set and drastically reduce the number of binary variables [https://arxiv.org/pdf/2010.06344.pdf , IEEE Trans. Power Systems]

   and others...

Thank you!

Spyros Chatzivasileiadis

Assoc. Prof, Head of Section

www.chatziva.com

spchatz@dtu.dk

- A. Venzke, S. Chatzivasileiadis. Verification of Neural Network Behaviour: Formal Guarantees for Power System Applications. Accepted at IEEE Trans. on Smartgrid. 2020. https://arxiv.org/pdf/1910.01624.pdf
- A. Venzke, G. Qu, S. Low, S. Chatzivasileiadis, Learning Optimal Power Flow: Worst-case Guarantees for Neural Networks. **Best Student Paper Award** at IEEE SmartGridComm 2020.[.pdf | slides | video]
- G. S. Misyris, A. Venzke, S. Chatzivasileiadis, Physics-Informed Neural Networks for Power Systems. Presented at the **Best Paper Session** of IEEE PES GM 2020. https://arxiv.org/pdf/1911.03737.pdf
- R. Nellikkath, S. Chatzivasileiadis, Physics-Informed Neural Networks for AC Optimal Power Flow https://arxiv.org/abs/2110.02672 [code]
- J. Stiasny, G. S. Misyris, S. Chatzivasileiadis, Transient Stability Analysis with Physics-Informed Neural Networks. https://arxiv.org/abs/2106.13638 [code]
- J. Stiasny, S. Chevalier, R. Nellikkath, B. Sævarsson, S. Chatzivasileiadis. **Closing the Loop: A Framework for Trustworthy Machine Learning in Power Systems**. Accepted to *2022 iREP Symposium - Bulk Power System Dynamics and Control - XI (iREP)*. Banff, Canada. 2022. [paper | code]

Article without any equations ☺

S. Chatzivasileiadis, A. Venzke, J. Stiasny and G. Misyris, **"Machine Learning in Power Systems: Is It Time to Trust It?,"** in *IEEE Power and Energy Magazine*, vol. 20, no. 3, pp. 32-41, May-June 2022 [.pdf]

All publications available at:

www.chatziva.com/publications.html

Some code available at:

www.chatziva.com/downloads.html

Appendix - Start

Decision Trees and Neural Networks for Optimal Power Flow: Capturing previously intractable security constraints

Main takeaway

Intractable/Non-linear Optimization Problem

$$\min_{\mathbf{x}} f(\mathbf{x})$$

linear bounds $\mathbf{x}_{\min} \leq \mathbf{x} \leq \mathbf{x}_{\max}$

linear constraints $\mathbf{A}\mathbf{x} = \mathbf{b}$

non-linear inequality constraints $\mathbf{g}(\mathbf{x}) \leq \mathbf{0}$

Intractable constraints $\phi(\mathbf{x}) \in \mathbf{S}$

e.g. based on differential equations, dynamic stability, etc

Main takeaway

DT: Decision Tree
NN: Neural Network

Intractable/Non-linear Optimization Problem

$$\min_{\mathbf{x}} f(\mathbf{x})$$

linear bounds $\mathbf{x}_{\min} \leq \mathbf{x} \leq \mathbf{x}_{\max}$

linear constraints $\mathbf{A}\mathbf{x} = \mathbf{b}$

non-linear inequality constraints $\mathbf{g}(\mathbf{x}) \leq \mathbf{0}$

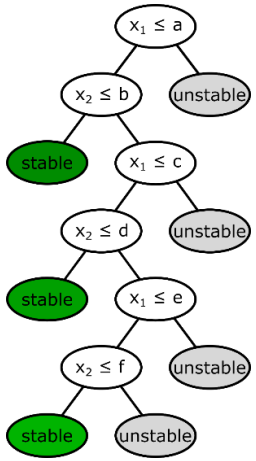
Intractable constraints

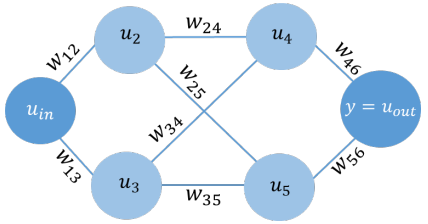
$\phi(\mathbf{x}) \in \mathbf{S}$

e.g. based on differential equations, dynamic stability, etc

Encode the feasible space to a DT or NN

Classify: feasible/infeasible





Main takeaway

DT: Decision Tree
NN: Neural Network
MILP: Mixed-Integer Linear Program

Intractable/Non-linear Optimization Problem

$$\min_{\mathbf{x}} f(\mathbf{x})$$

linear bounds $\mathbf{x}_{\min} \leq \mathbf{x} \leq \mathbf{x}_{\max}$

linear constraints $\mathbf{A}\mathbf{x} = \mathbf{b}$

non-linear inequality constraints $\mathbf{g}(\mathbf{x}) \leq \mathbf{0}$

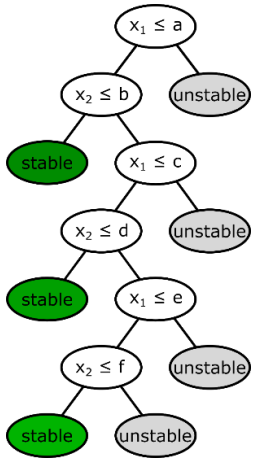
$\phi(\mathbf{x}) \in \mathbf{S}$

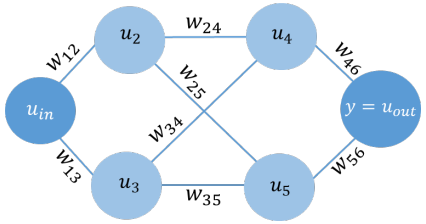
Intractable constraints

e.g. based on differential equations, dynamic stability, etc

Encode the feasible space to a DT or NN

Classify: feasible/infeasible





Exact Transformation: Convert DT or NN to a MILP

$$\min_{\mathbf{x}} f(\mathbf{x})$$

$$\mathbf{x}_{\min} \leq \mathbf{x} \leq \mathbf{x}_{\max}$$

$$\mathbf{A}\mathbf{x} = \mathbf{b}$$

MILP Constraints
(Exact transformation)

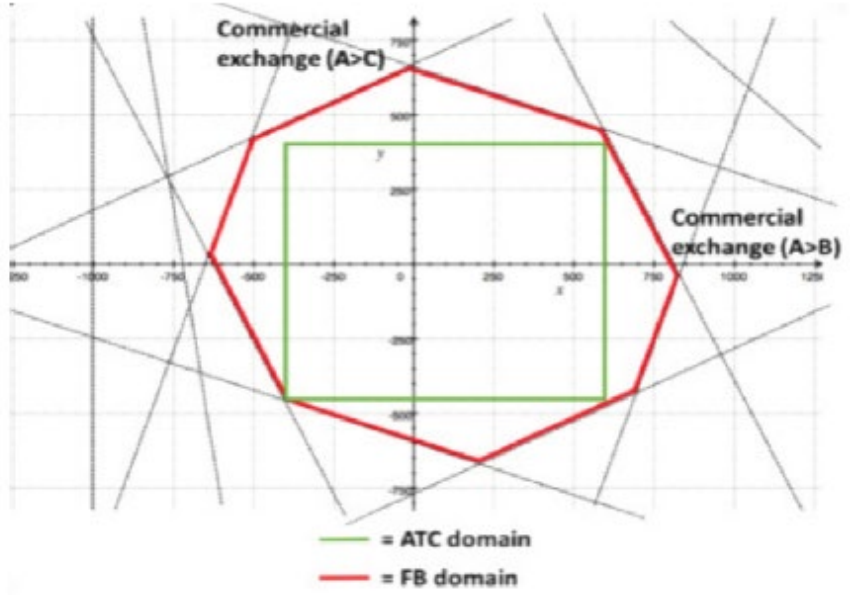
Capture previously intractable constraints and solve a MILP

Outline

- Guiding Example: Dynamic Stability Constrained Optimal Power Flow
- From Decision Trees to MILP
- From Neural Networks to MILP

Guiding example: What is the problem?

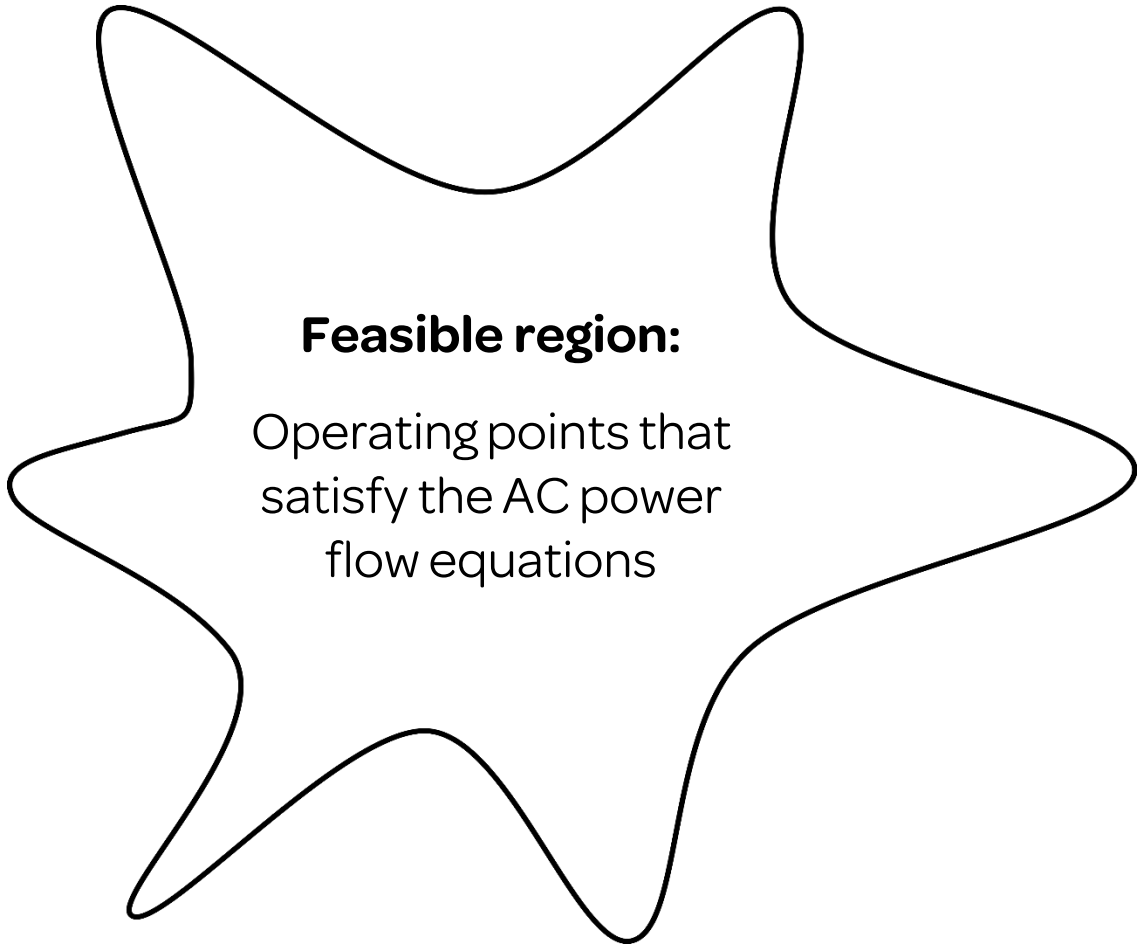
- **Power system optimization is primarily used in electricity markets,** and more recently for loss minimization and other functions
- The actual feasible region is non-convex (and very complex to identify it)
- **Electricity markets** consider the largest **convex** feasible region **and solve a MILP** (due to block offers, etc.)
- We are **missing** parts of the feasible region that can contain **"more optimal" points**
- **Goal:** find a **computationally tractable** way to consider the **actual feasible region in a MILP**



Largest Convex Region of the Feasible Space for Optimal Power Flow, for two types of Electricity Markets*

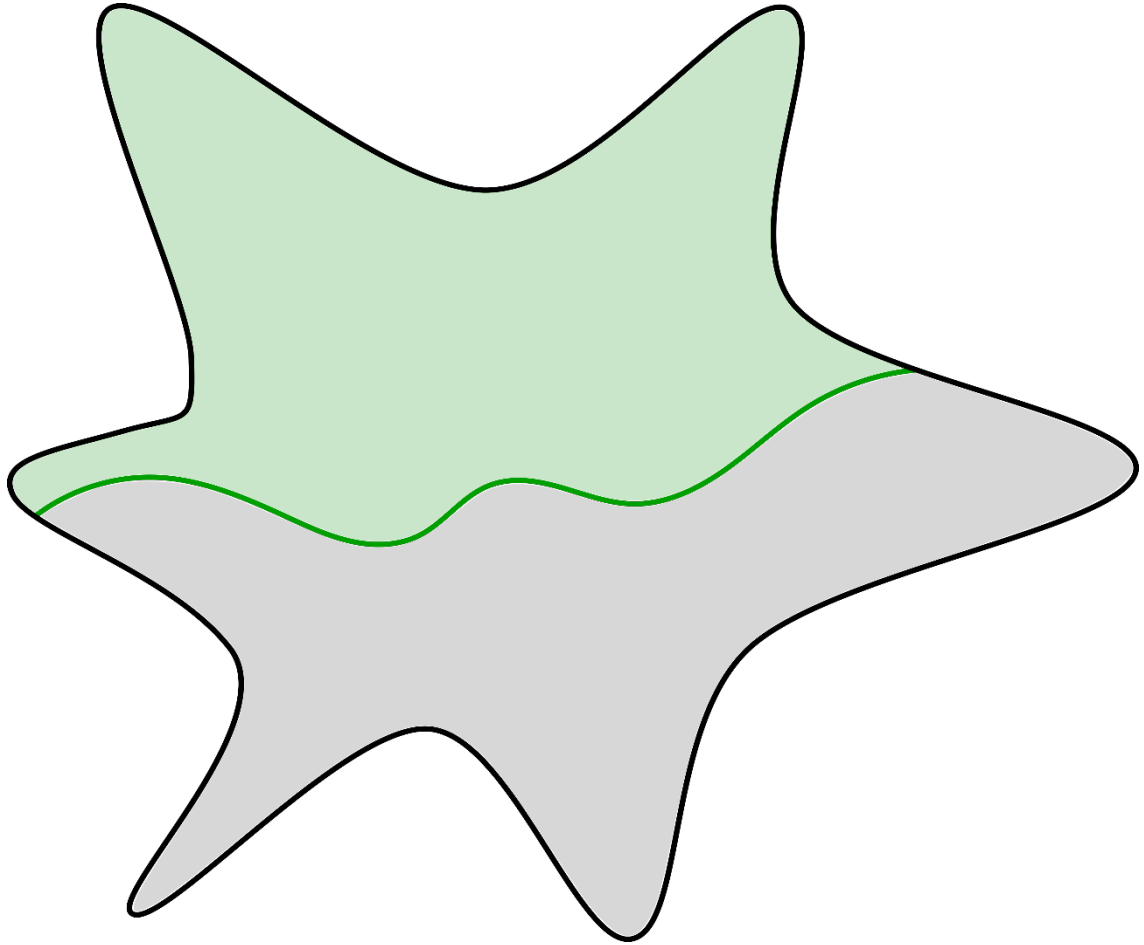
*KU Leuven Energy Institute, "EI Fact Sheet: Cross-border Electrcity Trading: Towards Flow-based Market Coupling," 2015. [Online]. Available: http://set.kuleuven.be/ei/factsheets

The safe operating region of power system operations



- Non-linear and non-convex AC power flow equations
- Component limits

The feasible space of power system operations

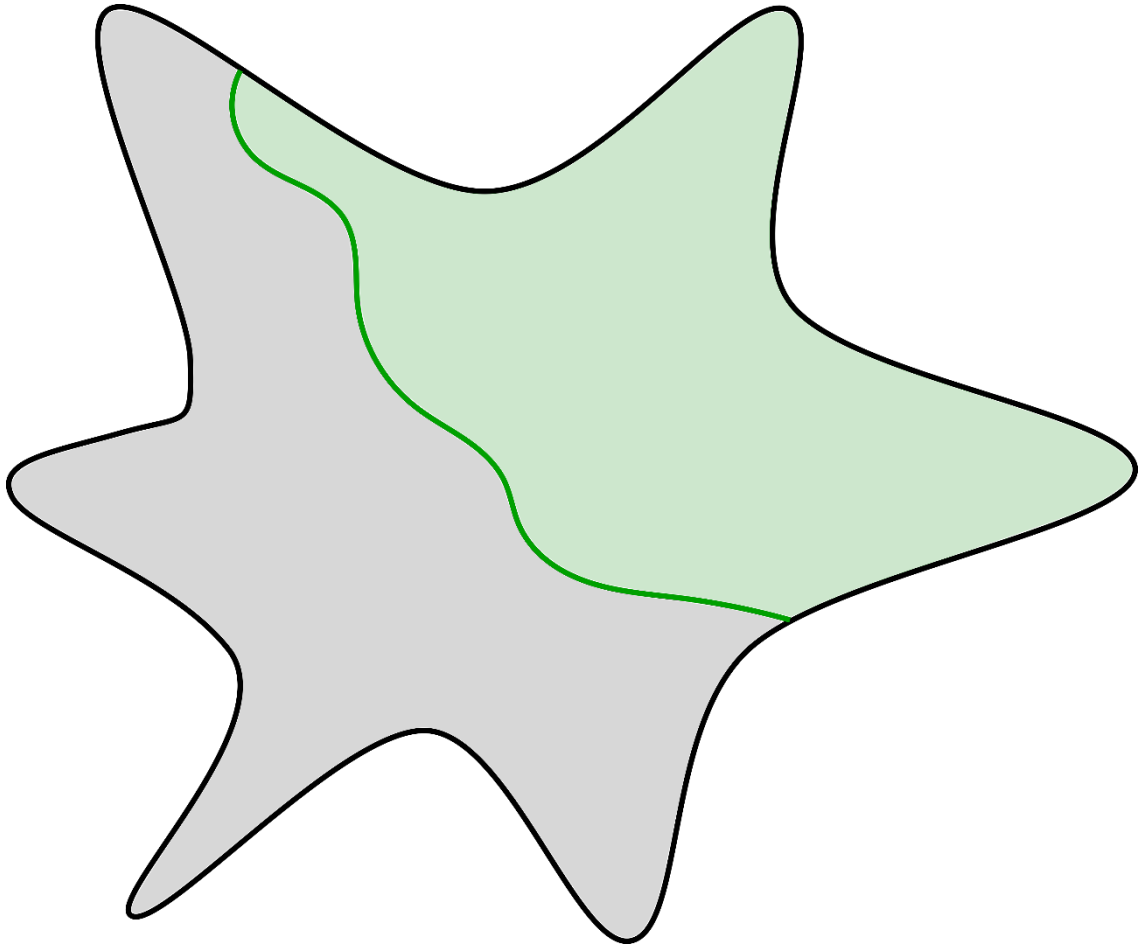


- Non-linear and non-convex AC power flow equations
- Component limits

+ N-1 security criterion

(non-linear algebraic inequality constraints)

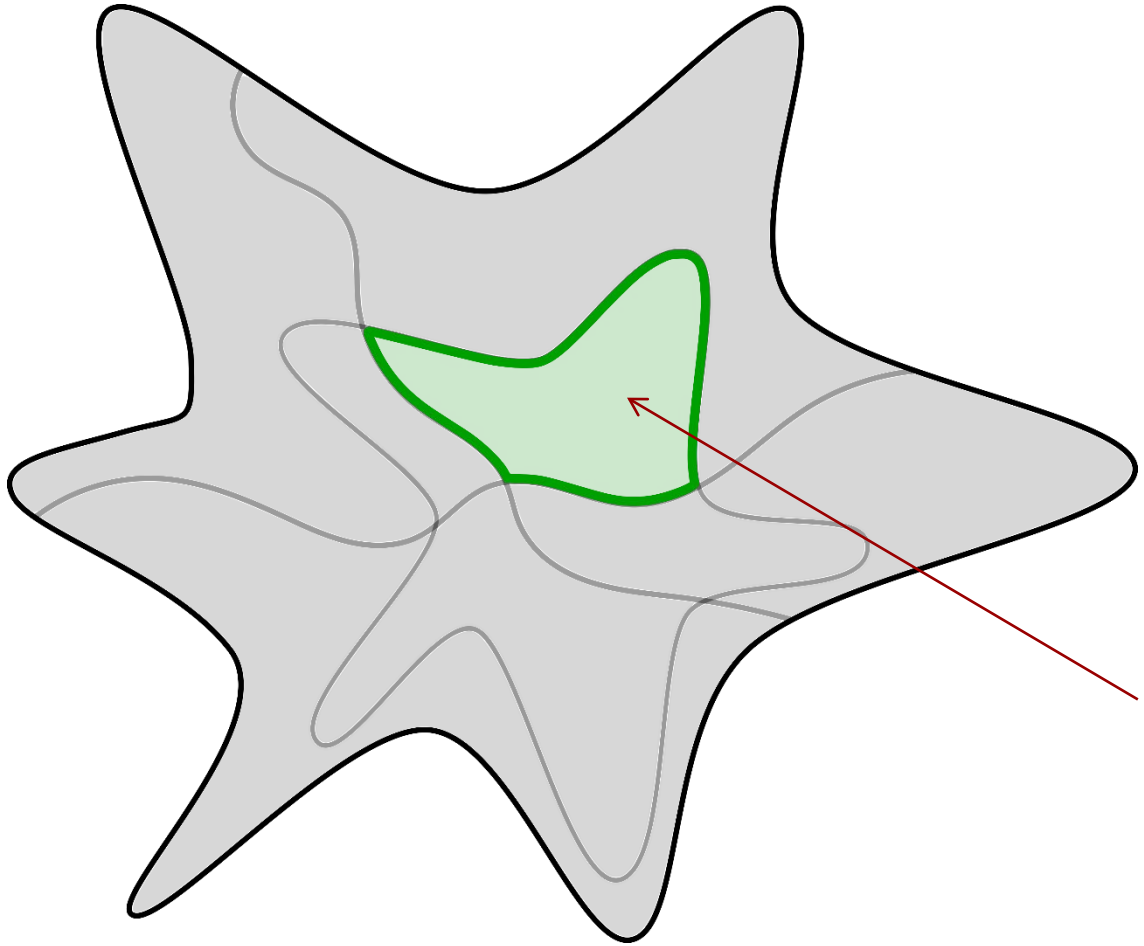
The feasible space of power system operations



- Non-linear and non-convex AC power flow equations
- Component limits

+ Stability Limits (inequality constraints based on differential equations)

The feasible space of power system operations



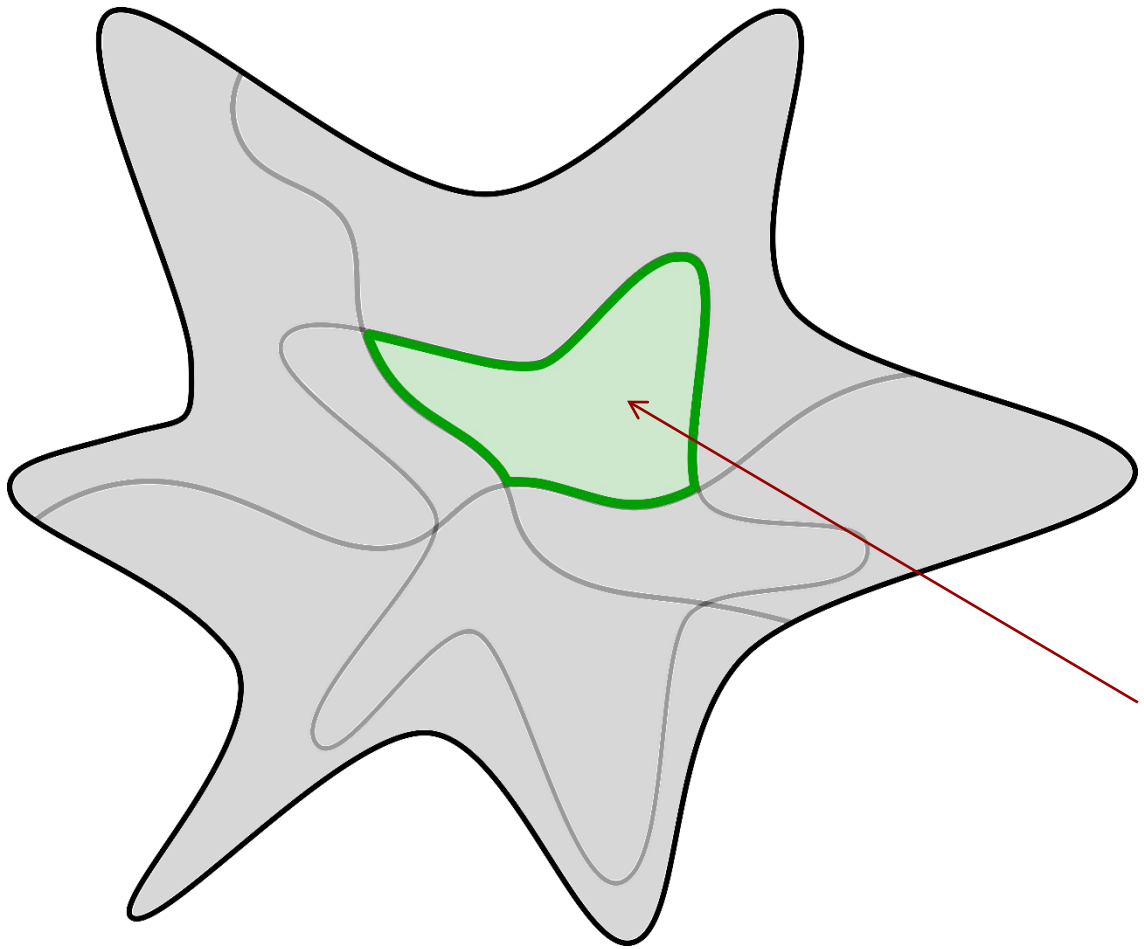
- Non-linear and non-convex AC power flow equations
- Component limits

+ N-1 security criterion (non-linear algebraic)

+ Stability Limits (differential equations)

Intersection of all security/stability criteria: Non-linear and non-convex security region

The feasible space of power system operations



Optimization constraints should represent this area

Impossible → differential and non-linear algebraic equations

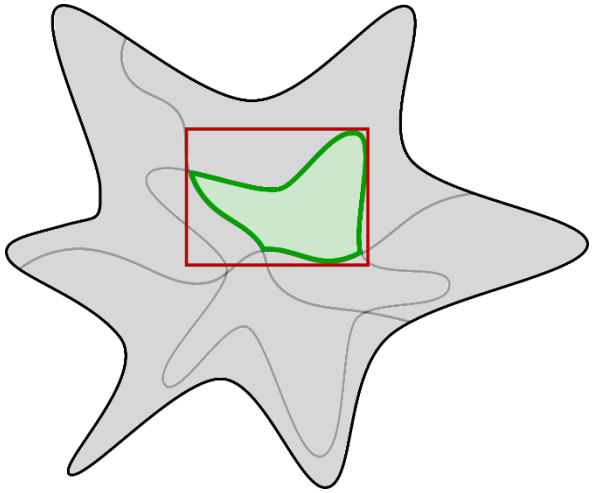
Intersection of all security/stability criteria: Non-linear and non-convex security region

What do TSOs and market operators do?

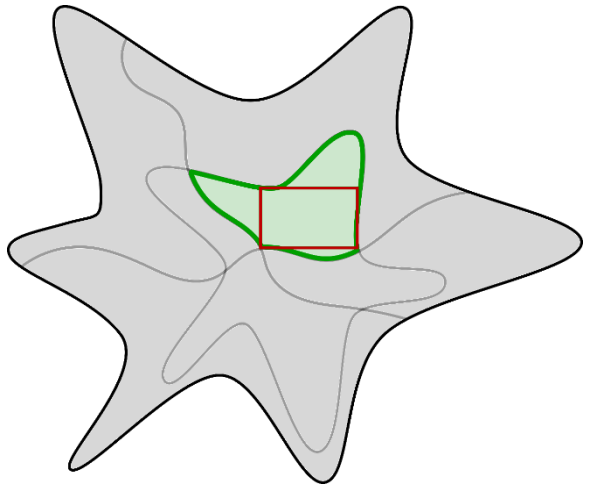
Linear approximations

Net Transfer Capacity[1]

Inaccurate



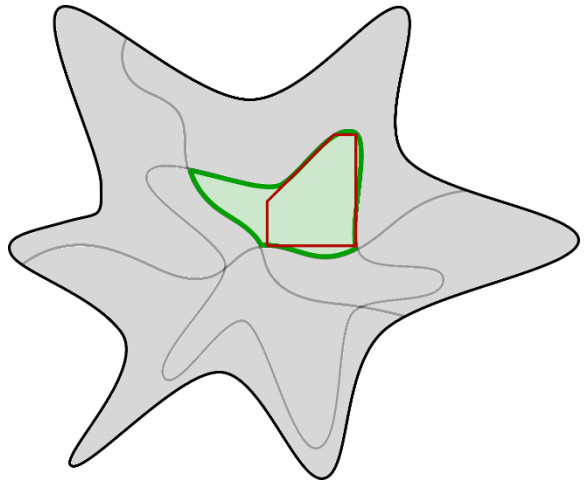
Too conservative



[1]e.g. Nordic Electricity Market

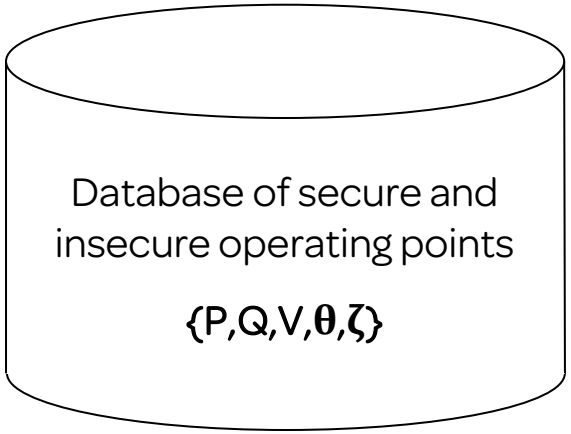
Flow-based market coupling[2]

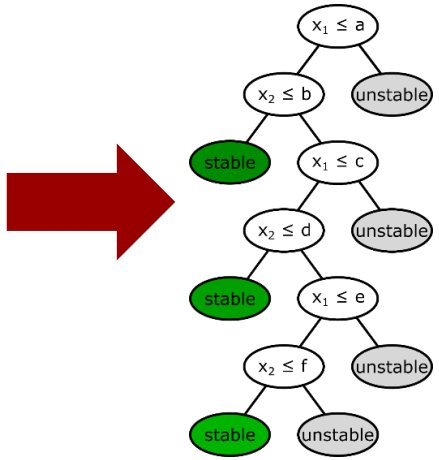
Single convex region

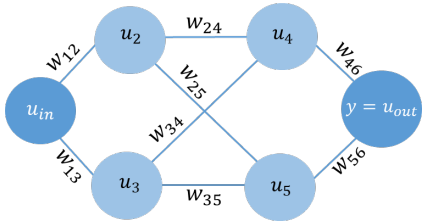


[2]e.g. Central European Market

Our proposal: Data-driven Security Constrained OPF
How does it work?







$$\mathbf{PTDF} \cdot \left(\mathbf{P_G} - \mathbf{P_D}\right) \leq \mathbf{F}_{\mathbf{L,p}}^{\mathbf{max}} y_p + \mathbf{F}_{\mathbf{L}}^{\mathbf{max}}(1 - y_p)$$
$$\mathbf{PTDF} \cdot \left(\mathbf{P_G} - \mathbf{P_D}\right) \geq \mathbf{F}_{\mathbf{L,p}}^{\mathbf{min}} y_p - \mathbf{F}_{\mathbf{L}}^{\mathbf{max}}(1 - y_p)$$

Operating points provided by the TSOs through simulated and real data

Train a decision tree or neural network to classify secure and insecure regions

Exact reformulation to MILP

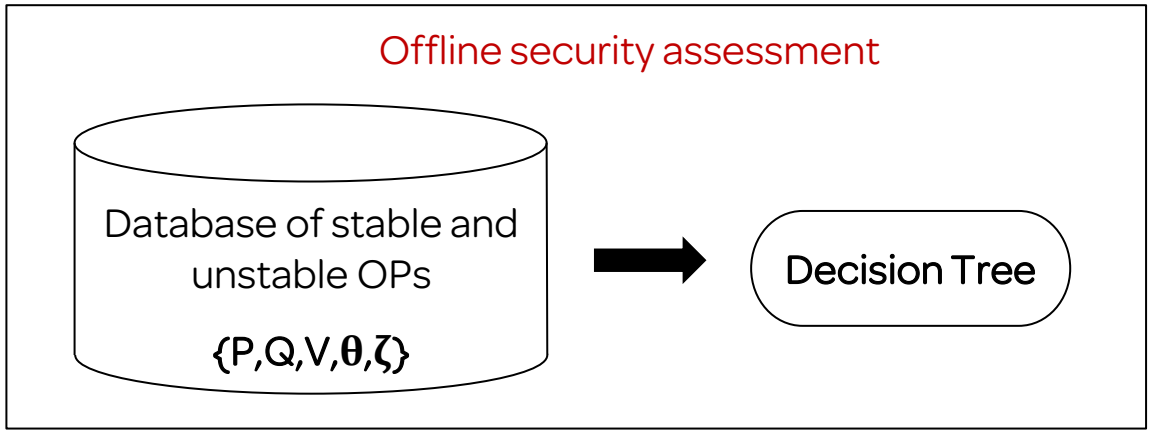
A. Venzke, D. T. Viola, J. Mermet-Guyennet, G. S. Misyris, S. Chatzivasileiadis. Neural Networks for Encoding Dynamic Security-Constrained Optimal Power Flow to Mixed-Integer Linear Programs. 2020. https://arxiv.org/pdf/2003.07939.pdf

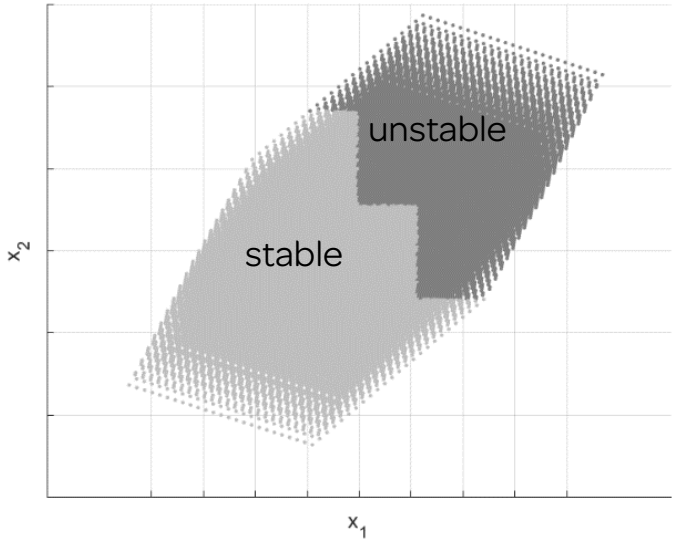
L. Halilbašić, F. Thams, A. Venzke, S. Chatzivasileiadis, and P. Pinson, "Data-driven security-constrained AC-OPF for operations and markets," *PSCC* 2018. [.pdf]

F. Thams, L. Halilbašić, P. Pinson, S. Chatzivasileiadis, and R. Eriksson, "Data-driven security-constrained OPF," *IREP* 2017. [.pdf]

From decision trees to mixed integer programming

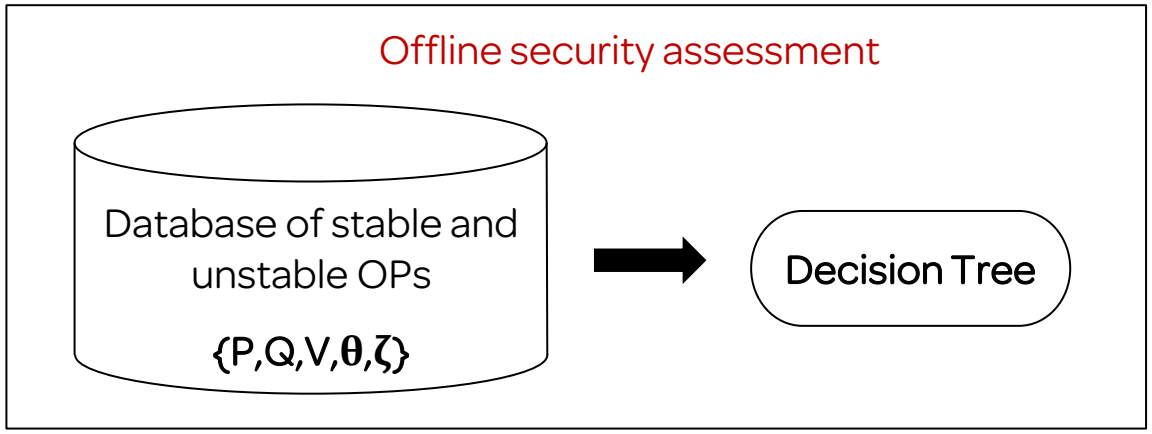
Data-driven security-constrained OPF



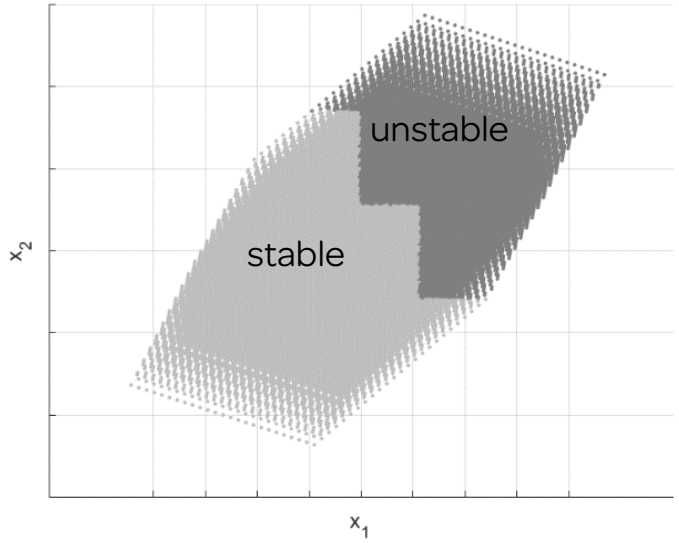


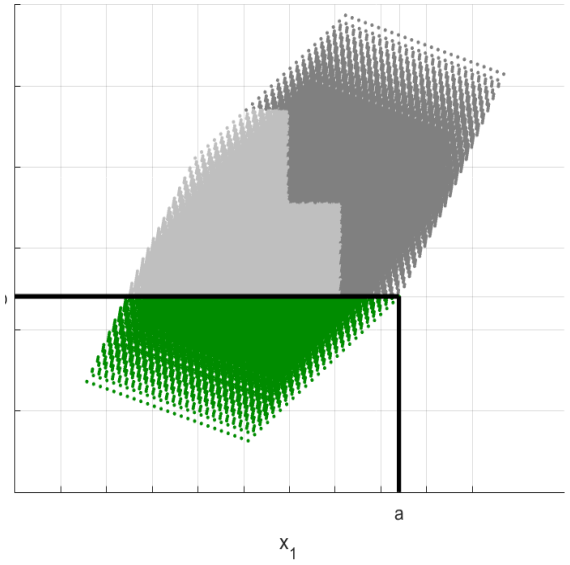
Thams et al IREP 2017,
Halilbasic et al PSCC 2018

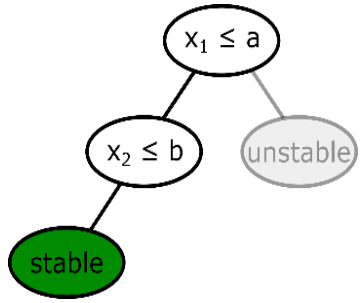
Data-driven security-constrained OPF



Partitioning the secure operating region

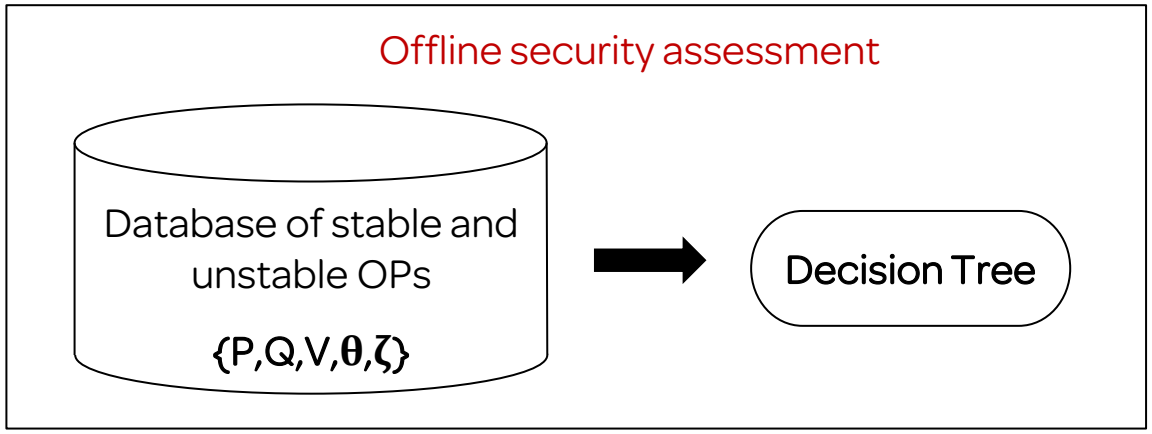




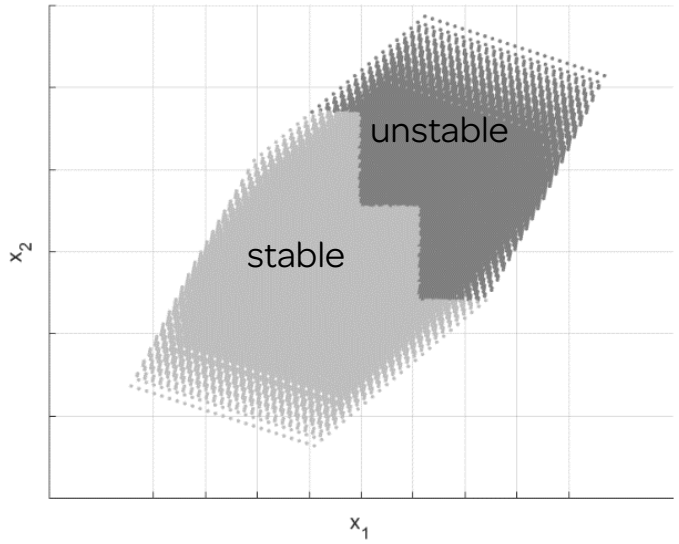


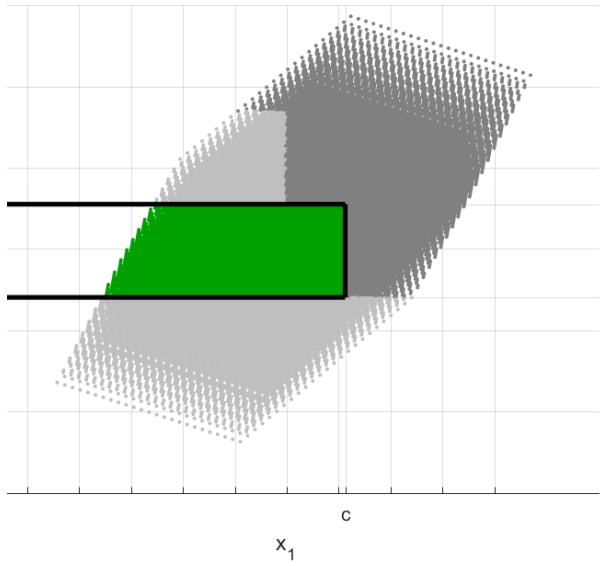
Thams et al IREP 2017,
Halilbasic et al PSCC 2018

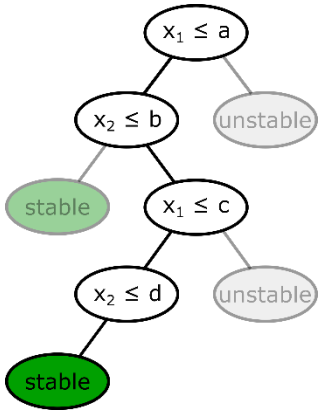
Data-driven security-constrained OPF



Partitioning the secure operating region

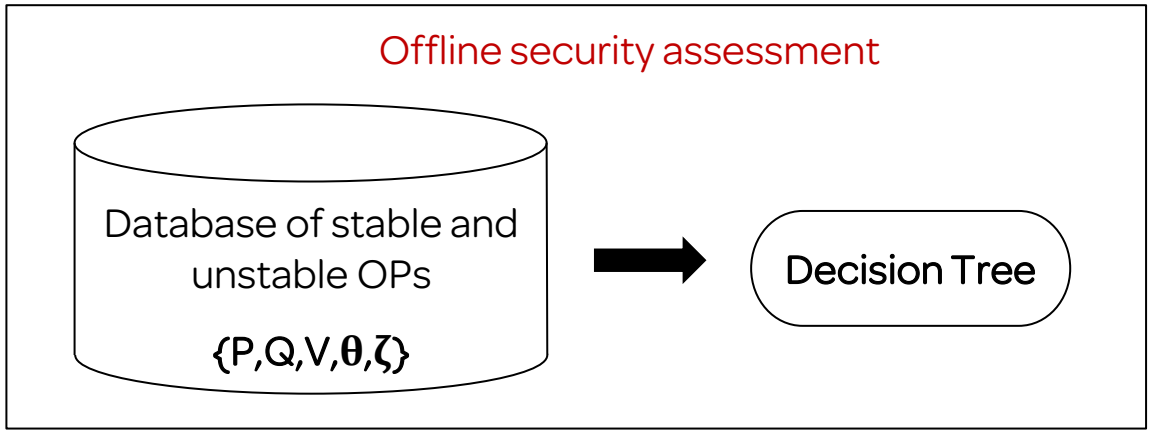




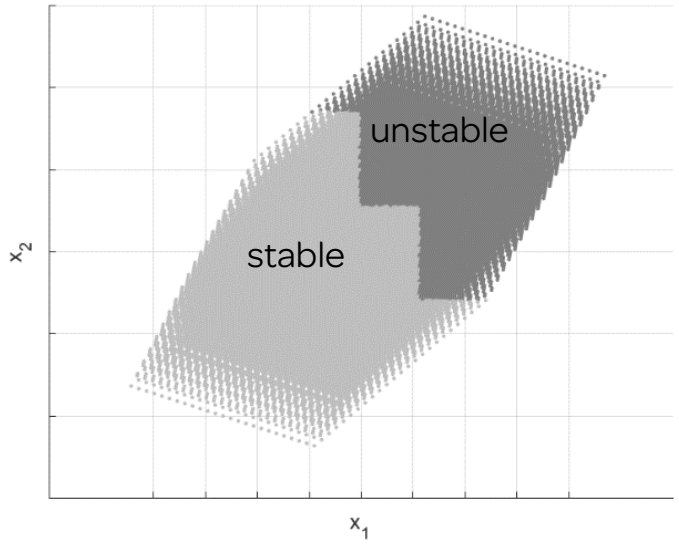


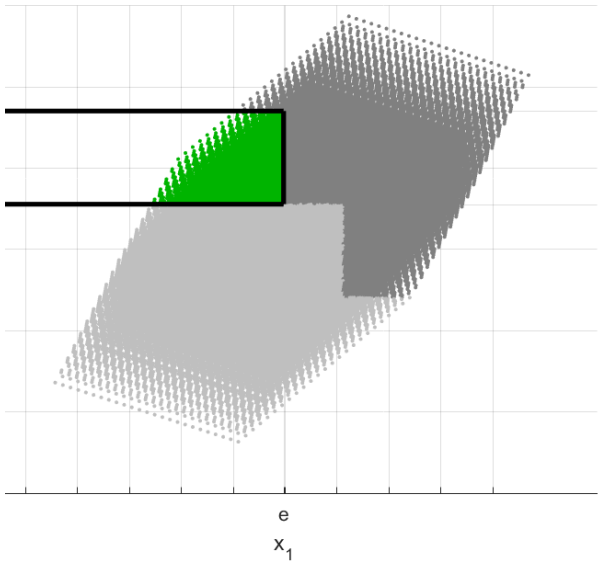
Thams et al IREP 2017,
Halilbasic et al PSCC 2018

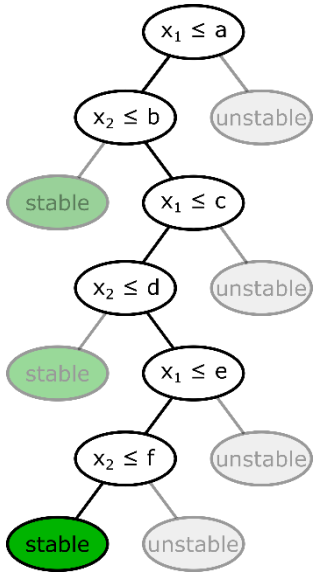
Data-driven security-constrained OPF



Partitioning the secure operating region







Thams et al IREP 2017,
Halilbasic et al PSCC 2018

Data-driven security-constrained OPF

Offline security assessment

Database of stable and unstable OPs

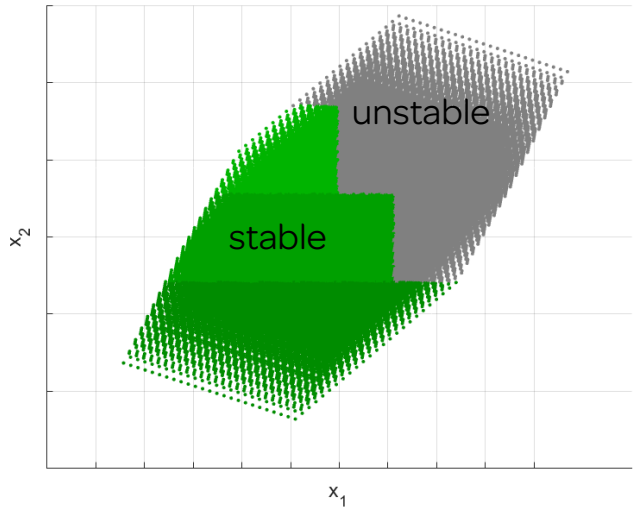
$\{P,Q,V,\theta,\zeta\}$

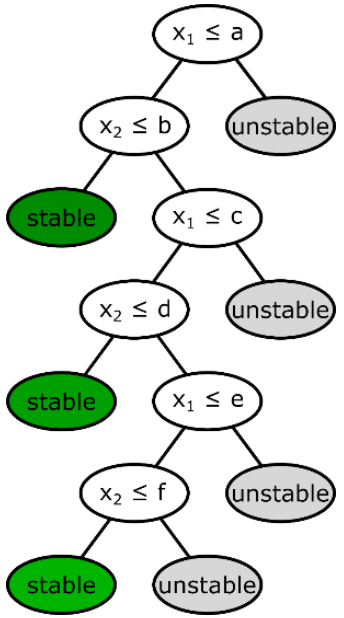
→ **Decision Tree**

Optimization

Integer Programming to incorporate partitions (DT)

- DC-OPF (MILP)
- AC-OPF (MINLP)
- Relaxation (MIQCP, MISOCP)

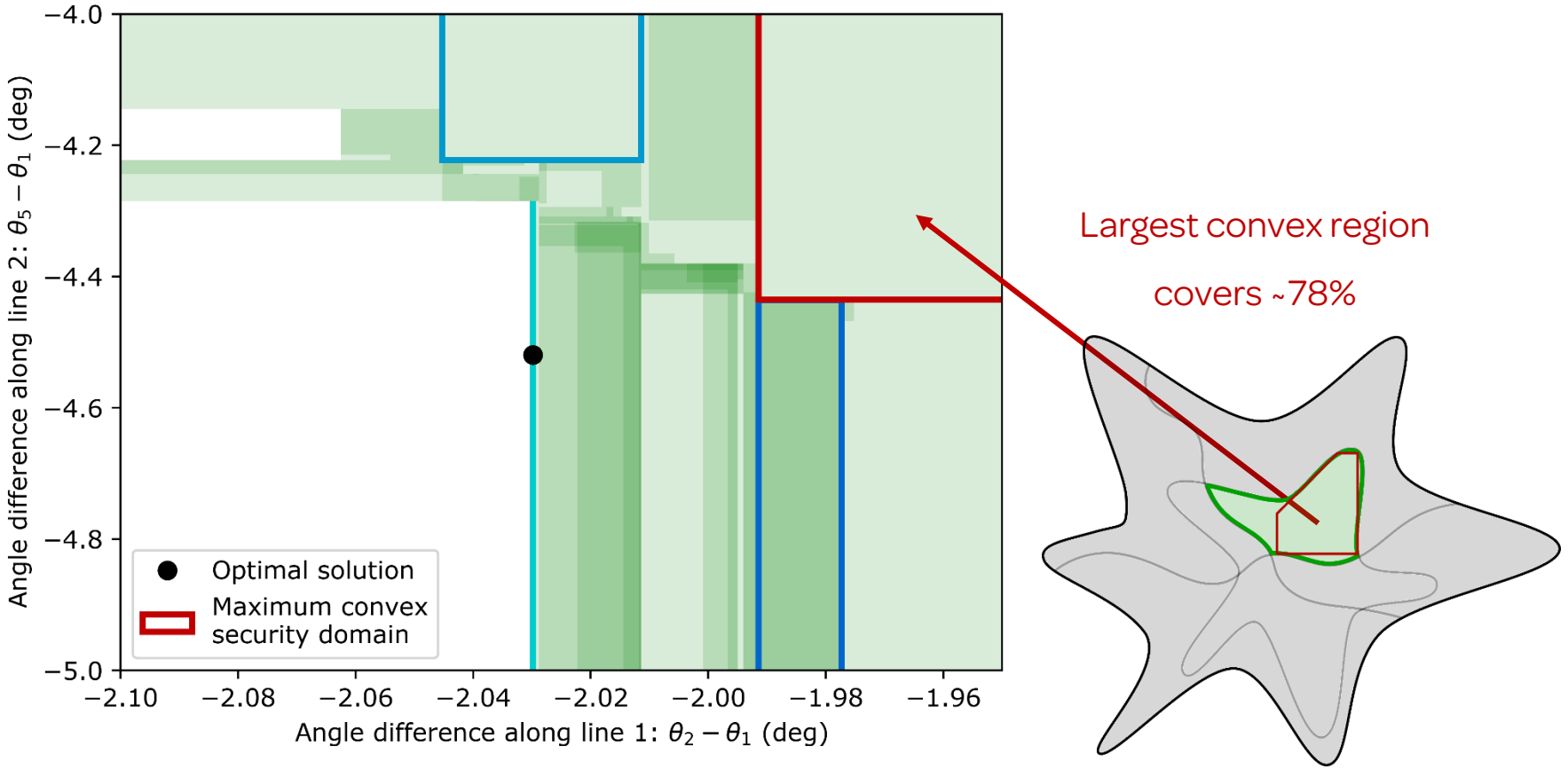




- Each leaf is a convex region
- Flow-based market coupling corresponds to the leaf that maps the largest convex region

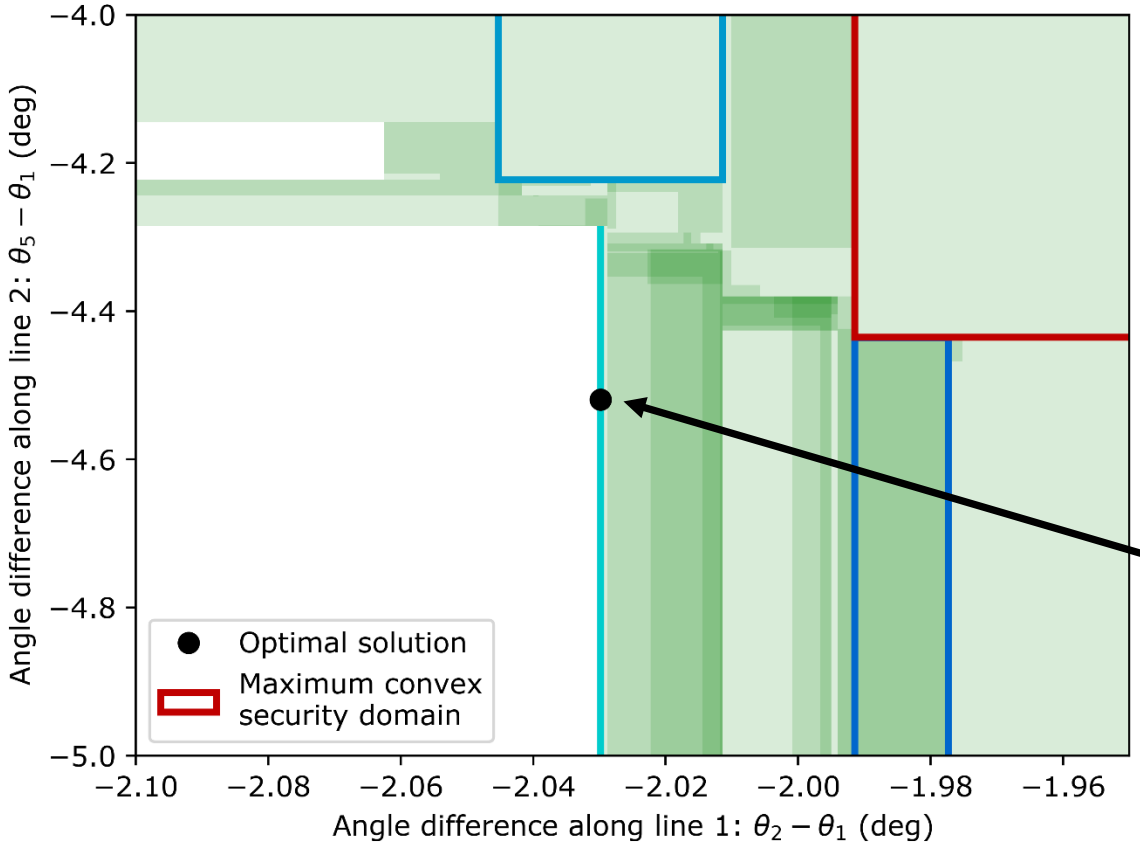
Thams et al IREP 2017,
Halilbasic et al PSCC 2018

We gain ~22% of the feasible space using data and Mixed Integer Programming



L. Halilbašić, F. Thams, A. Venzke, S. Chatzivasileiadis, and P. Pinson, "Data-driven security-constrained AC-OPF for operations and markets," *PSCC* 2018. [.pdf]

MISOCP finds better solutions than nonconvex problem!

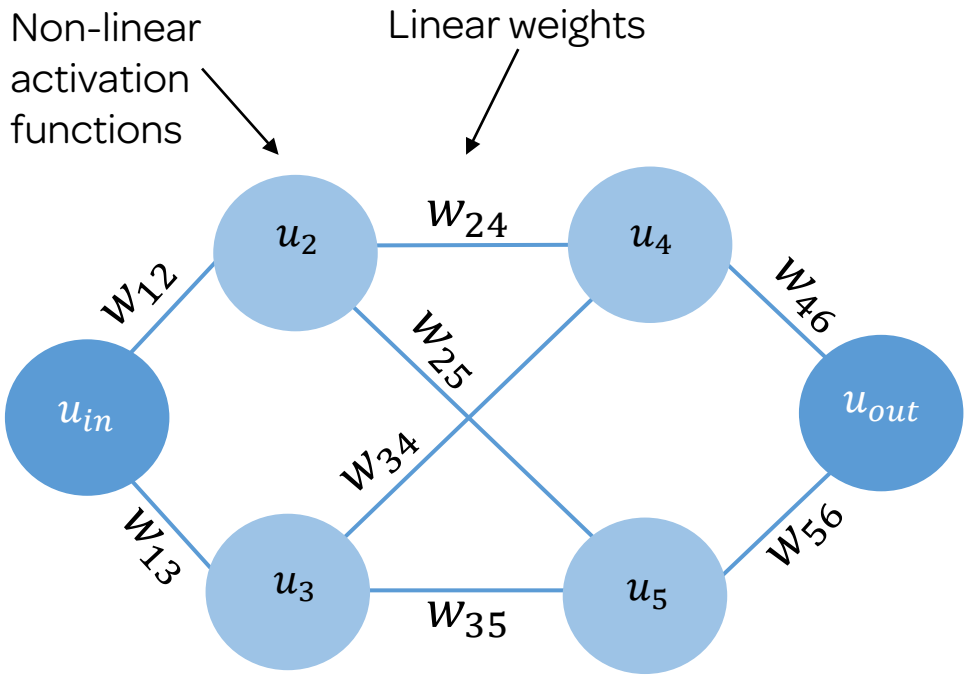


Optimum located at boundary of considered security region

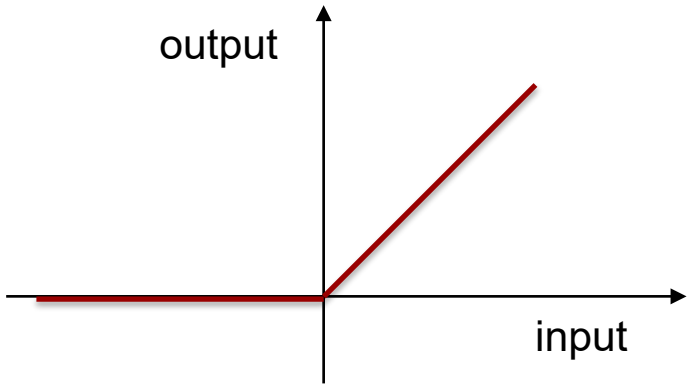
L. Halilbašić, F. Thams, A. Venzke, S. Chatzivasileiadis, and P. Pinson, "Data-driven security-constrained AC-OPF for operations and markets," *PSCC*2018. [.pdf]

From Neural Networks to Mixed Integer Linear Programming

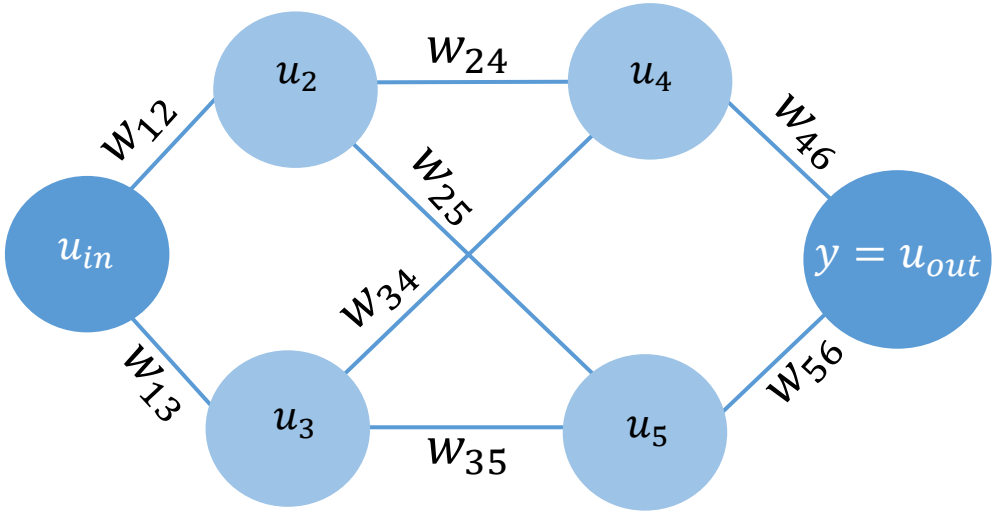
From Neural Networks to Mixed-Integer Linear Programming



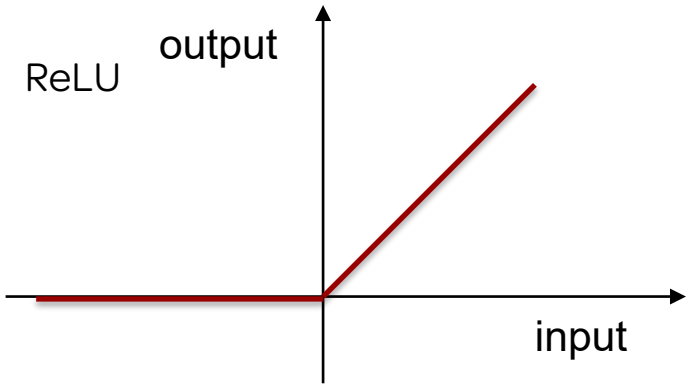
- Most usual activation function: ReLU
- ReLU: Rectifier Linear Unit



From Neural Networks to Mixed-Integer Linear Programming

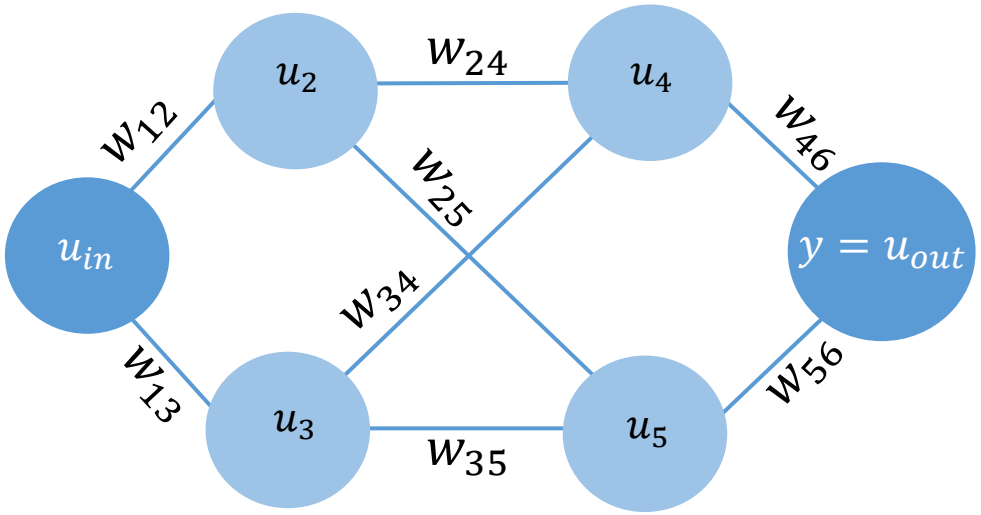


- Linear weights
- On every node: a non-linear activation function
  - ReLU: $\boldsymbol{u_j = \max\left(0, w_{ij} u_i + b_i\right)}$
- But ReLU can be transformed to a piecewise linear function with binaries



MILP

From Neural Networks to Mixed-Integer Linear Programming

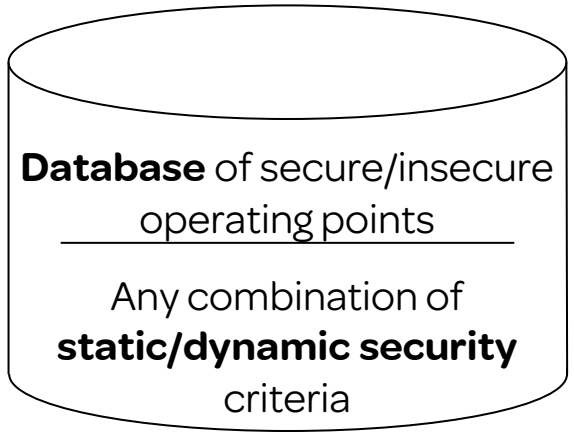


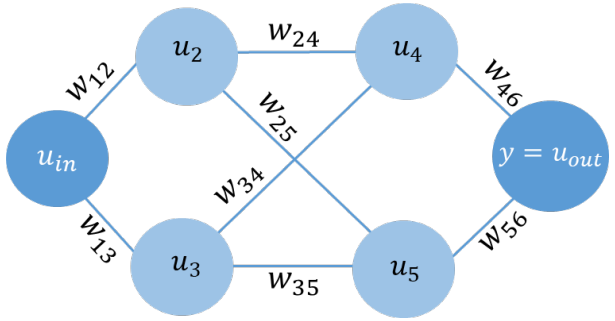
- Output
  - Binary classification: feasible/infeasible
  - ReLU is the most common activation function for Deep Neural Networks
  - Output vector y with two elements:

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

- $y_1 \geq y_2$: safe
- $y_2 \geq y_1$: unsafe

Data-driven Security Constrained OPF
How does it work?





Tractable small-signal stability-constrained OPF

$$\min f(\mathbf{p_g})$$
$$\text{s.t.}$$
$$\mathbf{p_g^{min}} \leq \mathbf{p_g} \leq \mathbf{p_g^{max}}$$
$$\mathbf{v_g^{min}} \leq \mathbf{v_g} \leq \mathbf{v_g^{max}}$$
$$\mathbf{s}_{\text{balance}}(\mathbf{p}^0, \mathbf{q}^0, \mathbf{v}^0, \boldsymbol{\theta}^0) = \mathbf{0}$$

NN → MILP

$$\hat{\mathbf{u}}_\mathbf{k} = \mathbf{W}_k \mathbf{u}_{\mathbf{k-1}} + \mathbf{b}_\mathbf{k}$$
$$\mathbf{u}_k = \max(\hat{\mathbf{u}}_k, 0) \Rightarrow \begin{cases} y_k \leq \hat{u}_k - \hat{u}_k^{\min}(1 - b_k) \\ u_k \geq \hat{u}_k \\ u_k \leq \hat{u}_k^{\max} b_k \\ u_k \geq 0 \\ b_k \in \{0,1\}^{N_k} \end{cases}$$
$$\mathbf{y} = \mathbf{u}_{out}$$
$$y_1 \geq y_2$$

e.g. N-1 & Small-signal stability (Small-Signal Stab. up to now impossible to *directly* include in an OPF)

Train a neural network → "encode" all information about secure and insecure regions

Exact reformulation to MILP

A. Venzke, D. T. Viola, J. Mermet-Guyennet, G. S. Misyris, S. Chatzivasileiadis. Neural Networks for Encoding Dynamic Security-Constrained Optimal Power Flow to Mixed-Integer Linear Programs. 2020. https://arxiv.org/pdf/2003.07939.pdf

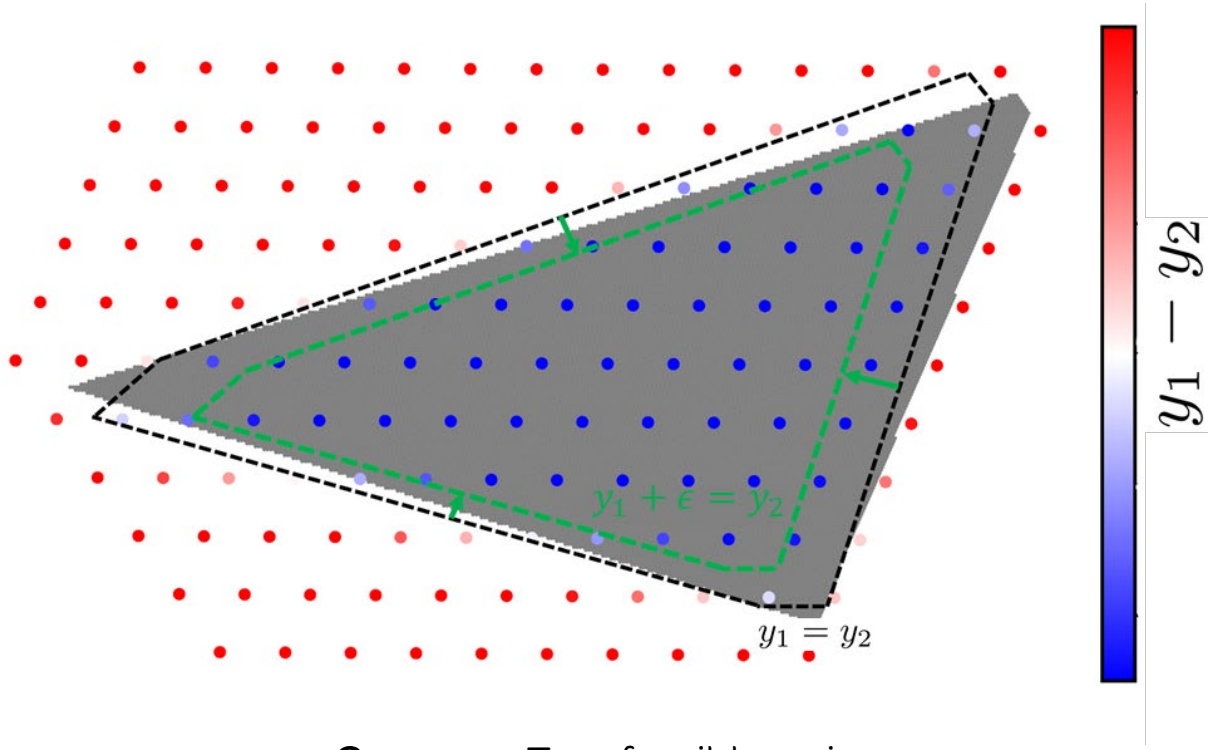
Code available: https://gitlab.com/violatimon/power_system_database_generation

Challenges

1. How do you ensure that the feasible region is captured accurately by the NN?

2. How do you handle the non-linear **equality** constraints?

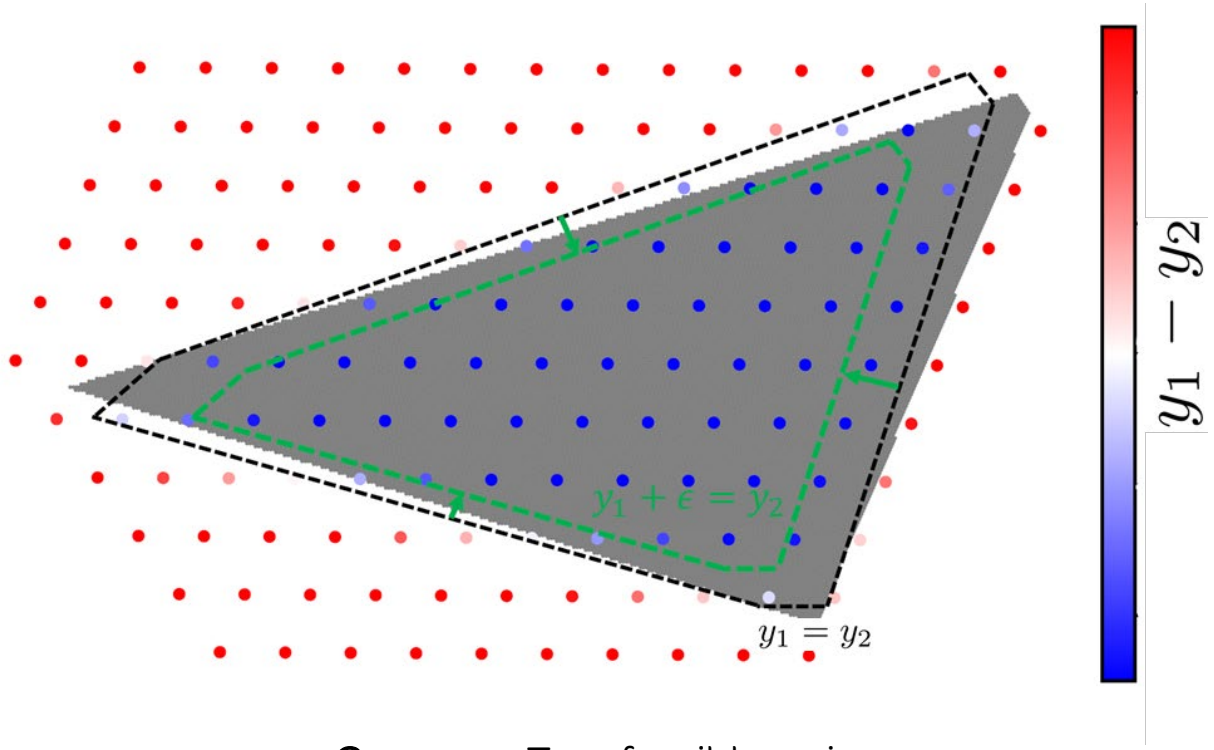
Challenge #1: Guiding NN to accurately capture the (previously intractable) feasible region



Grey area: True feasible region
Black dashed line: Original NN estimate
Green dashed line: ϵ-conservativeness

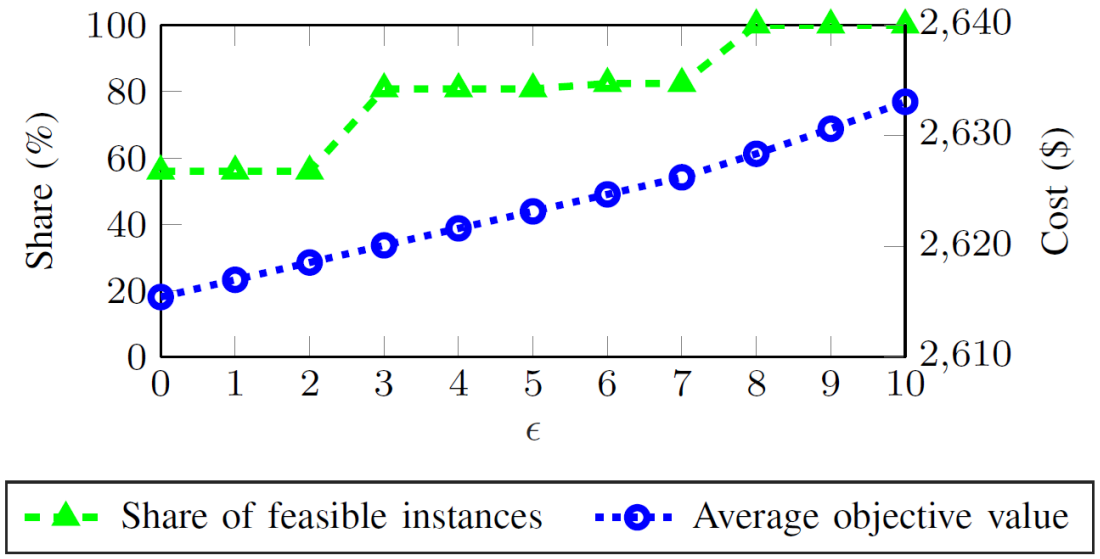
- Increase conservativeness:
  Replace $y_1 \geq y_2$ with $y_1 \geq y_2 + \epsilon$

Challenge #1: Guiding NN to accurately capture the (previously intractable) feasible region



Grey area: True feasible region
Black dashed line: Original NN estimate
Green dashed line: ϵ-conservativeness

- Increase conservativeness:
  Replace $y_1 \geq y_2$ with $y_1 \geq y_2 + \epsilon$



Challenge #2:
Handling the non-linear equality constraints

- The problem: $\mathbf{s}_{\text{balance}}(\mathbf{p}^0, \mathbf{q}^0, \mathbf{v}^0, \boldsymbol{\theta}^0) = \mathbf{0}$

quadratic constraints
(#constraints = #nodes)

- Three solution options to avoid solving a MINLP:
  a. Train a **Regression Neural Network** to estimate q^0, θ^0 from p^0, v^0 and insert it as a list of mixed-integer linear constraints to the problem
  b. **Convexify**, if possible, and solve a MISOCP; recover feasible (global?) optimal
  c. **Linearize** the non-linear equations and solve iteratively the MILP

Challenge #2: Handling the non-linear equality constraints

- The problem: $\mathbf{s}_{\text{balance}}(\mathbf{p}^0, \mathbf{q}^0, \mathbf{v}^0, \boldsymbol{\theta}^0) = \mathbf{0}$

quadratic constraints
(#constraints = #nodes)

- Three solution options to avoid solving a MINLP:
  a. Train a **Regression Neural Network** to estimate q^0, θ^0 from p^0, v^0 and insert it as a list of mixed-integer linear constraints to the problem
  b. **Convexify**, if possible, and solve a MISOCP; recover feasible (global?) optimal
  c. **Linearize** the non-linear equations and solve iteratively the MILP

- **Here: linearization**
  - Replace N constraints with 1 linearized constraint of the total active power nodal balance
  - **Iterative MILP converges very fast : 1.04 iterations on average in 125 instances**

$$\sum_{\mathcal{G}} \mathbf{p}_{\mathbf{g}}^{\mathbf{0}} + \sum_{\mathcal{N}} \mathbf{p}_{\mathbf{d}}^{\mathbf{0}} + p_{\text{losses}}|_i + \frac{\delta p_{\text{losses}}}{\delta \mathbf{p}_{\mathbf{g}}^{\mathbf{0}}}|_i (\mathbf{p}_{\mathbf{g}}^{\mathbf{0}} - \mathbf{p}_{\mathbf{g}}^{\mathbf{0}}|_i) + \frac{\delta p_{\text{losses}}}{\delta \mathbf{v}_{\mathbf{g}}^{\mathbf{0}}}|_i (\mathbf{v}_{\mathbf{g}}^{\mathbf{0}} - \mathbf{v}_{\mathbf{g}}^{\mathbf{0}}|_i) = 0$$

Results: average over 125 instances

w.r.t. N-1 and small-signal stability

	Problem formulation	Problem type	Generation cost ($)	Solver time (s)	Share of feasible instances (%)
Conventional non-linear program (interior-point)	AC-OPF	NLP	2425.94 (0.0%)	0.04	35.2
	+ N-1 security	NLP	2565.13 (5.7%)	0.15	35.2
Neural Networks → MILP	+ small-signal stability ($\epsilon = 0$)	MILP	2615.43 (7.8%)	0.22	56.0
	+ small-signal stability ($\epsilon = 8$)	MILP	2628.37 (8.3%)	0.12	100.0

Results: average over 125 instances

w.r.t. N-1 and small-signal stability

	Problem formulation	Problem type	Generation cost ($)	Solver time (s)	Share of feasible instances (%)
Conventional non-linear program (interior-point)	AC-OPF	NLP	2425.94 (0.0%)	0.04	35.2
	+ N-1 security	NLP	2565.13 (5.7%)	0.15	35.2
Neural Networks → MILP	+ small-signal stability ($\epsilon = 0$)	MILP	2615.43 (7.8%)	0.22	56.0
	+ small-signal stability ($\epsilon = 8$)	MILP	2628.37 (8.3%)	0.12	100.0

MILP solution time similar to NLP; and MILP contains constraints that are intractable for the NLP

*ACOPF is faster because it contains less than 15% of the constraints of the other problems

Results: average over 125 instances

w.r.t. N-1 and small-signal stability

	Problem formulation	Problem type	Generation cost ($)	Solver time (s)	Share of feasible instances (%)
Conventional non-linear program (interior-point)	AC-OPF	NLP	2425.94 (0.0%)	0.04	35.2
	+ N-1 security	NLP	2565.13 (5.7%)	0.15	35.2
Neural Networks → MILP	+ small-signal stability ($\epsilon = 0$)	MILP	2615.43 (7.8%)	0.22	56.0
	+ small-signal stability ($\epsilon = 8$)	MILP	2628.37 (8.3%)	0.12	100.0

Conservativeness ($\epsilon = 8$) helps obtain feasible solutions without increasing substantially the objective function

Takeaway #9

Both Decision Trees and Neural Networks can convert to a MILP and help capture optimization constraints that were impossible to capture before. E.g. small-signal stability constraints can now be accurately captured in an OPF

Wrap-up

- From decision trees and neural networks to Mixed Integer Linear Programs (MILP)
  - **Exact** transformation
  - **Capture efficiently constraints that were previously intractable** (e.g. based on differential equations)

- Challenges
  - Handling of non-linear equality constraints: SOCP or linearization appear effective
  - Accurately capturing the feasible space: **introducing ϵ-conservativeness**

- The framework can apply to any general non-linear program with intractable constraints

Appendix - End