Nonlinear Optimization with GPU-Accelerated Neural Network Constraints

Robert Parker
Los Alamos National Laboratory
Los Alamos, NM 87545, USA

Oscar Dowson
Dowson Farms
New Zealand

Nicole LoGiudice
Texas A&M University
College Station, TX 77843, USA

Manuel Garcia
Los Alamos National Laboratory
Los Alamos, NM 87545, USA

Russell Bent
Los Alamos National Laboratory
Los Alamos, NM 87545, USA

Abstract

We propose a reduced-space formulation for optimizing over trained neural networks where the network's outputs and derivatives are evaluated on a GPU. To do this, we treat the neural network as a "gray box" where intermediate variables and constraints are not exposed to the optimization solver. Compared to the full-space formulation, in which intermediate variables and constraints *are* exposed to the optimization solver, the reduced-space formulation leads to faster solves and fewer iterations in an interior point method. We demonstrate the benefits of this method on two optimization problems: Adversarial generation for a classifier trained on MNIST images and security-constrained optimal power flow with transient feasibility enforced using a neural network surrogate.

1 Introduction

Optimization over trained machine learning (ML) models can be used to verify ML models [1, 2], generate adversarial examples, and use these ML models for optimization-based design and control [3, 4]. For smooth, nonlinear ML models (e.g., neural networks with non-ReLU activation functions) or design or control tasks involving nonconvex constraints, the resulting optimization problem must be solved with a nonconvex optimization solver (as opposed to a mixed-integer, linear solver). Global optimization of nonconvex functions is well-known to be NP-hard, but local optimization of these functions can be done efficiently with interior point methods [5].

Despite favorable scalability of interior point methods in other domains (e.g., power grid operation [6]), we are unaware of recent work investigating the scalability of interior point methods with large neural networks (NNs) embedded. Recent work using interior point methods to optimize over trained NNs [7, 3, 8] has been limited to small NN models (fewer than 1M trained parameters). In this work, we examine the scalability of interior point methods with large NN models (over 100M trained parameters) embedded. We propose a reduced-space formulation with GPU acceleration as a scalable method of solving these optimization problems.

GPU-accelerated optimization has received much recent attention. First-order methods for linear programming [9] have led to implementations that offload matrix-vector multiplications to a GPU [10]. Simultaneously, the development of GPU-accelerated sparse matrix factorization algorithms, e.g., cuDSS [11], have enabled GPU-acceleration for a wider class of optimization problem [12, 13]. In contrast to these general-purpose methods, our method is tailored to nonlinear optimization problems with neural networks embedded.

39th Conference on Neural Information Processing Systems (NeurIPS 2025) Workshop: ScaleOpt: GPU-Accelerated and Scalable Optimization.

2 Nonlinear optimization

We study nonlinear optimization problems in the form given by Problem (1):

$$\min_x f(x) \text{ subject to } \begin{cases} g(x) = 0 \\ x \geq 0 \end{cases} \tag{1}$$

This formulation does not preclude general inequality constrains, which may be reformulated with a slack variable. We consider interior point methods, such as IPOPT [14], for solving (1), which require that functions f and g are twice continuously differentiable [5]. We note that this requirement precludes neural networks with non-differentiable activation functions such as ReLU. Interior point methods iteratively compute search directions d by solving the linear system (2):

$$\begin{bmatrix} (\nabla^2 \mathcal{L}(x) + \alpha) & \nabla g(x)^T \\ \nabla g(x) & 0 \end{bmatrix} d = - \begin{bmatrix} \nabla f(x) + \nabla g(x)^T \lambda + \beta \\ g(x) \end{bmatrix}, \tag{2}$$

where α and β are additional terms that are not shown for simplicity. The matrix on the left-hand side is referred to as the *Karush-Kuhn Tucker*, or *KKT*, matrix. To construct this system, solvers rely on callbacks (oracles) that provide the Jacobian, ∇g, and the Hessian of the Lagrangian function, $\nabla^2 \mathcal{L}$. These are typically provided by an automatic differentiation system [15].

3 Representing neural networks as constraints

We consider neural network predictors, $y = \text{NN}(x)$, defined by repeated application of an affine transformation and a nonlinear activation function σ over L layers:

$$y_l = \sigma_l(W_l y_{l-1} + b_l) \quad l \in \{1, \ldots, L\}, \tag{3}$$

where $y_0 = x$ and $y = y_L$. In this context, training weights W_l and b_l are considered constant. Pre-trained neural network predictors may be embedded into the constraints of an optimization problem using either *full-space* or *reduced-space* formulations [16, 17].

3.1 Full-space

In the *full-space* formulation, we add additional variables and constraints for each layer of the neural network:

$$\begin{aligned} z_l &= W_l y_{l-1} + b_l \quad && l \in \{1, \ldots, L\} \\ y_l &= \sigma_l(z_l) && l \in \{1, \ldots, L\}. \end{aligned} \tag{4}$$

The full-space approach prioritizes small, sparse nonlinear constraints at the cost of introducing (potentially) many new variables and constraints.

3.2 Reduced-space

In the *reduced-space* formulation, we add a single vector-valued decision variable y to represent the output of the final activation function and a single vector-valued nonlinear equality constraint that encodes the complete neural network:

$$y = \text{NN}(x) = \sigma_L(W_L(\ldots \sigma_l(W_l(\ldots \sigma_1(W_1 x + b_1)\ldots) + b_l)\ldots) + b_L) \tag{5}$$

The benefit of this formulation is that we only add a single decision variable and a single nonlinear equality constraint (each of dimension of the neural network's output). The drawback is that this nonlinear equality constraint contains a large, complicated algebraic expression that can be expensive to evaluate and differentiate. However, this can be alleviated by using dedicated neural network modeling software (e.g., PyTorch [18]) to represent the neural network constraint rather than general-purpose algebraic modeling software (e.g., JuMP [19]). Doing so also allows us to exploit GPU acceleration via PyTorch's CUDA interface. We note that this approach limits the information that can be communicated between the neural network and the optimization solver. Oracles (which PyTorch provides for function, Jacobian, and Hessian evaluation) can be queried, but the internal structure of the neural network cannot be exploited directly. This is suitable for nonlinear *local* optimization, where only oracles are required, but not for *global* optimization, where the algebraic form of constraints is exploited to construct relaxations.

Table 1: Structure of neural network models

Model	N. inputs	N. outputs	N. neurons	N. param.	Activation
MNIST	784	10	1k	167k	Tanh+SoftMax
MNIST	784	10	3k	1M	Tanh+SoftMax
MNIST	784	10	5k	5M	Tanh+SoftMax
MNIST	784	10	11k	18M	Tanh+SoftMax
MNIST	784	10	21k	70M	Tanh+SoftMax
MNIST	784	10	41k	274M	Sigmoid+SoftMax
SCOPF	117	37	254	15k	Tanh
SCOPF	117	37	1k	578k	Tanh
SCOPF	117	37	5k	4M	Tanh
SCOPF	117	37	36k	68M	Tanh
SCOPF	117	37	152k	592M	Tanh

3.2.1 The Hessian of the Lagrangian

Interior point methods require oracles to evaluate the Hessian of the Lagrangian, $\nabla^2\mathcal{L}$:

$$\nabla^2\mathcal{L}(x,\lambda) = \nabla^2 f(x) + \sum_i \lambda_i \nabla^2 g_i(x), \tag{6}$$

where λ is the vector of Lagrange multipliers of the equality constraints in Equation 1. Naively constructing $\sum_i \lambda_i \nabla^2 g_i$ for constraints involving a reduced-space neural network, $g(x,y) = y - \mathrm{NN}(x)$, would require (1) evaluating the Hessian of the neural network, $\nabla^2\mathrm{NN}(x)$, and (2) computing a sum-product with λ along the first rank of this third-order Hessian tensor. This sum-product is potentially expensive for a dense Hessian ($\mathcal{O}(mn^2)$, where m is the output dimension and n is the input dimension of the neural network). For this reason, we encode the Lagrangian of the neural network, $\lambda^T\mathrm{NN}(x)$, directly as a linear layer in PyTorch and differentiate through this scalar-valued function to directly compute the $n \times n$ Hessian matrix, $\sum_i \lambda_i \nabla^2 \mathrm{NN}_i(x)$.

4 Test problems

We test the full and reduced-space formulations on two nonlinear optimization problems with NNs embedded: (1) Adversarial image generation using a neural network MNIST classifier (denoted "MNIST") and (2) security-constrained optimal power flow with a neural network constraint enforcing transient feasibility (denoted "SCOPF").

4.1 Adversarial image generation for an MNIST classifier

We train a set of neural networks using smooth activation functions (hyperbolic tangent, sigmoid, and softmax) to serve as classifiers for images from the MNIST set of handwritten digits [20]. Inputs are the 28×28 grayscale pixel colors, flattened into a 784-dimensional vector, and outputs are scores for each digit, 0-9, that may be interpreted as the probability that the image represents the corresponding digit. The neural networks each have seven layers total and have between 128 and 8192 neurons per hidden layer. The networks are trained to have accuracies of at least 95% on the test set of 10,000 images from the dataset. The number of neurons, trained parameters, and activation functions for each network are shown in Table 1. Our optimization problem finds a minimal perturbation to a reference image that results in a misclassification:

$$\begin{aligned} \min_{x} \quad & \|x - x_{\mathrm{ref}}\|_1 \\ \text{subject to} \quad & y = \mathrm{NN}(x) \\ & y_t \geq 0.6 \end{aligned} \tag{7}$$

Here, x contains the grayscale values of the generated image, x_{ref} contains those of the reference image, and t is the coordinate of the neural network's output, y, corresponding to a target label (misclassification). The neural network constraint, $y = \mathrm{NN}(x)$, may be written in full-space or reduced-space formulations. The image must be misclassified with at least 60% confidence.

4.2 Transient-constrained optimal power flow

Security-constrained optimal power flow (SCOPF) is a well-established problem for dispatching generators in an electric power network in which feasibility of the network (i.e., the ability to meet demand) is enforced for a set of *contingencies* [6]. Each contingency k represents the loss of a set of generators and/or power lines. We consider a variant of this problem where, in addition to enforcing steady-state feasibility, we enforce feasibility of the transient response resulting from the contingency. In particular, we enforce that the transient frequency at each bus is at least $\eta = 59.4$ Hz for the 30 second interval following each contingency. This problem is given by Equation 8:

$$\min_{S^g, V} c(\mathbb{R}(S^g)) \text{ subject to } \begin{cases} F_k(S^g, V, \mathbf{S^d}) \leq 0 & k \in \{0, \ldots, K\} \\ G_k(S^g, \mathbf{S^d}) \geq \eta \mathbb{1} & k \in \{1, \ldots, K\}. \end{cases} \tag{8}$$

Here, S^g is a vector of complex AC power generations for each generator in the network, V is a vector of complex bus voltages, c is a quadratic cost function, and $\mathbf{S^d}$ is a constant vector of complex power demands. Here $F_k \leq 0$ represents the set of constraints enforcing feasibility of the power network for contingency k (see [21]), where $k = 0$ refers to the base network, and G_k maps generations and demands to the minimum frequency at each bus over the interval considered.

In this work, we consider an instance of Problem 8 defined on a 37-bus synthetic test grid [22, 23]. In this case, G_k has 117 inputs and 37 outputs. We consider a single contingency that outages generator 5 on bus 23. We choose a small grid model with a single contingency because our goal is to test the different neural network formulations, not the SCOPF formulation itself.

Stability surrogate model Instead of considering the differential equations describing transient behavior of the power network directly in the optimization problem, we approximate G_k with a neural network surrogate, as proposed by Garcia et al. [24]. Our surrogate is trained on data from 110 high-fidelity simulations using PowerWorld [25] with generations and loads uniformly sampled from within a $\pm 20\%$ interval of each nominal value. We use sequential neural networks with $\tanh$ activation functions with between two and 20 layers and between 50 and 4,000 neurons per layer. As shown in Table 1, these networks have between 7,000 and 592 million trained parameters. The networks are trained to minimize mean squared error using the Adam optimizer [26] until training loss (mean squared prediction error) is below 0.01 for 1,000 consecutive epochs. We use a simple training procedure and small amount of data because our goal is to test optimization formulations with embedded neural networks, rather than the neural networks themselves.

5 Results

5.1 Computational setting

We model the SCOPF problem using PowerModels [27], PowerModelsSecurityConstrained [28], and JuMP [19]. Neural networks are modeled using PyTorch [18] and embedded into the optimization problem using MathOptAI.jl [29]. Optimization problems are solved using the IPOPT interior point method [14] with MA57 [30] as the linear solver to a tolerance of 10^{-6}. We note that IPOPT runs exclusively on CPU. Interfacing our function evaluation methods with a GPU-enabled nonlinear optimization solver, e.g., MadNLP [12], would be valuable future work. We evaluate and differentiate the full-space model's functions on a CPU using JuMP. While we could use a framework that supports GPU-accelerated function evaluations such as PyTorch or ExaModels (see [12]), our results indicate that these function evaluations are not a significant contributor to solve time (see Table 4). Because our reduced-space models use PyTorch, they can be evaluated on a CPU or GPU. We run our experiments on the Darwin cluster at Los Alamos National Laboratory. The CPU used in these experiments is an AMD EPYC with 128 cores and 500 GB of RAM and the GPU is an NVIDIA A100 with 40 GB of on-device memory. The code used to produce these results can be found at https://github.com/Robbybp/moai-examples.

5.2 Structural results

Table 2 shows the numbers of variables, constraints, and nonzero entries of the derivative matrices for the two optimization problems with different neural networks and formulations. With the reduced-space formulations, the structure of the optimization problem does not change as the neural network

Table 2: Structure of optimization problems with embedded neural networks

Model	Formulation	NN param.	N. var.	N. con.	Jac. NNZ	Hess. NNZ
MNIST	Full-space	167k	3k	2k	171k	661
MNIST	Full-space	1M	7k	5k	1M	2k
MNIST	Full-space	5M	12k	11k	5M	5k
MNIST	Full-space	18M	22k	21k	18M	10k
MNIST	Full-space	70M	43k	41k	70M	20k
MNIST	Full-space	274M	84k	82k	275M	40k
MNIST	Reduced-space	All NNs	2k	805	10k	307k
SCOPF	Full-space	15k	1k	1k	17k	1k
SCOPF	Full-space	578k	4k	4k	567k	2k
SCOPF	Full-space	4M	11k	11k	4M	6k
SCOPF	Full-space	68M	73k	73k	68M	37k
SCOPF	Full-space	592M	305k	305k	592M	153k
SCOPF	Reduced-space	All NNs	1k	1k	10k	8k

surrogate adds more interior layers. By contrast, the full-space formulation grows in numbers of variables, constraints, and nonzeros as the neural network gets larger. In this case, the number of nonzeros in the Jacobian matrix is approximately the number of trained parameters in the embedded neural network model. These problem structures suggest that the full-space formulation will lead to expensive KKT matrix factorizations, while this will not be an issue for the reduced-space formulation.

Table 3: Solve times with different neural networks and formulations

Model	Formulation	Platform	NN param.	Solve time (s)	N. iter.	Time/iter. (s)	Objective
MNIST	Full-space	CPU	167k	19	290	0.07	3.3
MNIST	Full-space	CPU	1M	348	1157	0.3	3.3
MNIST	Full-space	CPU	5M	11536	2110	5	6.1
MNIST	Full-space	CPU	18M*	–	–	–	–
MNIST	Reduced-space	CPU	167k	3	28	0.1	3.3
MNIST	Reduced-space	CPU	1M	4	43	0.1	3.3
MNIST	Reduced-space	CPU	5M	10	77	0.1	5.9
MNIST	Reduced-space	CPU	18M	63	147	0.4	4.7
MNIST	Reduced-space	CPU	70M	58	80	0.7	2.0
MNIST	Reduced-space	CPU	274M	45	31	1	5.3
MNIST	Reduced-space	CPU+GPU	167k	2	28	0.07	3.3
MNIST	Reduced-space	CPU+GPU	1M	4	43	0.08	3.3
MNIST	Reduced-space	CPU+GPU	5M	5	67	0.08	5.9
MNIST	Reduced-space	CPU+GPU	18M	12	149	0.08	4.8
MNIST	Reduced-space	CPU+GPU	70M	8	82	0.1	2.0
MNIST	Reduced-space	CPU+GPU	274M	3	28	0.1	5.3
SCOPF	Full-space	CPU	15k	4	675	0.01	82379
SCOPF	Full-space	CPU	578k	399	1243	0.3	82379
SCOPF	Full-space	CPU	4M	8858	2505	4	82379
SCOPF	Full-space	CPU	68M*	–	–	–	–
SCOPF	Reduced-space	CPU	15k	7	358	0.02	82379
SCOPF	Reduced-space	CPU	578k	5	147	0.03	82379
SCOPF	Reduced-space	CPU	4M	3	53	0.06	82379
SCOPF	Reduced-space	CPU	68M	37	40	1	82379
SCOPF	Reduced-space	CPU	592M	144	42	3	82379
SCOPF	Reduced-space	CPU+GPU	15k	5	298	0.02	82379
SCOPF	Reduced-space	CPU+GPU	578k	3	162	0.02	82379
SCOPF	Reduced-space	CPU+GPU	4M	1	53	0.03	82379
SCOPF	Reduced-space	CPU+GPU	68M	2	40	0.04	82379
SCOPF	Reduced-space	CPU+GPU	592M	3	42	0.08	82379

* Fails to solve within 5 hour time limit

5.3 Runtime results

Runtimes for the different formulations with neural network surrogates of increasing size are given in Table 3. For the reduced-space formulation, we also compare CPU-only solves with solves that leverage a GPU for neural network evaluation (and differentiation). The results immediately show that the full-space formulation is not scalable to neural networks with more than a few million trained parameters. The full-space formulation exceeds the 5-hour time limit on networks above this size. A breakdown of solve times, given in Table 4, confirms the bottleneck in this formulation. The full-space formulation spends almost all of its solve time in the IPOPT algorithm, which we assume is dominated by KKT matrix factorization. [1]

By contrast, the reduced-space formulation is capable of solving the optimization problem with the largest neural network surrogates tested. While a CPU-only solve takes a relatively long 144 s for the SCOPF problem with a 592M-parameter neural network embedded, a GPU-accelerated solve of the same problem solves in only three seconds. In all cases, the solve time with the reduced-space formulation is dominated by Hessian evaluation, which explains the large speed-ups obtained with the GPU (9$\times$ for the MNIST problem with a 274M-parameter neural network and 48$\times$ for the SCOPF problem with a 592M-parameter neural network).

Finally, we observe that the interior point method requires many more iterations with the full-space formulation than with the reduced-space formulation. While the theory of interior point methods' convergence behavior with full and reduced-space formulations is not well-understood, this behavior is consistent with lower iteration counts and improved convergence reliability that have been observed for reduced-space formulations in other contexts [31, 32, 33]. In addition to iteration counts, the times-per-iteration are significantly higher for the full-space formulation, suggesting that its bottlenecks would not be remedied by using a different optimization algorithm that is able to converge in fewer iterations.

Table 4: Solve time breakdowns for selected neural networks and formulations

Model	Formulation	NN. param.	Platform	Solve time (s)	Percent of solve time (%)			
					Function	Jacobian	Hessian	Solver
MNIST	Full-space	5M	CPU	11536	0.05	0.05	0.07	99+
MNIST	Reduced-space	274M	CPU	45	5	13	81	1
MNIST	Reduced-space	274M	CPU+GPU	3	3	5	72	19
SCOPF	Full-space	4M	CPU	8858	0.1	0.1	0.2	99+
SCOPF	Reduced-space	592M	CPU	144	6	14	80	0.2
SCOPF	Reduced-space	592M	CPU+GPU	3	6	19	68	8

6 Conclusion

This work demonstrates that nonlinear local optimization problems may incorporate neural networks with hundreds of millions of trained parameters, with minimal overhead, using a reduced-space formulation that exploits efficient automatic differentiation and GPU acceleration. Further research should test this formulation on neural networks with larger input and output dimensions to measure the point at which CPU-GPU data transfer becomes a bottleneck; our experiments indicate that, for our test problems, this overhead is small compared to the effort of evaluating and differentiating the neural network itself. A disadvantage of our formulation is that it is not suitable for global optimization as the non-convex neural network constraints are not represented in a format that can be communicated to any global optimization solver we are aware of. Interfacing convex under and over-estimators of neural networks (e.g., CROWN [34]) with global optimization solvers is another interesting area for future work. Additionally, relative performance of the full and reduced-space formulations may change in different applications. This motivates future research and development to improve the performance of the full-space formulation, which may be achieved by linear algebra decompositions that exploit the structure of the neural network's Jacobian in the KKT matrix.

[1] This is difficult to confirm directly, but by parsing IPOPT's logs, we see that, for the SCOPF model with the 4M-parameter neural network, IPOPT reports spending 96% of its solve time in a category called "LinearSystemFactorization".

Acknowledgments and Disclosure of Funding

This work was funded by the LDRD program at Los Alamos National Laboratory under the Artimis project, the Information Science and Technology Institute, and the Center for Nonlinear Studies. LA-UR-25-29374.

References

[1] Vincent Tjeng, Kai Y. Xiao, and Russ Tedrake. "Evaluating Robustness of Neural Networks with Mixed Integer Programming". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=HyGIdiRqtm.

[2] Rudy Bunel, Ilker Turkaslan, Philip H.S. Torr, Pushmeet Kohli, and M. Pawan Kumar. "A unified view of piecewise linear neural network verification". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. NIPS'18. Montréal, Canada: Curran Associates Inc., 2018, 4795–4804.

[3] Sergio I. Bugosen, Carl D. Laird, and Robert B. Parker. "Process flowsheet optimization with surrogate and implicit formulations of a Gibbs reactor". In: *Systems and Control Transactions* 3 (2024), pp. 113–120. DOI: https://doi.org/10.69997/sct.148498.

[4] Francisco Javier López-Flores, César Ramírez-Márquez, and José María Ponce-Ortega. "Process Systems Engineering Tools for Optimization of Trained Machine Learning Models: Comparative and Perspective". In: *Industrial & Engineering Chemistry Research* 63.32 (2024), pp. 13966–13979. DOI: 10.1021/acs.iecr.4c00632.

[5] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, 2006.

[6] Ignacio Aravena, Daniel K. Molzahn, Shixuan Zhang, Cosmin G. Petra, Frank E. Curtis, Shenyinying Tu, Andreas Wächter, Ermin Wei, Elizabeth Wong, Amin Gholami, Kaizhao Sun, Xu Andy Sun, Stephen T. Elbert, Jesse T. Holzer, and Arun Veeramany. "Recent Developments in Security-Constrained AC Optimal Power Flow: Overview of Challenge 1 in the ARPA-E Grid Optimization Competition". In: *Operations Research* 71.6 (2023), pp. 1997–2014. DOI: 10.1287/opre.2022.0315.

[7] Zachary Kilwein, Jordan Jalving, Michael Eydenberg, Logan Blakely, Kyle Skolfield, Carl Laird, and Fani Boukouvala. "Optimization with Neural Network Feasibility Surrogates: Formulations and Application to Security-Constrained Optimal Power Flow". In: *Energies* 16.16 (2023). ISSN: 1996-1073. DOI: 10.3390/en16165913. URL: https://www.mdpi.com/1996-1073/16/16/5913.

[8] Carlos Andrés Elorza Casas, Luis A. Ricardez-Sandoval, and Joshua L. Pulsipher. "A comparison of strategies to embed physics-informed neural networks in nonlinear model predictive control formulations solved via direct transcription". In: *Computers & Chemical Engineering* 198 (2025), p. 109105. ISSN: 0098-1354. DOI: https://doi.org/10.1016/j.compchemeng.2025.109105.

[9] David Applegate, Mateo Diaz, Oliver Hinder, Haihao Lu, Miles Lubin, Brendan O' Donoghue, and Warren Schudy. "Practical Large-Scale Linear Programming using Primal-Dual Hybrid Gradient". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan. Vol. 34. Curran Associates, Inc., 2021, pp. 20243–20257. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/a8fbbd3b11424ce032ba813493d95ad7-Paper.pdf.

[10] Haihao Lu and Jinwen Yang. *cuPDLP.jl: A GPU Implementation of Restarted Primal-Dual Hybrid Gradient for Linear Programming in Julia*. 2024. arXiv: 2311.12180 [math.OC]. URL: https://arxiv.org/abs/2311.12180.

[11] *NVIDIA cuDSS (preview): A high-performance CUDA Library for Direct Sparse Solvers*. NVIDIA. 2025. URL: https://docs.nvidia.com/cuda/cudss/.

[12] Sungho Shin, Mihai Anitescu, and François Pacaud. "Accelerating optimal power flow with GPUs: SIMD abstraction of nonlinear programs and condensed-space interior-point methods". In: *Electric Power Systems Research* 236 (2024), p. 110651. ISSN: 0378-7796. DOI: https://doi.org/10.1016/j.epsr.2024.110651.

[13] François Pacaud and Sungho Shin. "GPU-accelerated dynamic nonlinear optimization with ExaModels and MadNLP". In: *2024 IEEE 63rd Conference on Decision and Control (CDC)*. 2024, pp. 5963–5968. DOI: 10.1109/CDC56724.2024.10886720.

[14] Andreas Wächter and Lorenz T Biegler. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". In: *Mathematical programming* 106.1 (2006), pp. 25–57.

[15] Andreas Griewank and Andrea Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. Second. USA: Society for Industrial and Applied Mathematics, 2008. ISBN: 0898716594.

[16] Francesco Ceccon, Jordan Jalving, Joshua Haddad, Alexander Thebelt, Calvin Tsay, Carl D Laird, and Ruth Misener. "OMLT: Optimization & Machine Learning Toolkit". In: *Journal of Machine Learning Research* 23.349 (2022), pp. 1–8. URL: http://jmlr.org/papers/v23/22-0277.html.

[17] Artur M Schweidtmann and Alexander Mitsos. "Deterministic global optimization with artificial neural networks embedded". In: *Journal of Optimization Theory and Applications* 180.3 (2019), pp. 925–948.

[18] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019).

[19] Miles Lubin, Oscar Dowson, Joaquim Dias Garcia, Joey Huchette, Benoît Legat, and Juan Pablo Vielma. "JuMP 1.0: Recent improvements to a modeling language for mathematical optimization". In: *Mathematical Programming Computation* 15.3 (2023), pp. 581–589. DOI: 10.1007/s12532-023-00239-3. URL: https://doi.org/10.1007/s12532-023-00239-3.

[20] Yann LeCun, Corinna Cortes, and Christopher J. C. Burges. "The MNIST Database of Handwritten Digits". In: (1998). URL: http://yann.lecun.com/exdb/mnist/.

[21] Mary B. Cain, Richard P. O'Neill, and Anya Castillo. *History of Optimal Power Flow and Formulations*. Tech. rep. Federal Energy Regulatory Commission, 2012.

[22] Adam B. Birchfield, Ti Xu, Kathleen M. Gegner, Komal S. Shetye, and Thomas J. Overbye. "Grid Structural Characteristics as Validation Criteria for Synthetic Networks". In: *IEEE Transactions on Power Systems* 32.4 (2017), pp. 3258–3265. DOI: 10.1109/TPWRS.2016.2616385.

[23] Adam Birchfield. *Hawaii Synthetic Grid – 37 Buses*. Accessed 2024-12-10. 2023. URL: https://electricgrids.engr.tamu.edu/hawaii40/.

[24] Manuel Garcia, Nicole LoGiudice, Robert Parker, and Russell Bent. *Transient Stability-Constrained OPF: Neural Network Surrogate Models and Pricing Stability*. 2025. arXiv: 2502.01844 [math.OC]. URL: https://arxiv.org/abs/2502.01844.

[25] *PowerWorld Simulator Manual*. https://www.powerworld.com/WebHelp/ Accessed December 10, 2024. PowerWorld Corporation. Champaign, IL, USA.

[26] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG]. URL: https://arxiv.org/abs/1412.6980.

[27] Carleton Coffrin, Russell Bent, Kaarthik Sundar, Yeesian Ng, and Miles Lubin. "PowerModels.jl: An Open-Source Framework for Exploring Power Flow Formulations". In: *2018 Power Systems Computation Conference (PSCC)*. 2018, pp. 1–8. DOI: 10.23919/PSCC.2018.8442948.

[28] Carleton Coffrin. *PowerModelsSecurityConstrained.jl*. Accessed 2024-12-10. 2022. URL: https://github.com/lanl-ansi/PowerModelsSecurityConstrained.jl.

[29] Oscar Dowson, Robert B Parker, and Russel Bent. *MathOptAI.jl: Embed trained machine learning predictors into JuMP models*. 2025. arXiv: 2507.03159 [cs.LG]. URL: https://arxiv.org/abs/2507.03159.

[30] Iain S. Duff. "MA57—a code for the solution of sparse symmetric definite and indefinite systems". In: 30.2 (2004). ISSN: 0098-3500. DOI: 10.1145/992200.992202.

[31] François Pacaud, Daniel Adrian Maldonado, Sungho Shin, Michel Schanen, and Mihai Anitescu. "A feasible reduced space method for real-time optimal power flow". In: *Electric Power Systems Research* 212 (2022), p. 108268. ISSN: 0378-7796. DOI: https://doi.org/10.1016/j.epsr.2022.108268.

[32] Robert Parker, Bethany Nicholson, John Siirola, Carl Laird, and Lorenz Biegler. "An implicit function formulation for optimization of discretized index-1 differential algebraic systems". In: *Computers & Chemical Engineering* 168 (2022), p. 108042. ISSN: 0098-1354. DOI: https://doi.org/10.1016/j.compchemeng.2022.108042.

[33] Sakshi Naik, Lorenz Biegler, Russell Bent, and Robert Parker. *Variable aggregation for nonlinear optimization problems*. 2025. arXiv: 2502.13869 [math.OC]. URL: https://arxiv.org/abs/2502.13869.

[34] Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. "Efficient Neural Network Robustness Certification with General Activation Functions". In: *Advances in Neural Information Processing Systems (NuerIPS)*. 2018.