OptNet and DC3

Topic: Differentiable Projections & Feasibility Layers

Ya-Chi Chu, Jan. 20, 2026

Quiz Questions

Required reading: OptNet

An OptNet layer takes the form

$$
\begin{aligned}
z_{i+1} = \underset{z}{\text{argmin}} \quad & \frac{1}{2} z^T Q(z_i) z + q(z_i)^T z \\
\text{subject to} \quad & A(z_i) z = b(z_i) \\
& G(z_i) z \leq h(z_i)
\end{aligned}
$$

where z_i is the input to the layer and z_{i+1} is the output of the layer.

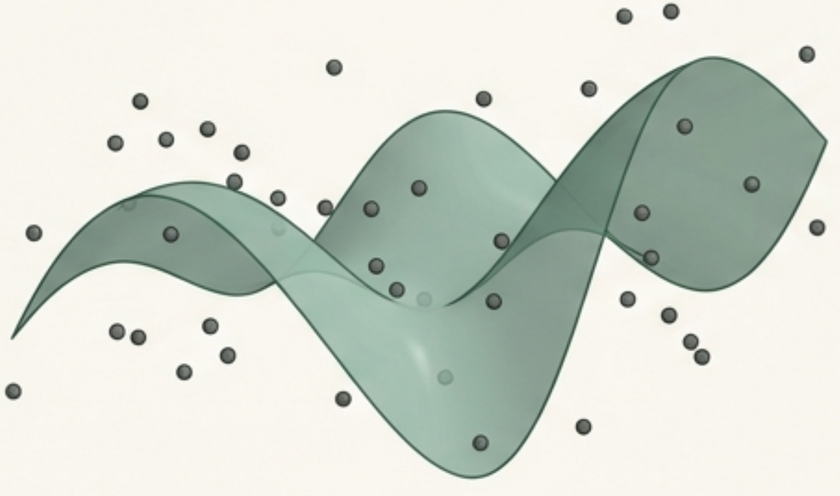
- What are the learnable parameters for the optimization layer in OptNet?
- What relation between optimal primal and dual solutions is used to derive the formula for backpropagation?

The Clash of Two Worlds: Approximation vs. Guarantees

Deep learning has revolutionized fields that rely on flexible function approximation. However, many critical real-world systems are governed by non-negotiable physical laws and logical rules. Applying standard neural networks in these domains can lead to solutions that are not just suboptimal, but physically impossible or nonsensical.

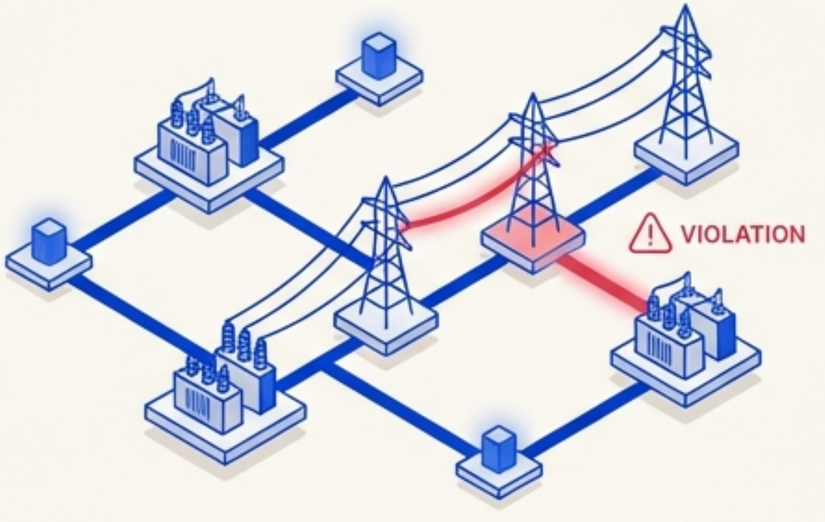
How can we bridge the gap between the fluid, unconstrained nature of deep learning and the rigid, rule-based world of hard constraints?

Approximation



Neural networks excel at learning complex, continuous functions.

Guarantees



Real-world systems demand strict feasibility. Violations can be catastrophic.

A Brief History of Constraints in Neural Networks

The idea of constraints in neural networks is not new, but the methods have evolved from simple, implicit encodings to powerful, explicit layers.

Implicit Constraints

Baked into the Architecture

Common layers that inherently enforce simple constraints on their outputs.

Softmax: Enforces that outputs sum to 1 (a simplex constraint).

ReLU: Enforces non-negativity...

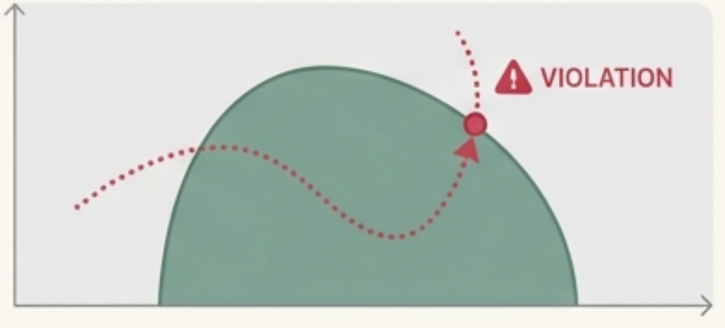
Sigmoid: Enforces upper and lower bounds [0, 1].

Soft Constraints

Penalties in the Loss Function

Violations are added as a penalty term to the loss function. This encourages, but does not guarantee, feasibility.

$$Loss = Original\_Loss + \lambda \cdot \|Constraint\_Violation\|^2$$

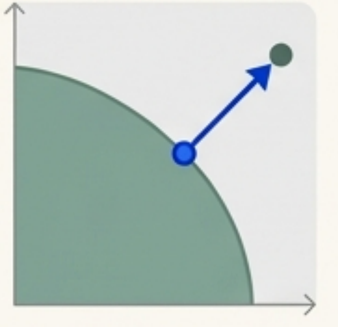


Explicit Differentiable Layers

The Modern Approach

A dedicated layer is added to the network that explicitly projects the output onto the feasible set, guaranteeing satisfaction. The key innovation is making making this projection layer differentiable, allowing for end-to-end training.

Key papers:
OptNet (2017),
DC3 (2021),
Πnet (2024),
HardNet (2025)

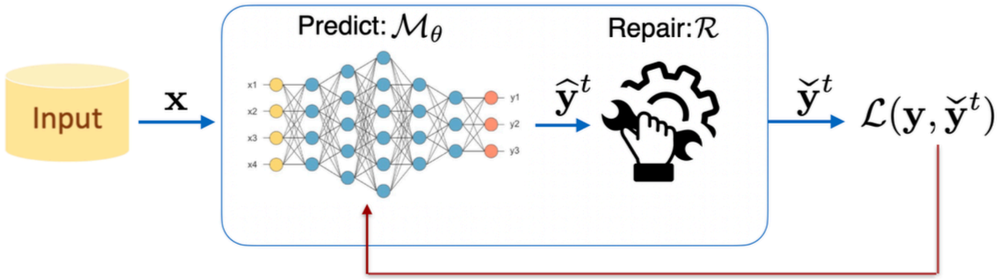


What are Differentiable Projections & Feasibility Layers?

- Traditional approach: Neural network → Post-processing with constraints
  - Network and constraints are separate
  - No feedback during training

Compute the projection onto feasible region $\mathcal{C}_x$: $\arg\min_{y \in \mathcal{C}_x} \|y - \mathcal{M}_\theta(x)\|^2$.

- Preferable approach: Neural network → Differentiable Projection Layer → Output
  - Projection layer encodes constraints as part of the architecture
  - Backpropagation flows through the projection layer
  - Network learns to produce outputs that work with constraints
- **Key Technical Challenge:** Computing gradients through the projection layer



Gradient Step: $\theta^{t+1} = \theta^t - \alpha \frac{\partial \mathcal{L}(\mathbf{y}, \check{\mathbf{y}}^t)}{\partial \theta}$

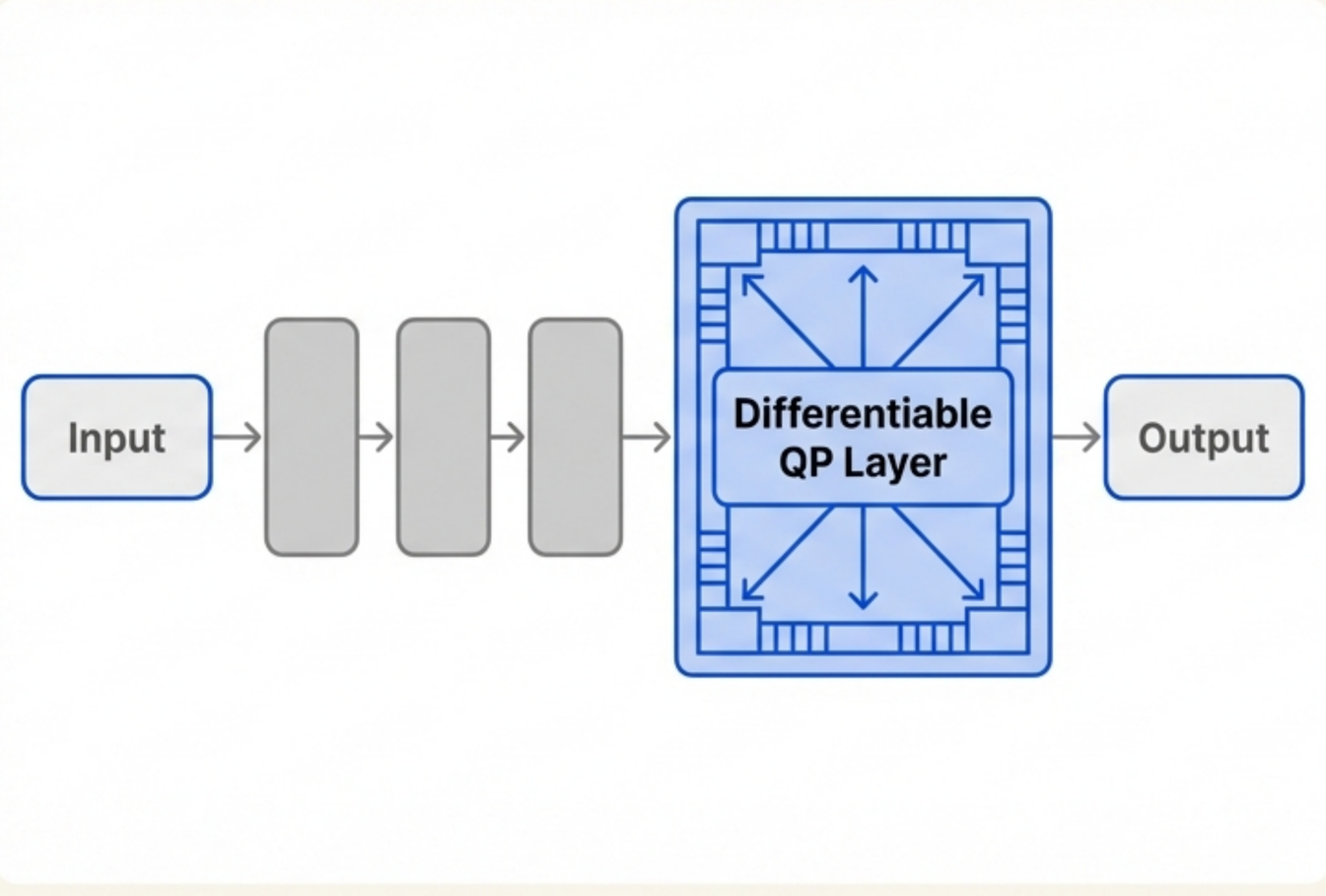
$$\frac{\partial \mathcal{L}(\mathbf{y}, \check{\mathbf{y}}^t)}{\partial \theta} = \frac{\partial \mathcal{L}(\mathbf{y}, \check{\mathbf{y}}^t)}{\partial \check{\mathbf{y}}^t} \boxed{\frac{\partial \mathcal{R}(\hat{\mathbf{y}}^t)}{\partial \hat{\mathbf{y}}^t}} \frac{\partial \mathcal{M}_{\theta^t}(\mathbf{x})}{\partial \theta}$$

Milestone 1: OptNet (2017) - The First Principled Bridge

Core Contribution: OptNet was a seminal work that introduced the concept of a **differentiable Quadratic Programming (QP) solver** as a layer within a neural network.

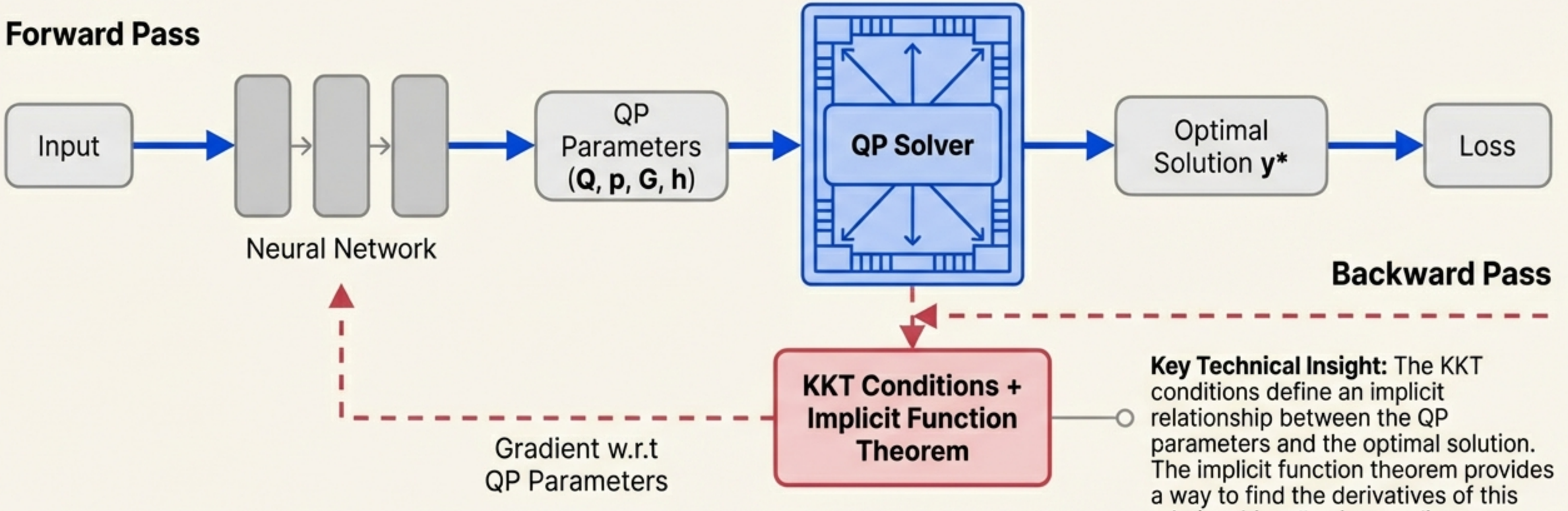
Why it was a breakthrough: For the first time, it demonstrated that a classic, constrained optimization problem could be treated as a differentiable building block in a deep learning model. This enabled true end-to-end training through hard, explicit linear constraints, paving the way for a new class of architectures.

"OptNet: Differentiable Optimization as a Layer in Neural Networks" — Amos & Kolter, ICML 2017.



How OptNet Works: Differentiation Through Optimization

OptNet's key insight was to differentiate through the solution of a QP without needing to differentiate the solver's iterative steps. The gradient is computed analytically based on the problem's optimality conditions.



Optimization Layer

$$
\begin{aligned}
z_{i+1} = \underset{z}{\operatorname{argmin}} \quad & \frac{1}{2} z^T Q(z_i) z + q(z_i)^T z \\
\text{subject to} \quad & A(z_i) z = b(z_i) \\
& G(z_i) z \leq h(z_i)
\end{aligned}
$$

- z is the optimization variable
- $Q(z_i), q(z_i), A(z_i), b(z_i), G(z_i), h(z_i)$ are parameters of the layer

Use KKT to compute Jacobian

$$L(z, \nu, \lambda) = \frac{1}{2} z^T Q z + q^T z + \nu^T (Az - b) + \lambda^T (Gz - h)$$

KKT Condition

$$\begin{aligned} Qz^\star + q + A^T\nu^\star + G^T\lambda^\star &= 0 \\ Az^\star - b &= 0 \\ D(\lambda^\star)(Gz^\star - h) &= 0, \end{aligned}$$

Take differentials

$$\begin{bmatrix} Q & G^T & A^T \\ D(\lambda^\star)G & D(Gz^\star - h) & 0 \\ A & 0 & 0 \end{bmatrix} \begin{bmatrix} \mathrm{d}z \\ \mathrm{d}\lambda \\ \mathrm{d}\nu \end{bmatrix} = - \begin{bmatrix} \mathrm{d}Qz^\star + \mathrm{d}q + \mathrm{d}G^T\lambda^\star + \mathrm{d}A^T\nu^\star \\ D(\lambda^\star)\mathrm{d}Gz^\star - D(\lambda^\star)\mathrm{d}h \\ \mathrm{d}Az^\star - \mathrm{d}b \end{bmatrix}.$$

Example: To compute Jacobian $\frac{\partial z^\star}{\partial b} \in \mathbb{R}^{n \times m}$

substitute $\mathrm{d}b = I$ and set all other differential terms on the right to zero

and solve for $\mathrm{d}z$

Numerical Results

Total Variation Denoising

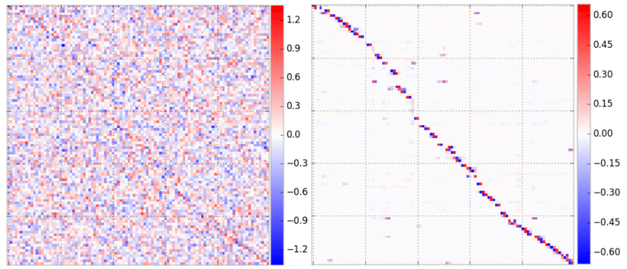
$$\underset{z}{\text{argmin}} \frac{1}{2}||y - z|| + \lambda||Dz||_1$$

y : input signal

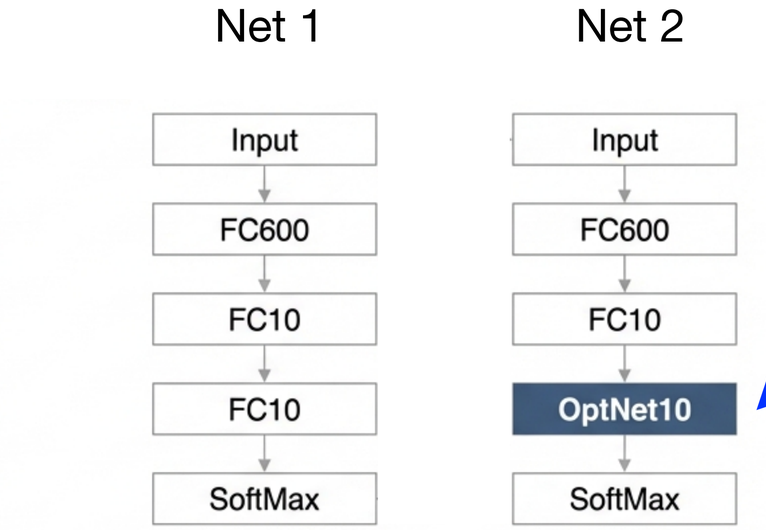
Method	Train MSE	Test MSE
FC Net	18.5	29.8
Pure OptNet	52.9	53.3
Total Variation	16.3	16.5
OptNet Tuned TV	13.8	**14.4**

TV: solve opt with $0 \leq \lambda \leq 100$. Best is $\lambda = 13$

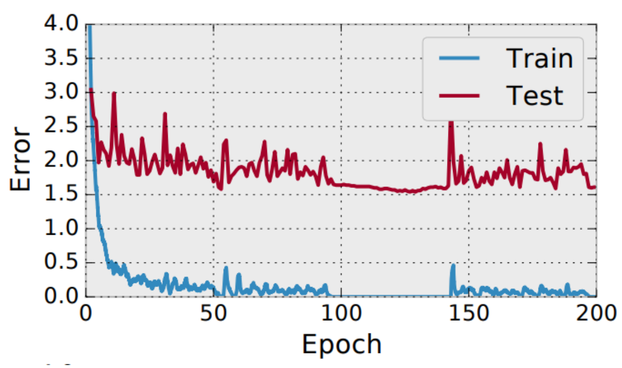
NN approaches: learn a mapping $f_\theta : y \mapsto z$.
(Supervised learning: MSE loss between true and predicted signal)

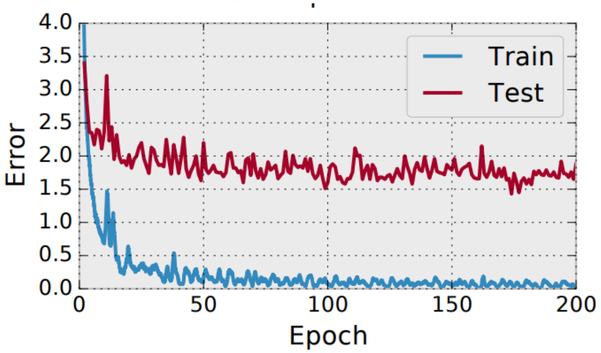


MNIST Classification



Opt. problem in this layer is 10-dim'l





4x4 Mini-Sudoku

			3
1			
		4	
4			1

2	4	1	3
1	3	2	4
3	1	4	2
4	2	3	1

Motivation: Sudoku can be approximated well with a linear program.

Their experiments do NOT use the exact mathematical formulation for Sudoku, nor encode the rules of Soudoku to model.

They only "semi-blindly" build a NN with Opt layer and train by supervised learning with 9000 instances.

Design of their Opt layer: a completely generic QP in "standard form" with only positivity inequality constraints but an arbitrary constraint matrix $Ax = b$, a small $Q = 0.1I$, and with the linear term q being the input one-hot encoding of the problem.

Findings:

- CNN is able to learn all of the necessary logic but over-fits
- OptNet learns most of the correct hard constraints and is able to generalize much better to unseen examples.

OptNet — Discussion Questions

- Based on your research domain, would you consider using OptNet's optimization layer in your work? Why or why not?
  - What potential benefits do you see for your applications?
  - What concerns or limitations might prevent you from adopting this approach?
- OptNet allows optimization layer in QP formulation. Can we generalize to integer programs? What are potential challenges?
- In which tasks might you use an optimization layer?

AC Optimal Power Flow

Example Application

- Problem for electrical grid operation: Determine optimal power generation by each generator to meet demand
- Increasing renewable energy integration requires more frequent solving
- **Network Representation:**
  - Power network modeled as graph with b nodes (buses)
  - Edges weighted by complex admittances $w_{ij} \in \mathbb{C}$

AC Optimal Power Flow

Non-Convex Optimization Problem Formulation

Input (coefficients):

p_d : real power demand

q_d : reactive power demand

Output (variables):

p_g : real power generation

q_g : reactive power generation

v : real and imaginary voltage

$$\underset{p_g \in \mathbb{R}^b,\ q_g \in \mathbb{R}^b,\ v \in \mathbb{C}^b}{\text{minimize}} \quad p_g^T A p_g + b^T p_g \qquad \text{cost}$$

$$\text{subject to} \quad p_g^{\min} \leq p_g \leq p_g^{\max}, \quad q_g^{\min} \leq q_g \leq q_g^{\max}, \quad v^{\min} \leq |v| \leq v^{\max}, \qquad \text{hardware limits}$$

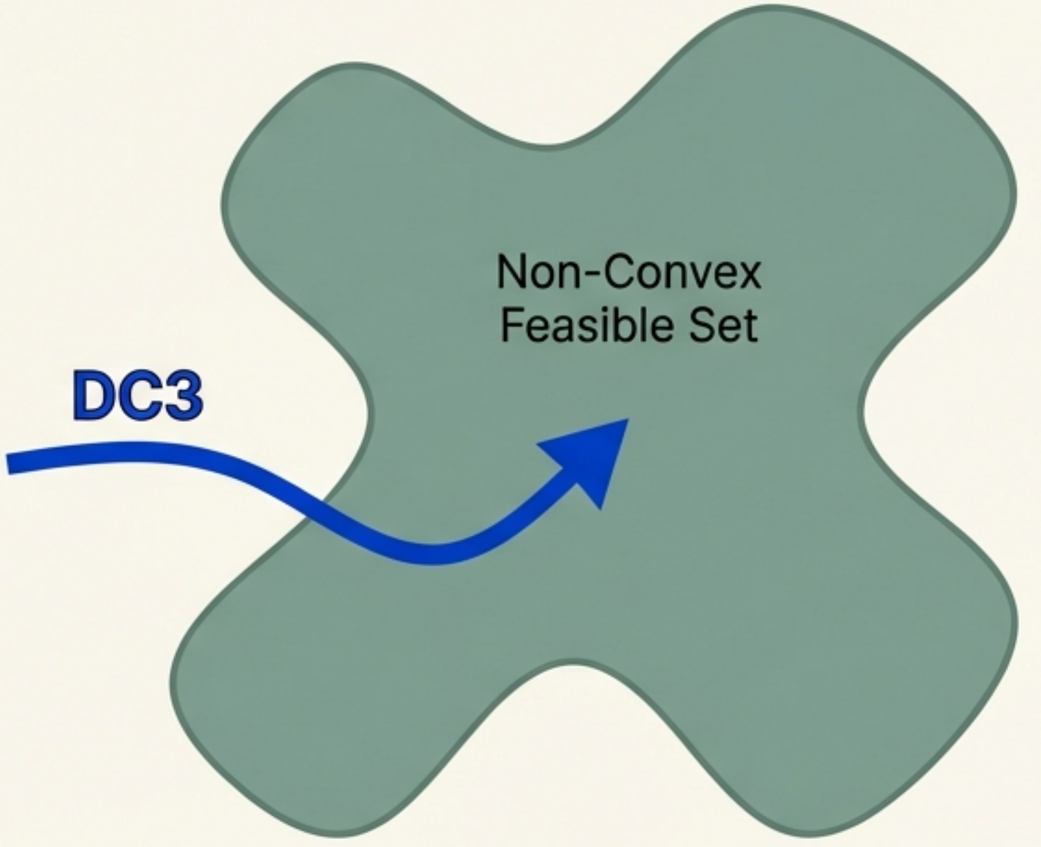
$$(p_g - p_d) + (q_g - q_d)i = \operatorname{diag}(v)\overline{W}\overline{v}. \qquad \text{Kirchhoff's laws (Power flow eq.)}$$

Milestone 2: DC3 (2021) - Generalizing to Non-Convex Constraints

Core Contribution: Deep Constraint Completion and Correction (DC3) provides a **general and efficient framework** for incorporating hard, potentially non-convex, equality and inequality constraints into deep learning models.

Why it's a step forward: Instead of embedding a full, expensive solver, DC3 uses a novel two-step differentiable procedure to enforce feasibility. This makes it faster and applicable to a much broader class of problems, including those with complex non-convex constraints found in real-world applications like power systems.

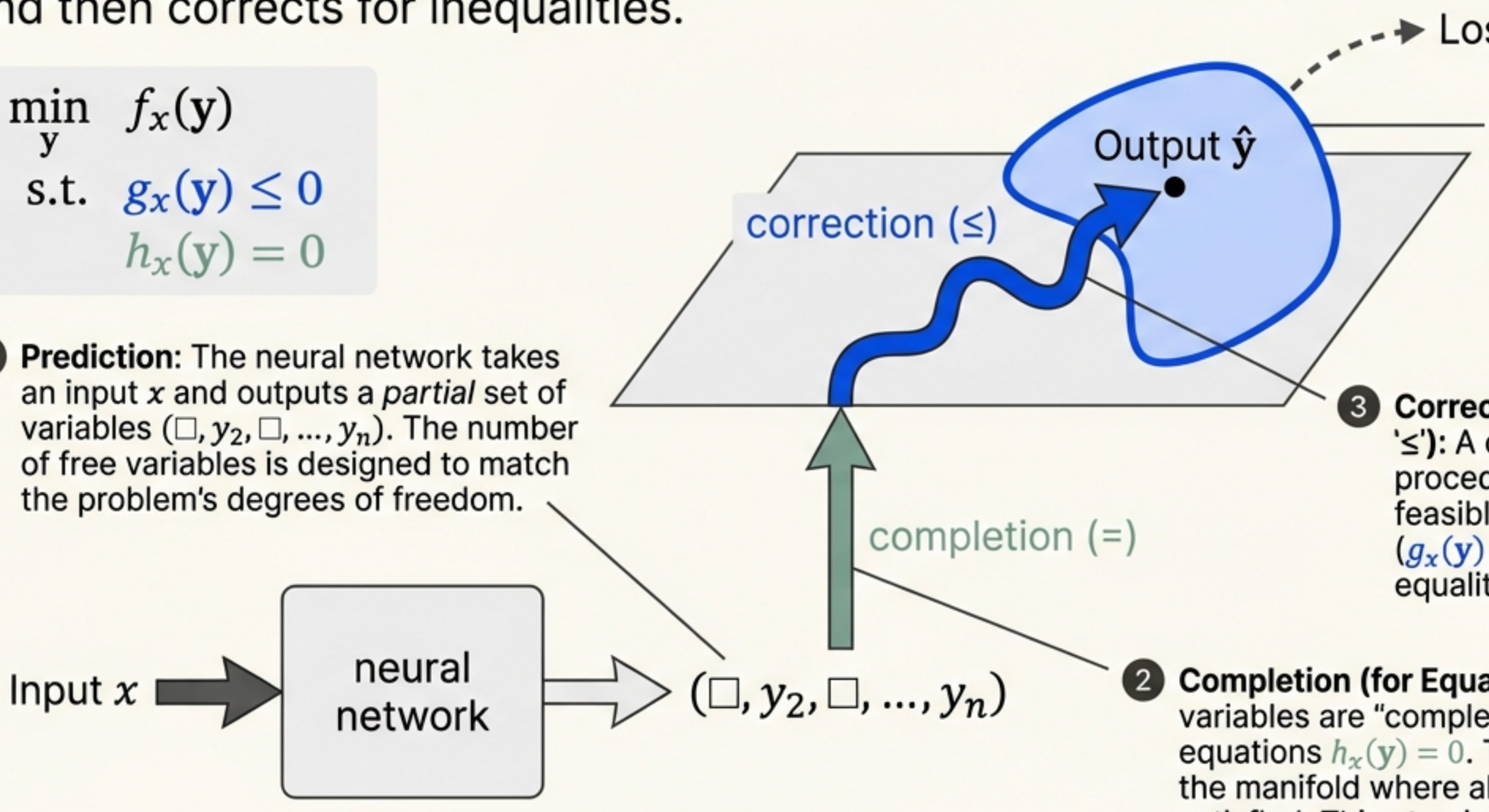
"Our approach embeds differentiable operations into the training of the neural network to ensure feasibility."
— Donti, Rolnick & Kolter, ICLR 2021.



The DC3 Philosophy: Completion and Correction

DC3 enforces constraints through a two-stage process that first satisfies equalities and then corrects for inequalities.

$$\min_{\mathbf{y}} \; f_x(\mathbf{y})$$
$$\text{s.t.} \; g_x(\mathbf{y}) \leq 0$$
$$h_x(\mathbf{y}) = 0$$



1. **Prediction**: The neural network takes an input x and outputs a *partial* set of variables $(\square, y_2, \square, \ldots, y_n)$. The number of free variables is designed to match the problem's degrees of freedom.

2. **Completion (for Equality Constraints '='):** The missing variables are "completed" by solving the equality equations $h_x(\mathbf{y}) = 0$. This forces the solution onto the manifold where all equality constraints are satisfied. This step is made differentiable via the implicit function theorem.

3. **Correction (for Inequality Constraints '≤'):** A differentiable, gradient-based procedure 'pulls' the solution into the feasible region for inequalities ($g_x(\mathbf{y}) \leq 0$) while staying on the equality manifold.

4. **Loss:** The final, feasible output $\hat{\mathbf{y}}$ is used to compute a soft loss $\ell_{\text{soft}}(\hat{\mathbf{y}})$, which guides the training of the initial neural network.

DC3 Example — Family of Convex QPs

$$\underset{y\in\mathbb{R}^n}{\text{minimize}} \ \frac{1}{2}y^TQy + p^Ty, \ \text{ s.t. } Ay = x, \ Gy \le h,$$

1. NN output $= z$

2. Equality completion. Let $A = [A_1, A_2]$ to rewrite equality constraint as $A_1z + A_2z_{\text{dep}} = x$. Then we have

$$\varphi_x(z) := z_{\text{dep}} = A_2^{-1}x - A_2^{-1}A_1z\,.$$

3.Inequality correction. The output from equality completion is denoted by $y = [z, \varphi_x(z)]$. Let $G = [G_1, G_2]$. Then inequality constraint reads as

$$\begin{aligned} &G_1z + G_2(A_2^{-1}x - A_2^{-1}A_1z) \le h \\ \Longrightarrow\ &(G_1 - G_2A_2^{-1}A_1)z \le h - G_2A_2^{-1}x \end{aligned}$$

Let $G_{\text{eff}} = G_1 - G_2A_2^{-1}A_1$ and $h_{\text{eff}} = h - G_2A_2^{-1}x$. Then

$$\begin{aligned} \Delta z &= 2G_{\text{eff}}^T \max\{G_{\text{eff}}z - h_{\text{eff}}, 0\}; \\ \Delta\varphi_x(z) &= -A_2^{-1}A_1\Delta z\,. \end{aligned}$$

The inequality correction then update y by making $t_{\text{train}} = 10$ gradient steps given by

$$\begin{bmatrix} z^+ \\ \varphi_x(z)^+ \end{bmatrix} \leftarrow \begin{bmatrix} z \\ \varphi_x(z) \end{bmatrix} - \gamma \begin{bmatrix} \Delta z \\ \Delta\varphi_x(z) \end{bmatrix}$$

Numerical Results

Convex QPs

	Obj. value	Max eq.	Mean eq.	Max ineq.	Mean ineq.	Time (s)
Optimizer (OSQP)	-15.05 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.002 (0.000)
Optimizer (qpth)	-15.05 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	1.335 (0.012)
DC3	-13.46 (0.01)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.017 (0.001)
DC3, $\neq$	-12.58 (0.04)	0.35 (0.00)	0.13 (0.00)	0.00 (0.00)	0.00 (0.00)	0.008 (0.000)
DC3, $\not\leq$ train	-1.39 (0.97)	0.00 (0.00)	0.00 (0.00)	0.02 (0.02)	0.00 (0.00)	0.017 (0.000)
DC3, $\not\leq$ train/test	-1.23 (1.21)	0.00 (0.00)	0.00 (0.00)	0.09 (0.13)	0.01 (0.01)	0.001 (0.000)
DC3, no soft loss	-21.84 (0.00)	0.00 (0.00)	0.00 (0.00)	23.83 (0.11)	4.04 (0.01)	0.017 (0.000)
NN	-12.57 (0.01)	0.35 (0.00)	0.13 (0.00)	0.00 (0.00)	0.00 (0.00)	0.001 (0.000)
NN, $\leq$ test	-12.57 (0.01)	0.35 (0.00)	0.13 (0.00)	0.00 (0.00)	0.00 (0.00)	0.008 (0.000)
Eq. NN	-9.16 (0.75)	0.00 (0.00)	0.00 (0.00)	8.83 (0.72)	0.91 (0.09)	0.001 (0.000)
Eq. NN, $\leq$ test	-14.68 (0.05)	0.00 (0.00)	0.00 (0.00)	0.89 (0.05)	0.07 (0.01)	0.018 (0.001)

Table 1: Results on QP task for 100 variables, 50 equality constraints, and 50 inequality constraints.

Numerical Results

Linearly Constrained Non-Convex Programs

$$\underset{y \in \mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2} y^T Q y + p^T \sin(y), \quad \text{s. t. } Ay = x, \; Gy \leq h,$$

	Obj. value	Max eq.	Mean eq.	Max ineq.	Mean ineq.	Time (s)
Optimizer	-11.59 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.121 (0.000)
DC3	-10.66 (0.03)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.013 (0.000)
DC3, $\neq$	-10.04 (0.02)	0.35 (0.00)	0.13 (0.00)	0.00 (0.00)	0.00 (0.00)	0.009 (0.000)
DC3, $\not\leq$ train	-0.29 (0.67)	0.00 (0.00)	0.00 (0.00)	0.01 (0.01)	0.00 (0.00)	0.010 (0.004)
DC3, $\not\leq$ train/test	-0.27 (0.67)	0.00 (0.00)	0.00 (0.00)	0.03 (0.03)	0.00 (0.00)	0.001 (0.000)
DC3, no soft loss	-13.81 (0.00)	0.00 (0.00)	0.00 (0.00)	15.21 (0.04)	2.33 (0.01)	0.013 (0.000)
NN	-10.02 (0.01)	0.35 (0.00)	0.13 (0.00)	0.00 (0.00)	0.00 (0.00)	0.001 (0.000)
NN, $\leq$ test	-10.02 (0.01)	0.35 (0.00)	0.13 (0.00)	0.00 (0.00)	0.00 (0.00)	0.009 (0.000)
Eq. NN	-3.88 (0.56)	0.00 (0.00)	0.00 (0.00)	6.87 (0.43)	0.72 (0.05)	0.001 (0.000)
Eq. NN, $\leq$ test	-10.99 (0.03)	0.00 (0.00)	0.00 (0.00)	0.87 (0.04)	0.06 (0.00)	0.013 (0.000)

Numerical Results

AC Optimal Power Flow (57-node power system)

$$\begin{aligned} \underset{p_g \in \mathbb{R}^b,\ q_g \in \mathbb{R}^b,\ v \in \mathbb{C}^b}{\text{minimize}} \quad & p_g^T A p_g + b^T p_g \\ \text{subject to} \quad & p_g^{\min} \leq p_g \leq p_g^{\max}, \quad q_g^{\min} \leq q_g \leq q_g^{\max}, \quad v^{\min} \leq |v| \leq v^{\max}, \\ & (p_g - p_d) + (q_g - q_d)i = \operatorname{diag}(v)\overline{W}\overline{v}. \end{aligned}$$

	Obj. value	Max eq.	Mean eq.	Max ineq.	Mean ineq.	Time (s)
Optimizer	3.81 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.949 (0.002)
DC3	3.82 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.089 (0.000)
DC3, $\neq$	3.67 (0.01)	0.14 (0.01)	0.02 (0.00)	0.00 (0.00)	0.00 (0.00)	0.040 (0.000)
DC3, $\not\leq$ train	3.82 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.00 (0.00)	0.089 (0.000)
DC3, $\not\leq$ train/test	3.82 (0.00)	0.00 (0.00)	0.00 (0.00)	0.01 (0.00)	0.00 (0.00)	0.039 (0.000)
DC3, no soft loss	3.11 (0.05)	2.60 (0.35)	0.07 (0.00)	2.33 (0.33)	0.03 (0.01)	0.088 (0.000)
NN	3.69 (0.02)	0.19 (0.01)	0.03 (0.00)	0.00 (0.00)	0.00 (0.00)	0.001 (0.000)
NN, $\leq$ test	3.69 (0.02)	0.16 (0.00)	0.02 (0.00)	0.00 (0.00)	0.00 (0.00)	0.040 (0.000)
Eq. NN	3.81 (0.00)	0.00 (0.00)	0.00 (0.00)	0.15 (0.01)	0.00 (0.00)	0.039 (0.000)
Eq. NN, $\leq$ test	3.81 (0.00)	0.00 (0.00)	0.00 (0.00)	0.15 (0.01)	0.00 (0.00)	0.078 (0.000)

Out of 100 test instances, there were 3 on which DC3 output lower-than-optimal objective values of up to a few percent (-0.30%, -1.85%, -5.51%), reflecting slight constraint violations. Over the other 97 instances, the per-instance optimality gap compared to the classical optimizer was 0.22%.

DC3 — Discussion Questions

- What are the applications that have natural split of free v.s. dependent variables? Is there also a natural equality completion process?
- When would you prefer DC3 over traditional solvers or pure learning approaches?
- What are the applications where “fast approximate solutions with occasional violations” is more valuable than “slower exact solutions”?
- Are there non-iterative based differentiable projection layers?

Quiz Questions

Required reading: OptNet

An OptNet layer takes the form

$$
\begin{aligned}
z_{i+1} = \underset{z}{\text{argmin}} \quad & \frac{1}{2} z^T Q(z_i) z + q(z_i)^T z \\
\text{subject to} \quad & A(z_i) z = b(z_i) \\
& G(z_i) z \leq h(z_i)
\end{aligned}
$$

where z_i is the input to the layer and z_{i+1} is the output of the layer.

- What are the learnable parameters for the optimization layer in OptNet?
  Coefficients matrices/vectors in QP problem.

- What relation between optimal primal and dual solutions is used to derive the formula for backpropagation?
  KKT condition