Learning to Optimize via Implicit Networks

Samy Wu Fung[1]

Joint work with Howard Heaton[2], Daniel McKenzie[1], Qiuwei Li[3] Wotao Yin[3], Stanley Osher[4]

Colorado School of Mines[1], Typal Academy[2], Alibaba Group[3], UCLA[4]

Deep Learning

$$x^\star_d = \sigma(W^m \cdot + b^m) \circ \cdots \circ \sigma(W^1 d + b^1)$$

[1] Heaton and Wu Fung (2023). "Explainable AI via Learning to Optimize." Scientific Reports

Motivation: ML vs. Optimization Models

Deep Learning

$$x_d^\star = \sigma(W^m \cdot + b^m) \circ \cdots \circ \sigma(W^1 d + b^1)$$

✓ data driven

✓ flexible architectures

✓ expressive capacity

satisfy constraints / optimality

[1] Heaton and Wu Fung (2023). "Explainable AI via Learning to Optimize." Scientific Reports

Motivation: ML vs. Optimization Models

Deep Learning

$x_{d}^{\star} = \sigma(W^{m} \cdot + b^{m}) \circ \cdots \circ \sigma(W^{1} d + b^{1})$

✓ data driven

✓ flexible architectures

✓ expressive capacity

satisfy constraints / optimality

Optimization

$x_{d}^{\star} = \arg\min_{x \in C} f(x; d)$

[1] Heaton and Wu Fung (2023). "Explainable AI via Learning to Optimize." Scientific Reports

Deep Learning

$x^\star_{d} = \sigma(W^m \cdot + b^m) \circ \cdots \circ \sigma(W^1 d + b^1)$

✓ data driven

✓ flexible architectures

✓ expressive capacity

satisfy constraints / optimality

Optimization

$x^\star_{d} = \arg\min_{x \in C} f(x; d)$

✓ scalable algorithms

✓ guaranteed feasibility/optimality

✓ interpretable models

leverage data

[1] Heaton and Wu Fung (2023). "Explainable AI via Learning to Optimize." Scientific Reports

Deep Learning

$x_d^\star = \sigma(W^m \cdot + b^m) \circ \cdots \circ \sigma(W^1 d + b^1)$

✓ data driven

✓ flexible architectures

✓ expressive capacity

satisfy constraints / optimality

Optimization

$x_d^\star = \arg\min_{x \in C} f(x; d)$

✓ scalable algorithms

✓ guaranteed feasibility/optimality

✓ interpretable models

leverage data

Today:

- Design and efficiently train network architectures that benefit from optimization theory. $\Longrightarrow$ NN outputs *satisfy optimality/feasibility* conditions.[1]

[1] Heaton and Wu Fung (2023). "Explainable AI via Learning to Optimize." Scientific Reports

Learning to Optimize

Make a model by parameterizing an optimization problem via Θ to get

$$x_{\Theta,d} = \arg\min_{x \in C} f_\Theta(x; d) \tag{1}$$

Note: Constraints/analytic cost functions can be included by domain experts.

[2] El Ghaoui, Laurent, et al. (2021) "Implicit deep learning." SIMODS

[3] Bai et al (2019) "Deep Equilibrium Models" NeurIPS '19

L2O Description and Forward Propagation

Make a model by parameterizing an optimization problem via Θ to get

$$x_{\Theta,d} = \operatorname*{arg\,min}_{x\in C} f_\Theta(x;d) \quad \iff \quad x_{\Theta,d} = T_\Theta(x_{\Theta,d};d) \tag{1}$$

Note: Constraints/analytic cost functions can be included by domain experts.

[2] El Ghaoui, Laurent, et al. (2021) "Implicit deep learning." SIMODS

[3] Bai et al (2019) "Deep Equilibrium Models" NeurIPS '19

L2O Description and Forward Propagation

Make a model by parameterizing an optimization problem via Θ to get

$$x_{\Theta,d} = \arg\min_{x\in C} f_\Theta(x;d) \quad \Longleftrightarrow \quad x_{\Theta,d} = T_\Theta(x_{\Theta,d};d) \tag{1}$$

Note: Constraints/analytic cost functions can be included by domain experts.

[2] El Ghaoui, Laurent, et al. (2021) "Implicit deep learning." SIMODS

[3] Bai et al (2019) "Deep Equilibrium Models" NeurIPS '19

Make a model by parameterizing an optimization problem via Θ to get

$$x_{\Theta,d} = \arg\min_{x\in C} f_\Theta(x;d) \quad \iff \quad x_{\Theta,d} = T_\Theta(x_{\Theta,d};d) \tag{1}$$

Note: Constraints/analytic cost functions can be included by domain experts.

Example: projected gradient descent: $T_\Theta(x^k;d) = P_C(x^k - \alpha\nabla_x f_\Theta(x^k;d))$

[2] El Ghaoui, Laurent, et al. (2021) "Implicit deep learning." SIMODS

[3] Bai et al (2019) "Deep Equilibrium Models" NeurIPS '19

Make a model by parameterizing an optimization problem via Θ to get

$$x_{\Theta,d} = \arg\min_{x \in C} f_\Theta(x; d) \quad \Longleftrightarrow \quad x_{\Theta,d} = T_\Theta(x_{\Theta,d}; d) \tag{1}$$

Note: Constraints/analytic cost functions can be included by domain experts.

Example: projected gradient descent: $T_\Theta(x^k; d) = P_C(x^k - \alpha \nabla_x f_\Theta(x^k; d))$

Forward propagation consists of applying an iterative (or fixed point) algorithm repeatedly

$$x^{k+1} = T_\Theta(x^k; d), \quad k = 1, \ldots, K \tag{2}$$

[2] El Ghaoui, Laurent, et al. (2021) "Implicit deep learning." SIMODS

[3] Bai et al (2019) "Deep Equilibrium Models" NeurIPS '19

L2O Description and Forward Propagation

Make a model by parameterizing an optimization problem via Θ to get

$$x_{\Theta,d} = \operatorname*{arg\,min}_{x \in C} f_\Theta(x; d) \quad \iff \quad x_{\Theta,d} = T_\Theta(x_{\Theta,d}; d) \tag{1}$$

Note: Constraints/analytic cost functions can be included by domain experts.

Example: projected gradient descent: $T_\Theta(x^k; d) = P_C(x^k - \alpha \nabla_x f_\Theta(x^k; d))$

Forward propagation consists of applying an iterative (or fixed point) algorithm repeatedly

$$x^{k+1} = T_\Theta(x^k; d), \quad k = 1, \dots, K \tag{2}$$

- Fixed number of iterations, e.g., $K = 10 \implies$ unrolled network

[2] El Ghaoui, Laurent, et al. (2021) "Implicit deep learning." SIMODS

[3] Bai et al (2019) "Deep Equilibrium Models" NeurIPS '19

L2O Description and Forward Propagation

Make a model by parameterizing an optimization problem via Θ to get

$$x_{\Theta,d} = \underset{x\in C}{\arg\min}\, f_\Theta(x;d) \quad \Longleftrightarrow \quad x_{\Theta,d} = T_\Theta(x_{\Theta,d};d) \tag{1}$$

Note: Constraints/analytic cost functions can be included by domain experts.

Example: projected gradient descent: $T_\Theta(x^k;d) = P_C(x^k - \alpha\nabla_x f_\Theta(x^k;d))$

Forward propagation consists of applying an iterative (or fixed point) algorithm repeatedly

$$x^{k+1} = T_\Theta(x^k;d), \quad k = 1,\dots,K \tag{2}$$

- Fixed number of iterations, e.g., $K = 10 \implies$ unrolled network
- Iterate until convergence $(K \gg 1) \implies$ **Implicit Networks**[2] [3] (outputs satisfy optimality/feasibility guarantees!)

[2] El Ghaoui, Laurent, et al. (2021) "Implicit deep learning." SIMODS

[3] Bai et al (2019) "Deep Equilibrium Models" NeurIPS '19

Assumption 1: Contractiveness of T_Θ

For fixed d, there exists $\gamma \in [0, 1)$ such that

$$\|T_\Theta(x_1; d) - T_\Theta(x_2; d)\| \leq \gamma \|x_1 - x_2\|, \quad \text{for all } x_1, x_2 \in \mathcal{U}.$$

Assumption 1: Contractiveness of T_Θ

For fixed d, there exists $\gamma \in [0,1)$ such that

$$\|T_\Theta(x_1; d) - T_\Theta(x_2; d)\| \leq \gamma \|x_1 - x_2\|, \quad \text{for all } x_1, x_2 \in \mathcal{U}.$$

Banach Fixed Point Theorem

For any $x^1 \in \mathcal{U}$, if Assumption 1 holds, then the sequence $\{x^k\}$ generated by

$$x^{k+1} = T_\Theta(x^k;\ d), \quad \text{for all } k \in \mathbb{N}, \tag{3}$$

converges linearly to the unique fixed point $x_d^\star$.

Well-Defined Implicit Network

Assumption 1: Contractiveness of T_Θ

For fixed d, there exists $\gamma \in [0, 1)$ such that

$$\|T_\Theta(x_1; d) - T_\Theta(x_2; d)\| \leq \gamma \|x_1 - x_2\|, \quad \text{for all } x_1, x_2 \in \mathcal{U}.$$

Banach Fixed Point Theorem

For any $x^1 \in \mathcal{U}$, if Assumption 1 holds, then the sequence $\{x^k\}$ generated by

$$x^{k+1} = T_\Theta(x^k;\ d), \quad \text{for all } k \in \mathbb{N}, \tag{3}$$

converges linearly to the unique fixed point $x_d^\star$.

Remark: If T_Θ is a contraction in x, we have well-defined implicit network

Training Implicit Networks

Implicit L2O: The Learning Problem

- Given data $\{(d^i, x^\star_{d^i})\}$ the training problem is given by

$$\min_{\Theta} \quad {}_{i} \quad \ell \quad x_{\Theta,d^i}, x^\star_{d^i} \tag{4}$$

Implicit L2O: The Learning Problem

- Given data $\{(d^i, x^\star_{d^i})\}$ the training problem is given by

$$\min_{\Theta} \quad {}_{i} \quad \ell \quad x_{\Theta,d^i}, x^\star_{d^i} \tag{4}$$

where

- $x_{\Theta,d^i} = \arg\min_{x\in C} f_\Theta(x; d^i)$

- Given data $\{(d^i, x^\star_{d^i})\}$ the training problem is given by

$$\min_{\Theta} \sum_{i} \ell\left(x_{\Theta,d^i}, x^\star_{d^i} \right) \tag{4}$$

  where

  - $x_{\Theta,d^i} = \arg\min_{x \in C} f_\Theta(x; d^i)$
  - or equivalently: $x_{\Theta,d^i} = T_\Theta(x_{\Theta,d^i}; d^i)$

- Given data $\{(d^i, x^\star_{d^i})\}$ the training problem is given by

$$\min_{\Theta} \quad {}_{i} \quad \ell \quad x_{\Theta,d^i}, x^\star_{d^i} \tag{4}$$

  where

  - $x_{\Theta,d^i} = \arg\min_{x\in C} f_\Theta(x; d^i)$
  - or equivalently: $x_{\Theta,d^i} = T_\Theta(x_{\Theta,d^i}; d^i)$

- $\dfrac{d\ell}{d\Theta} = \dfrac{d\ell}{dx_\Theta}\dfrac{dx_\Theta}{d\Theta}$. How to compute $\dfrac{dx_\Theta}{d\Theta}$?

Backpropagating Through Implicit Networks

Standard Backpropagation

To train the implicit L2O network, we need to compute the gradient of the loss function

$$\frac{d}{d\Theta}[\ell(x_{\Theta,d}, x_d)] = \frac{d\ell}{dx}\frac{dx_{\Theta,d}}{d\Theta}.$$

Standard Backpropagation

To train the implicit L2O network, we need to compute the gradient of the loss function

$$\frac{d}{d\Theta}[\ell(x_{\Theta,d}, x_d)] = \frac{d\ell}{dx}\frac{dx_{\Theta,d}}{d\Theta}.$$

If network converges in K iterations, we need to apply the chain rule K times:

Standard Backpropagation

To train the implicit L2O network, we need to compute the gradient of the loss function

$$\frac{d}{d\Theta}[\ell(x_{\Theta,d}, x_d)] = \frac{d\ell}{dx}\frac{dx_{\Theta,d}}{d\Theta}.$$

If network converges in K iterations, we need to apply the chain rule K times:

$$\begin{aligned}
\frac{dx_\Theta(d)}{d\Theta} = \frac{dx_\Theta^K}{d\Theta} &= \frac{dx_\Theta^K}{dx_\Theta^{K-1}}\frac{dx_\Theta^{K-1}}{d\Theta} + \frac{\partial x^K}{\partial\Theta} \\
\frac{dx_\Theta^{K-1}}{d\Theta} &= \frac{dx_\Theta^{K-1}}{dx_\Theta^{K-2}}\frac{dx_\Theta^{K-2}}{d\Theta} + \frac{\partial x_\Theta^{K-1}}{\partial\Theta}
\end{aligned}$$

Standard Backpropagation

To train the implicit L2O network, we need to compute the gradient of the loss function

$$\frac{d}{d\Theta}[\ell(x_{\Theta,d}, x_d)] = \frac{d\ell}{dx}\frac{dx_{\Theta,d}}{d\Theta}.$$

If network converges in K iterations, we need to apply the chain rule K times:

$$\frac{dx_\Theta(d)}{d\Theta} = \frac{dx_\Theta^K}{d\Theta} = \frac{dx_\Theta^K}{dx_\Theta^{K-1}}\frac{dx_\Theta^{K-1}}{d\Theta} + \frac{\partial x^K}{\partial\Theta}$$

$$\frac{dx_\Theta^{K-1}}{d\Theta} = \frac{dx_\Theta^{K-1}}{dx_\Theta^{K-2}}\frac{dx_\Theta^{K-2}}{d\Theta} + \frac{\partial x_\Theta^{K-1}}{\partial\Theta}$$

$$\vdots$$

$$\frac{dx_\Theta^2}{d\Theta} = \frac{dx_\Theta^2}{dx_\Theta^1}\frac{dx_\Theta^1}{d\Theta} + \frac{\partial x_\Theta^2}{\partial\Theta}$$

Standard Backpropagation

To train the implicit L2O network, we need to compute the gradient of the loss function

$$\frac{d}{d\Theta}[\ell(x_{\Theta,d}, x_d)] = \frac{d\ell}{dx}\frac{dx_{\Theta,d}}{d\Theta}.$$

If network converges in K iterations, we need to apply the chain rule K times:

$$\begin{aligned}
\frac{dx_\Theta(d)}{d\Theta} = \frac{dx_\Theta^K}{d\Theta} &= \frac{dx_\Theta^K}{dx_\Theta^{K-1}}\frac{dx_\Theta^{K-1}}{d\Theta} + \frac{\partial x^K}{\partial\Theta} \\
\frac{dx_\Theta^{K-1}}{d\Theta} &= \frac{dx_\Theta^{K-1}}{dx_\Theta^{K-2}}\frac{dx_\Theta^{K-2}}{d\Theta} + \frac{\partial x_\Theta^{K-1}}{\partial\Theta} \\
&\vdots \\
\frac{dx_\Theta^2}{d\Theta} &= \frac{dx_\Theta^2}{dx_\Theta^1}\frac{dx_\Theta^1}{d\Theta} + \frac{\partial x_\Theta^2}{\partial\Theta}
\end{aligned}$$

- memory requirements grow linearly in depth ($\mathcal{O}(K)$) $\implies$ intractable for implicit networks

Recalling the output of an implicit network is the fixed point of T_Θ:

$$x_{\Theta,d} = T_\Theta(x_{\Theta,d}; d),$$

Implicit Jacobian-based Backpropagation

Recalling the output of an implicit network is the fixed point of T_Θ:

$$x_{\Theta,d} = T_\Theta(x_{\Theta,d}; d),$$

Computation of its gradient can be done using the Implicit Function Theorem:

Recalling the output of an implicit network is the fixed point of T_Θ:

$$x_{\Theta,d} = T_\Theta(x_{\Theta,d}; d),$$

Computation of its gradient can be done using the Implicit Function Theorem:

$$\frac{dx_\Theta}{d\Theta} = \frac{dT_\Theta}{dx}\frac{dx_\Theta}{d\Theta} + \frac{\partial T_\Theta}{\partial\Theta}$$

Implicit Jacobian-based Backpropagation

Recalling the output of an implicit network is the fixed point of T_Θ:

$$x_{\Theta,d} = T_\Theta(x_{\Theta,d}; d),$$

Computation of its gradient can be done using the Implicit Function Theorem:

$$\begin{aligned} \frac{dx_\Theta}{d\Theta} &= \frac{dT_\Theta}{dx}\frac{dx_\Theta}{d\Theta} + \frac{\partial T_\Theta}{\partial\Theta} \\ \implies \frac{dx_\Theta}{d\Theta} &= \mathcal{J}_\Theta^{-1}\frac{\partial T_\Theta}{\partial\Theta}, \quad \text{where} \quad \mathcal{J}_\Theta \triangleq I - \frac{dT_\Theta}{dx}. \end{aligned}$$

Recalling the output of an implicit network is the fixed point of T_Θ:

$$x_{\Theta,d} = T_\Theta(x_{\Theta,d}; d),$$

Computation of its gradient can be done using the Implicit Function Theorem:

$$\begin{aligned} \frac{dx_\Theta}{d\Theta} &= \frac{dT_\Theta}{dx}\frac{dx_\Theta}{d\Theta} + \frac{\partial T_\Theta}{\partial \Theta} \\ \implies \frac{dx_\Theta}{d\Theta} &= \mathcal{J}_\Theta^{-1}\frac{\partial T_\Theta}{\partial \Theta}, \quad \text{where} \quad \mathcal{J}_\Theta \triangleq I - \frac{dT_\Theta}{dx}. \end{aligned}$$

- Memory required to compute $\frac{dx_{\Theta,d}}{\mathrm{d}\Theta}$ is *constant* in depth! ✓

Recalling the output of an implicit network is the fixed point of T_Θ:

$$x_{\Theta,d} = T_\Theta(x_{\Theta,d}; d),$$

Computation of its gradient can be done using the Implicit Function Theorem:

$$\begin{aligned} \frac{dx_\Theta}{d\Theta} &= \frac{dT_\Theta}{dx}\frac{dx_\Theta}{d\Theta} + \frac{\partial T_\Theta}{\partial \Theta} \\ \implies \frac{dx_\Theta}{d\Theta} &= \mathcal{J}_\Theta^{-1}\frac{\partial T_\Theta}{\partial \Theta}, \quad \text{where} \quad \mathcal{J}_\Theta \triangleq I - \frac{dT_\Theta}{dx}. \end{aligned}$$

- Memory required to compute $\frac{dx_{\Theta,d}}{\mathrm{d}\Theta}$ is *constant* in depth! ✓
- Unfortunately, solving for $\mathcal{J}_\Theta^{-1}\frac{\partial T}{\partial \Theta}$ is often very expensive

Jacobian-free Backpropagation (JFB)

Proposed: Jacobian-free Backprop (JFB)

Goal: Avoid solving Jacobian-based equation and memory issues when training implicit networks.

[4] Wu Fung et al. (2022) "JFB: Jacobian-Free Backpropagation for Implicit Networks." AAAI 22

Proposed: Jacobian-free Backprop (JFB)

Goal: Avoid solving Jacobian-based equation and memory issues when training implicit networks.

Key Idea: replace J_Θ with identity[4].

[4] Wu Fung et al. (2022) "JFB: Jacobian-Free Backpropagation for Implicit Networks." AAAI 22

Proposed: Jacobian-free Backprop (JFB)

Goal: Avoid solving Jacobian-based equation and memory issues when training implicit networks.

Key Idea: replace J_Θ with identity[4].

That is, approximate true gradient

$$\frac{d}{d\Theta}[\ell(x_{\Theta,d}, x_d)] = \frac{d\ell}{dx_{\Theta,d}} \mathcal{J}_\Theta^{-1} \frac{\partial T}{\partial \Theta}$$

with

$$p_\Theta = \frac{d\ell}{dx_{\Theta,d}} \frac{\partial T}{\partial \Theta}$$

[4] Wu Fung et al. (2022) "JFB: Jacobian-Free Backpropagation for Implicit Networks." AAAI 22

Proposed: Jacobian-free Backprop (JFB)

Goal: Avoid solving Jacobian-based equation and memory issues when training implicit networks.

Key Idea: replace J_Θ with identity[4].

That is, approximate true gradient

$$\frac{d}{d\Theta}[\ell(x_{\Theta,d}, x_d)] = \frac{d\ell}{dx_{\Theta,d}} \mathcal{J}_\Theta^{-1} \frac{\partial T}{\partial \Theta}$$

with

$$p_\Theta = \frac{d\ell}{dx_{\Theta,d}} \frac{\partial T}{\partial \Theta}$$

Remark: cost equivalent to computing the gradient of *one layer*!

[4] Wu Fung et al. (2022) "JFB: Jacobian-Free Backpropagation for Implicit Networks." AAAI 22

Jacobian-free Backprop (JFB)

Assumption 1

T_Θ is continuously diff'ble w.r.t Θ

Assumption 2

$M = \frac{\partial T}{\partial \Theta}$ has full column rank and is well-conditioned s.t. $\kappa(M^\top M) \leq \frac{1}{\gamma}$

Jacobian-free Backprop (JFB)

Assumption 1

T_Θ is continuously diff'ble w.r.t Θ

Assumption 2

$M = \frac{\partial T}{\partial \Theta}$ has full column rank and is well-conditioned s.t. $\kappa(M^\top M) \leq \frac{1}{\gamma}$

Main Theorem: Descent of JFB

If Assumptions 1 and 2 hold for given Θ and d, then

$$-p_\Theta = -\frac{d\ell}{dx_{\Theta,d}} \frac{\partial T}{\partial \Theta}$$

forms a descent direction for $\ell(x_{\Theta,d}, x_d)$ with respect to Θ.

Implementation of JFB

Implicit Forward + Proposed Backprop

```
x_fxd_pt = find_fixed_point(d)
x_star = apply_T(x_fxd_pt, d)
loss = criterion(x_star, labels)
loss.backward()
optimizer.step()
```

Figure 1: Sample PyTorch code for backpropagation

Remark: backpropagation is simple in PyTorch framework!

JFB works!

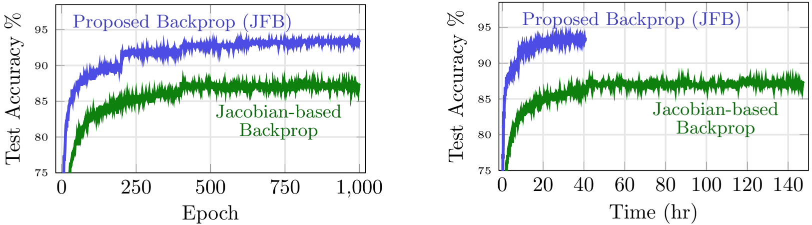
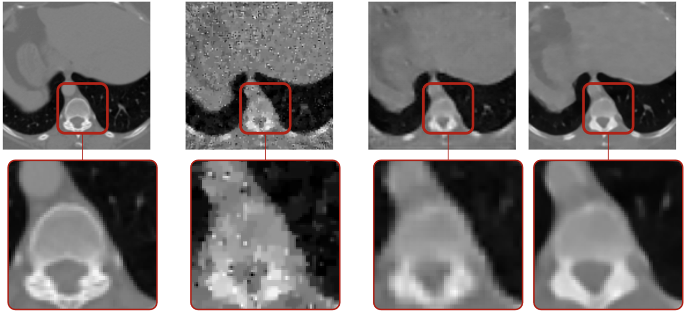


Figure 2: Training an implicit neural network on CIFAR10. JFB is faster and yields higher test accuracy than Jacobian-based backprop.

L2O Experiment: Computed Tomography



Ground truth	TV Min	U-Net	Implicit L2O
SSIM: 1.000	SSIM: 0.379	SSIM: 0.806	SSIM: 0.891
PSNR: ∞	PSNR: 21.17	PSNR: 27.83	PSNR: 31.87
	Fail ✖ – Data Regularization	Fail ✖ – Fidelity	Trustworthy ✔

Comparison of techniques, ranging from traditional to fully data-driven

$$\text{(L2O Model)} = N_\Theta(d) = \operatorname*{arg\,min}_{x\in[0,1]^n} \ f_\Theta(Kx) \quad \text{s.t.} \quad Ax = b \tag{5}$$

Shortest Path Problem: given a graph and origin/destination nodes, find the path that incurs minimal cost.

L2O Experiment: Shortest Path Problem

Shortest Path Problem: given a graph and origin/destination nodes, find the path that incurs minimal cost.

Problem Formulation:

$$x_\Theta(d) \triangleq \underset{x \in \mathcal{C}}{\arg\min}\, w_\Theta(d)^\top x \tag{6}$$

where $\mathcal{C} = \{x\colon \underbrace{Ax = b}_{\mathcal{C}_1},\ \underbrace{x \geq 0}_{\mathcal{C}_2}\}$ encodes origin-destination constraints and flow conservation constraints.

L2O Experiment: Shortest Path Problem

Shortest Path Problem: given a graph and origin/destination nodes, find the path that incurs minimal cost.

Problem Formulation:

$$x_\Theta(d) \triangleq \underset{x \in \mathcal{C}}{\arg\min}\, w_\Theta(d)^\top x \tag{6}$$

where $\mathcal{C} = \{x \colon \underbrace{Ax = b}_{\mathcal{C}_1},\ \underbrace{x \geq 0}_{\mathcal{C}_2}\}$ encodes origin-destination constraints and flow conservation constraints.

Problem: projection onto $\mathcal{C}$ nontrivial $\implies$ fwd prop. and backprop. expensive, even with JFB.

L2O Experiment: Shortest Path Problem

Shortest Path Problem: given a graph and origin/destination nodes, find the path that incurs minimal cost.

Problem Formulation:

$$x_\Theta(d) \triangleq \arg\min_{x\in\mathcal{C}} w_\Theta(d)^\top x \tag{6}$$

where $\mathcal{C} = \{x\colon \underbrace{Ax = b}_{\mathcal{C}_1},\ \underbrace{x \geq 0}_{\mathcal{C}_2}\}$ encodes origin-destination constraints and flow conservation constraints.

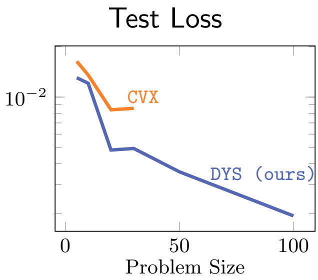
Problem: projection onto $\mathcal{C}$ nontrivial $\implies$ fwd prop. and backprop. expensive, even with JFB.

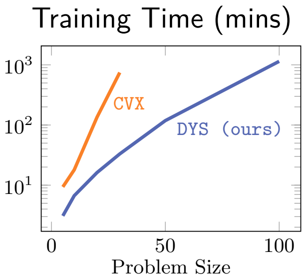
Remedy: Define implicit network using three-operator splitting:

$$T(x;d) = x - P_{\mathcal{C}_1}(x) + P_{\mathcal{C}_2}\left(2P_{\mathcal{C}_1}(x) - x - \nabla f_\Theta(P_{\mathcal{C}_1}(x);d)\right), \tag{7}$$

where note that $P_{\mathcal{C}_1}$ and $P_{\mathcal{C}_2}$ are trivial to compute.

L2O Experiment: Shortest Path Problem





Summary

- **Implicit L2O:** optimization-based network architectures with *guarantees on their outputs*

Summary

- **Implicit L2O:** optimization-based network architectures with *guarantees on their outputs*
- Implicit L2O models can be efficiently trained using Jacobian-Free Backpropagation

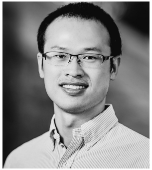
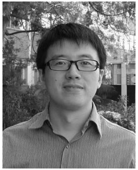
Summary

- **Implicit L2O:** optimization-based network architectures with *guarantees on their outputs*
- Implicit L2O models can be efficiently trained using Jacobian-Free Backpropagation
- Implicit L2O + JFB have found success in traffic flow, computed tomography, and knapsack problem

Summary

- **Implicit L2O:** optimization-based network architectures with *guarantees on their outputs*
- Implicit L2O models can be efficiently trained using Jacobian-Free Backpropagation
- Implicit L2O + JFB have found success in traffic flow, computed tomography, and knapsack problem
- More details can be found in papers:
  - *Explainable AI via Learning to Optimize*, Scientific Reports, 2023
  - *JFB: Jacobian-Free Backpropagation for Implicit Networks*, AAAI, 2022
  - Three Operator Splitting for Learning to Predict Equilibria in Convex Games, SIMODS, 2024
  - Learning to Solve Integer Linear Programs with Davis-Yin Splitting, TMLR, 2024

Collaborators and Acknowledgments

				
Howard Heaton	Daniel McKenzie	Stanley Osher	Qiuwei Li	Wotao Yin
Typal Academy	CO School of Mines	UCLA	Alibaba	Alibaba