# Learning to Optimize

Wotao Yin

Alibaba Damo Academy

CUHK MATH-IMS Applied Mathematics Colloquium Series

# 1.
# Background

# Machine learning and Optimization

Answers are given as existing data

ML learns from data to give answers in the future

**Induction**

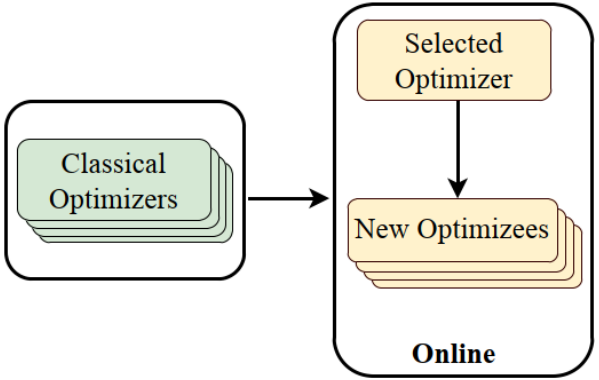
No answer is given; but we know how to evaluate how answers are.

OPT finds answers with best evaluations

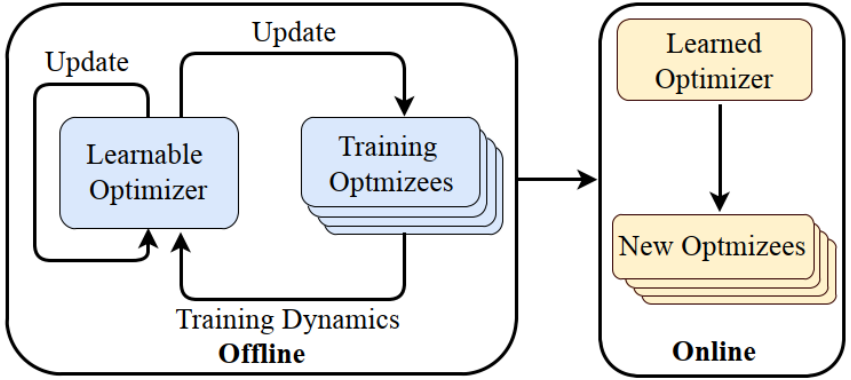
**Prescription**

L2O uses experience to “optimize faster” or “generate better solutions” in the future.

## Classic optimization




## Learning-to-optimize



# When to consider L2O

- Having **many samples** of good solutions
- But it is **hard** to write a **good analytic model** (e.g., inverse problems)

L2O finds better solutions by **learning a model or method**.

- Having **many samples** of good solutions
- Solving similar optimization problems **repeatedly**

L2O finds similar solutions by taking a **“fast shortcut”**.

# Basic formulation

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

- Consider GD iteration: $\mathbf{x}_{t+1} = \mathbf{x}_t - \alpha \nabla f(\mathbf{x}_t)$

# Basic formulation

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

- Consider GD iteration: $\mathbf{x}_{t+1} = \mathbf{x}_t - \alpha \nabla f(\mathbf{x}_t)$
- Introduce free parameters
  - Let $\mathbf{z}_t$ represent all the iterates and gradients till $t$
  - Use $\mathbf{x}_{t+1} = \mathbf{x}_t - g(\mathbf{z}_t, \phi)$ where $g$ is parameterized by $\phi$

# Basic formulation

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

- Consider GD iteration: $\mathbf{x}_{t+1} = \mathbf{x}_t - \alpha \nabla f(\mathbf{x}_t)$
- Introduce free parameters
  - Let $\mathbf{z}_t$ represent all the iterates and gradients till $t$
  - Use $\mathbf{x}_{t+1} = \mathbf{x}_t - g(\mathbf{z}_t, \phi)$ where $g$ is parameterized by $\phi$
- L2O formulation (Andrychowicz et al'NIPS16):

$$\min_{\phi} \mathbb{E}_{f \in \mathcal{T}} \left[ \sum_{t=1}^{T} w_t f(\mathbf{x}_t) \right]$$

$$\mathbf{x}_{t+1} = \mathbf{x}_t - g(\mathbf{z}_t, \phi), \ t = 1, \ldots, T-1$$

# model-based vs model-free

- $g$ has a form of an existing method or uses it as a starting point
- L2O searches for the **best values of some parameters**
- You may combine this L2O with classic methods in various ways

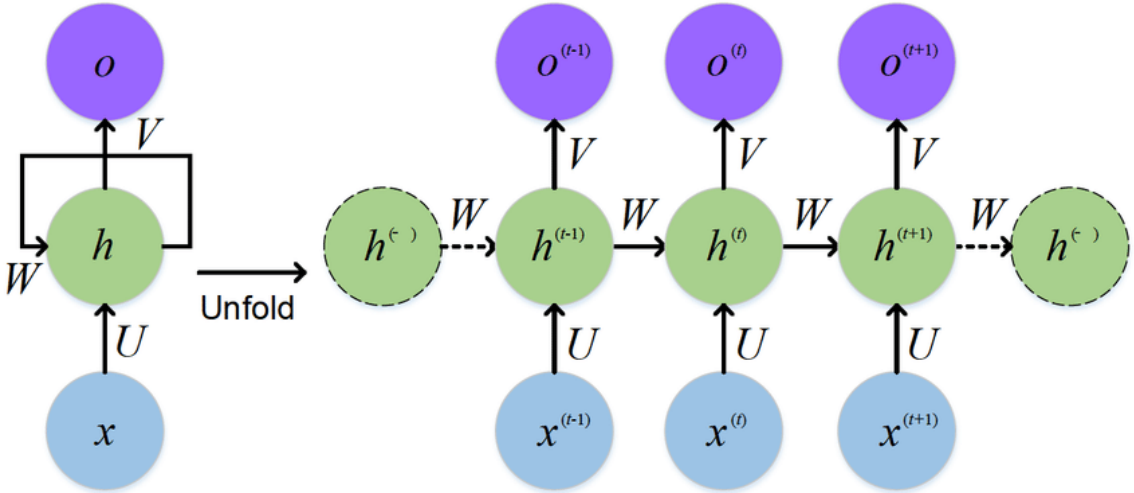
- $g$ is based on universal approximators, e.g., multi-layer neural networks or recurrent neural networks.
- L2O is set to discover **completely new** update rules without referring to any existing updates (other than being iterative)

# 2.

# Model-Free L2O

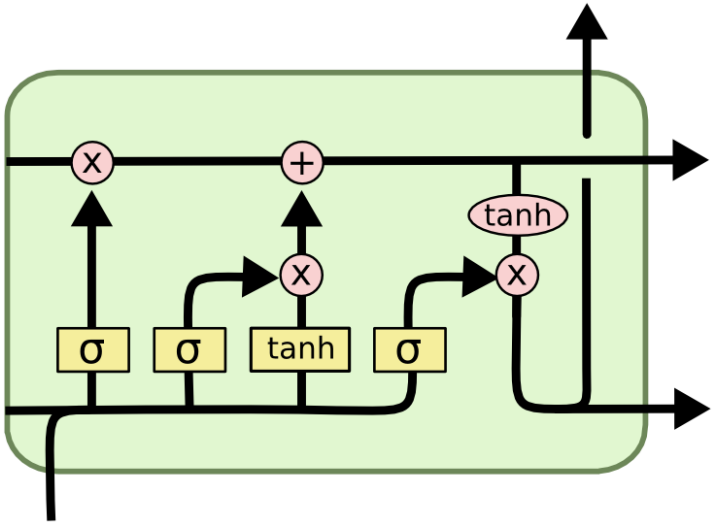
**based on RNNs, especially LSTMs**

# RNN and unfolding




Wichrowska et al'17; Metz et al'ICML19; Li-Malik'ICLR17; Bello et al'ICC17; Jiang et al'18;

# Many model-free L2O uses LSTM

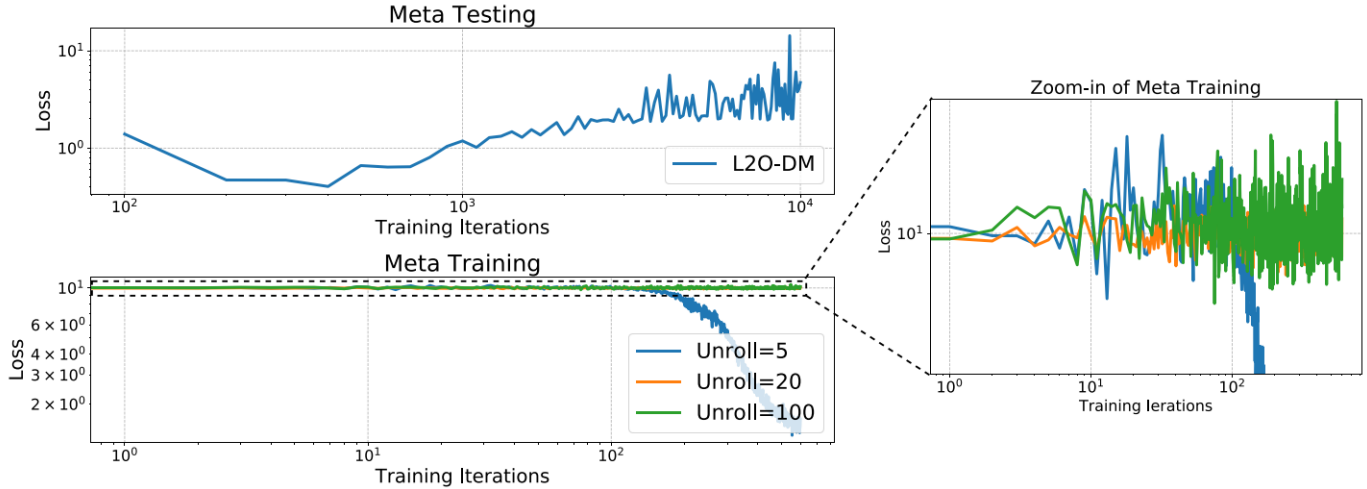



Andrychowicz et al'NIPS16; Chen et al'ICML17; Lv-Jiang-Li'17; Cao et al'NeurIPS19; Xiong-Hsieh'20

| Optimizer Architecture | Input Feature | Meta Training Objective | Additional Technique | Evaluation Metric |
|---|---|---|---|---|
| LSTM | Gradient | Meta Loss | Transform input gradient $\nabla$ into $\log(\nabla)$ and $\text{sign}(\nabla)$ | Training Loss |
| LSTM | Objective Value | Objective Value | N/A | Objective Value |
| LSTM | Gradient | Meta Loss | Random Scaling<br>Combination with Convex Functions | Training Loss |
| Hierarchical RNNs | Scaled averaged gradients,<br>relative log gradient magnitudes,<br>relative log learning rate | Log Meta Loss | Gradient History Attention<br>Nesterov Momentum | Training Loss |
| MLP | Gradient | Meta Loss | Unbiased Gradient Estimators | Training Loss<br>Testing Loss |
| RNN Controller | Loss, Gradient | Meta Loss | Coordinate Groups | Training Loss |
| Searched Mathematical Rule by Primitive Functions | Scaled averaged gradients | Meta Loss | N/A | Testing Accuracy |
| Multiple LSTMs | Gradient, momentum,<br>particle's velocity and attraction | Meta Loss and<br>Entropy Regularizer | Sample- and Feature- Attention | Training Loss |
| RNN | Input Images, Input Gradient | Meta Loss | N/A | Standard and Robust<br>Test Accuracies |
| LSTM | Input Gradient | Meta Loss | N/A | Training Loss and<br>Robust Test Accuracy |

# Challenges: network depth

- Deep networks have high memory costs
- Shallow networks cannot run more iterations



# 3.
# Model-Based L2O

**Plug-n-Play, Unrolling, Safeguarding, etc.**

# Unrolling by example: LASSO

**LASSO** model:

$$x^{\text{lasso}} \leftarrow \underset{x}{\text{minimize}} \ \frac{1}{2}\|b - Ax\|_2^2 + \lambda\|x\|_1$$

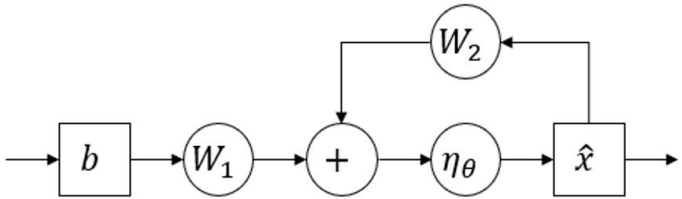
Iterative Shrinkage and Thresholding Algorithm (**ISTA**):

$$x^{(k+1)} = \eta_{\frac{\lambda}{L}}\left(x^{(k)} + \frac{1}{L}A^T(b - Ax^{(k)})\right)$$

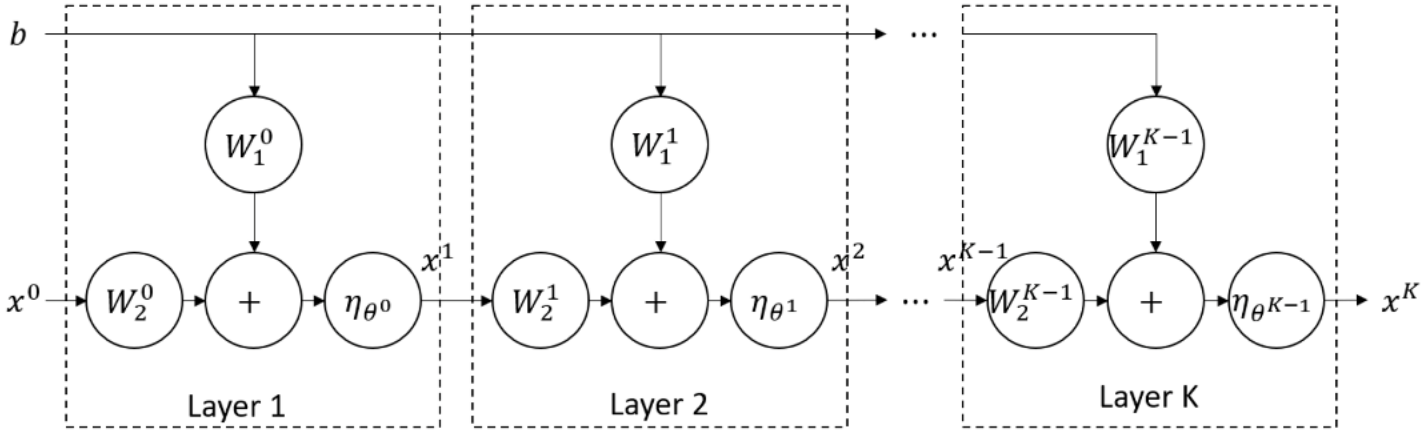
Rewrite ISTA as

$$x^{(k+1)} = \eta_\theta(W_1 b + W_2 x^{(k)}),$$

where $W_1 = \frac{1}{L}A^T, W_2 = I_n - \frac{1}{L}A^T A$ and $\theta = \frac{\lambda}{L}$.



# Unrolling




- Limit to K iterations, trained end-to-end trained
- Okay to reduce parameters without performance loss (Chen et. al. NeurIPS'18 & Liu et. al. ICLR'19)
- Popular and successful in inverse problems, PDEs, and graphical models

# Challenges

- Unroll length: more layers yield better performance but are difficult to train.
- Capacity: only provably better in very narrow cases (Liu et al'ICLR18).
- Trainability: lacks performance guarantee.
- Generalization: when and what if L2O fails?

# Safe-guarding

(Heaton et al.20) L2O convergence can be ensured by incorporating an "energy" $E$

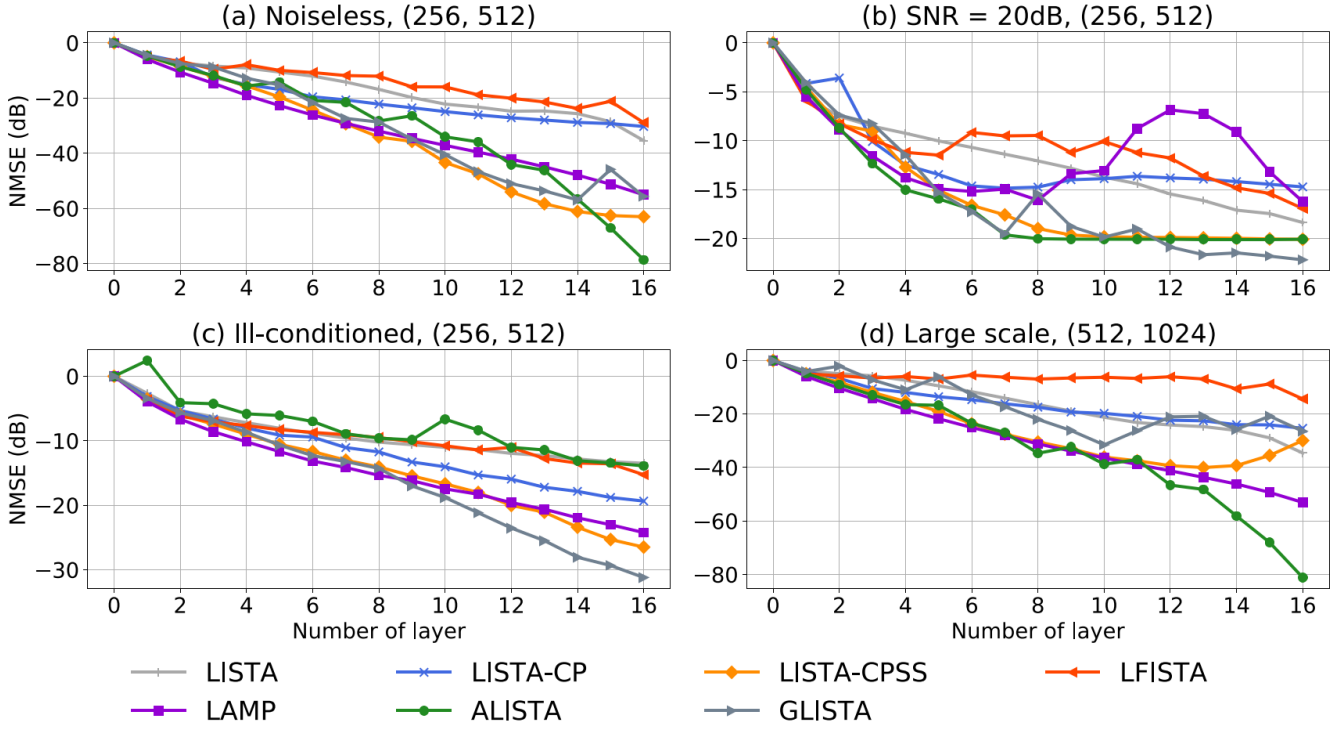
$$x^{k+1} = \begin{cases} \text{L2O update } z^k & \text{if } E^k(z^k) \le E^k(x^k) \\ \text{classic update } T(x^k) & \text{otherwise} \end{cases}$$

When L2O fails to decrease the energy, the classic update $T$ will take over in that iteration

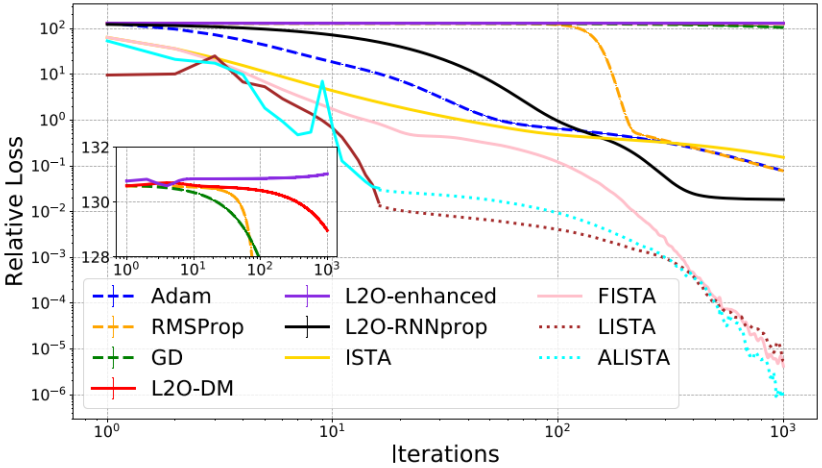
# Test: Recover a sparse vector from noisy measurements $b_q = Ax_q^* + \varepsilon_q$

- Fix $A$; vary sparse vectors and noise
- Training loss is squared-L2 to the true signal
- Unroll to 16 layers

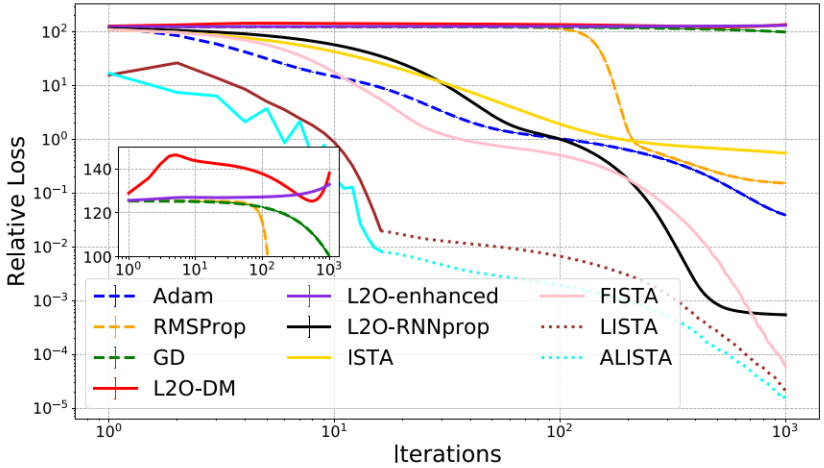
# Learn to recover the signal



# Learn to solve the LASSO model (with safeguards)




(a) $(m,n)=(5,10)$




(b) $(m,n)=(25,50)$

# Plug-and-Play: background

- Consider

$$\underset{x\in\mathbb{R}^d}{\text{minimize}} \quad f(x)+\gamma g(x)$$

- ADMM:

$$\begin{aligned} x^{k+1} &= \text{Prox}_{\sigma^2 g}(y^k - u^k) \\ y^{k+1} &= \text{Prox}_{\alpha f}(x^{k+1} + u^k) \\ u^{k+1} &= u^k + x^{k+1} - y^{k+1}. \end{aligned}$$

- Interpretation in imaging
  - Step 1: noisy image → less noisy image
  - Step 2: less consistent → more consistent with data

# Non-prox denoisers

- State-of-the-art denoisers are prox of certain functions:
  - NLM, BM3D, CNN
- However, still have the interpretation

$$H_\sigma : \text{noisy image} \mapsto \text{less noisy image}$$

- Q: how to integrate into iterations like ADMM?

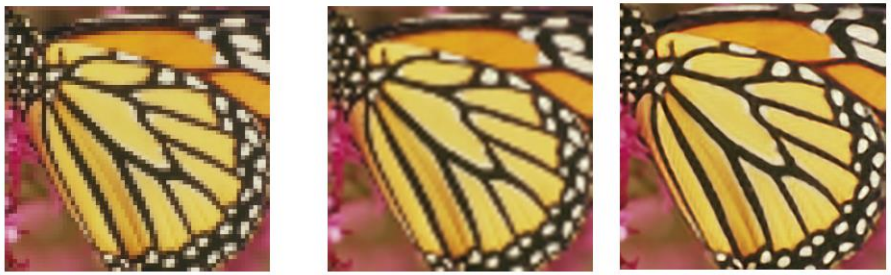
# Plug-and-Play

- (Venkatakrishnan et al'GlobalSIP13) PnP ADMM:

$$
\begin{aligned}
x^{k+1} &= \mathrm{Prox}_{\sigma^2 g}(y^k - u^k) \\
y^{k+1} &= \mathrm{Prox}_{\alpha f}(x^{k+1} + u^k) \\
u^{k+1} &= u^k + x^{k+1} - y^{k+1}.
\end{aligned}
\quad \Rightarrow \quad
\begin{aligned}
x^{k+1} &= H_\sigma(y^k - u^k) \\
y^{k+1} &= \mathrm{Prox}_{\alpha f}(x^{k+1} + u^k) \\
u^{k+1} &= u^k + x^{k+1} - y^{k+1}.
\end{aligned}
$$

- Works surprisingly well!

# Example: Super resolution



**Low resolution input** Other method Other method

Other method Other method Other method **PnP-ADMM with BM3D**

(Chen-Wang-Elgendy'17)

## Good:

- Infinite depth (unrolling length is not an issue)
- Has convergence guarantees (Ryu et al'ICML19) if
  - $I - H_\sigma$ is Lipschitz
  - $f$ is strongly convex

## Limited:

- Denoise $H_\sigma$ is pre-trained before plugged in (Training is not end-to-end)
- Good performance cannot be explained

# Deep Equilibrium (Fixed Point Network)

- (Bai et al'NeurIPS19) Instead of finite iterations, use infinite iterations (in theory) to output a fixed point
- Related to (Chen et al'NeurIPS18) Neural ODE, based on black-box ODE solver
- We can modify (i.e., train) the iterator or ODE model in an end-to-end manner

# Example

- Explicit network

$$u = Q_\Theta(d)$$
$$y = S_\Theta(u)$$

- Fixed-point network

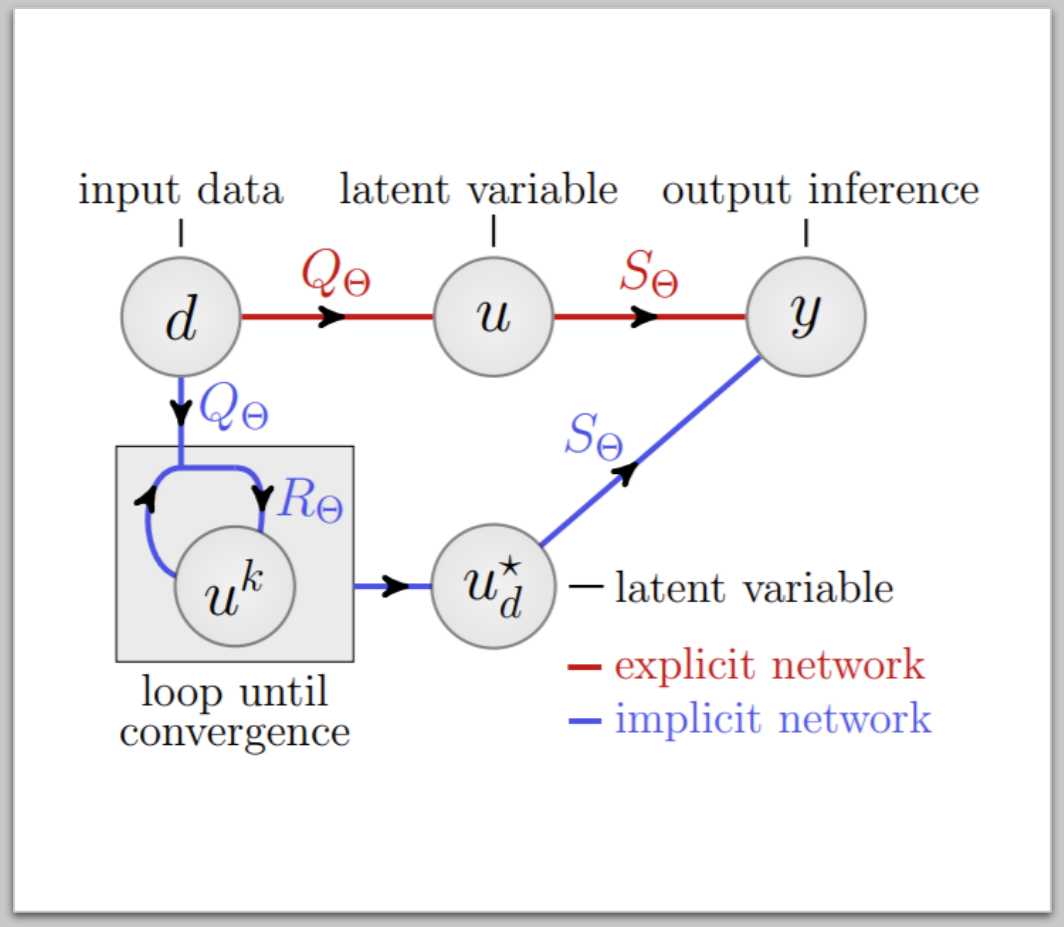
$$u = Q_\Theta(d)$$

solve $u^\star = R_\Theta(u^\star; u)$

$$y = S_\Theta(u^\star)$$

(Can replace $u^\star$ by an approximate $\tilde{u}$)

(Heaton et al'21, Gilton et al'21)



# Back propagation

- Compute gradients w.r.t. parameters $\Theta$
- Define $T_\Theta(u;d) \triangleq R_\Theta(u, Q_\Theta(d))$ and $\tilde{u}_d = T_\Theta(\tilde{u}_d; d)$

# Back propagation

- Compute gradients w.r.t. parameters $\Theta$
- Define $T_\Theta(u;d) \triangleq R_\Theta(u, Q_\Theta(d))$ and $\tilde{u}_d = T_\Theta(\tilde{u}_d; d)$
- $\mathcal{J}_\Theta(u;d) \triangleq \mathrm{I} - \dfrac{\mathrm{d}T_\Theta}{\mathrm{d}u}(u;d)$ exists a.e. if $T_\Theta$ is Lipschitz and $T_\Theta$ is a contraction for each $d$

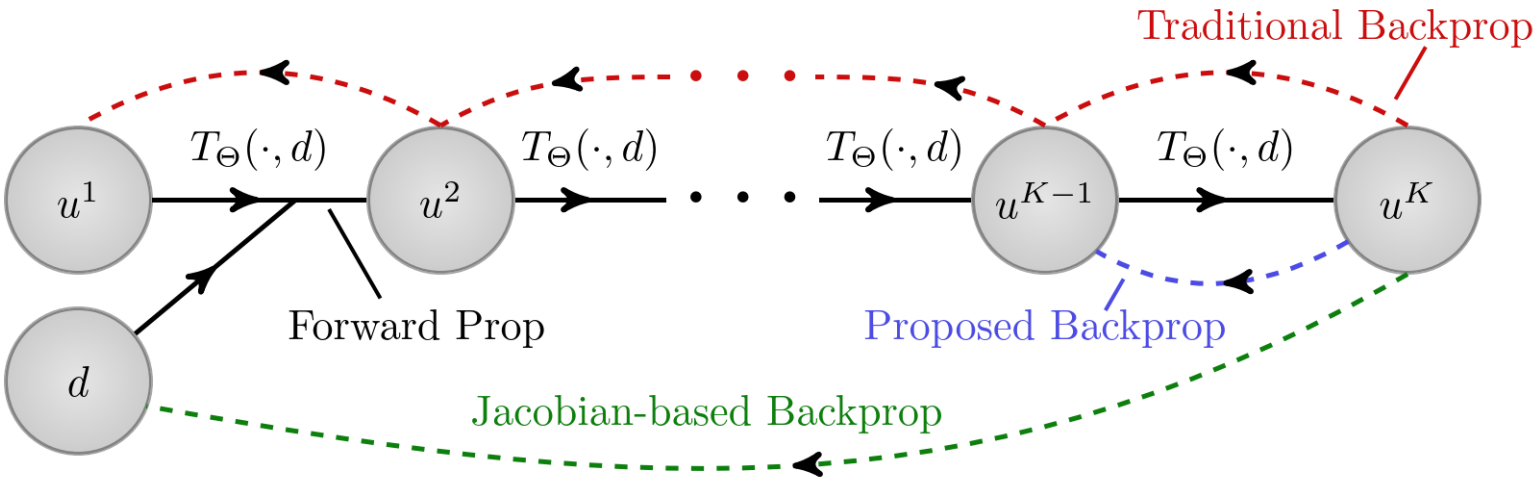
# Back propagation

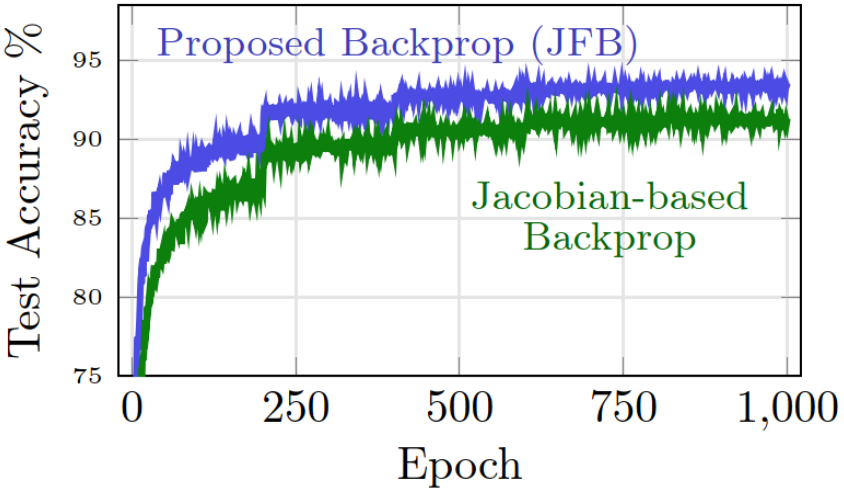
- Compute gradients w.r.t. parameters $\Theta$
- Define $T_\Theta(u;d) \triangleq R_\Theta(u, Q_\Theta(d))$ and $\tilde{u}_d = T_\Theta(\tilde{u}_d; d)$
- $\mathcal{J}_\Theta(u;d) \triangleq \mathrm{I} - \dfrac{\mathrm{d}T_\Theta}{\mathrm{d}u}(u;d)$ exists a.e. if $T_\Theta$ is Lipschitz and $T_\Theta$ is a contraction for each $d$
- Despite of infinite depth, backprop through $T_\Theta$ has finite computation and storage!

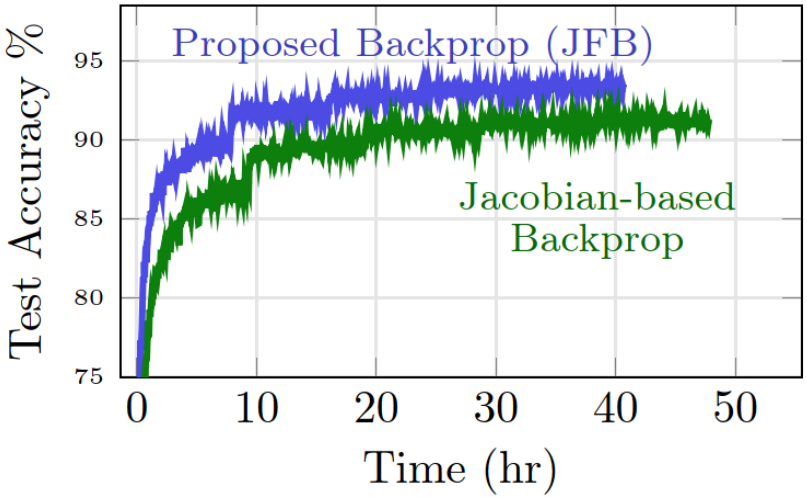
$$\frac{\mathrm{d}\tilde{u}_d}{\mathrm{d}\Theta} = \frac{\partial T_\Theta}{\partial u}\frac{\mathrm{d}\tilde{u}_d}{\mathrm{d}\Theta} + \frac{\partial T_\Theta}{\partial \Theta} \quad \Longrightarrow \quad \frac{\mathrm{d}\tilde{u}_d}{\mathrm{d}\Theta} = \mathcal{J}_\Theta^{-1} \cdot \frac{\partial T_\Theta}{\partial \Theta}$$

# Jacobian-free back propagation

- (Heaton et al'21) while forward propagation uses a fixed point, it suffices to use $T_\Theta$ during back prop. No Jacobian, no matrix inverse!
- Significance: huge speedup, allows complicated $T_\Theta$ (e.g., from operator splitting)
- Has a proof under Lipschitz and contraction conditions


Traditional Backprop
$T_\Theta(\cdot, d)$
$T_\Theta(\cdot, d)$
$T_\Theta(\cdot, d)$
$T_\Theta(\cdot, d)$
$u^1$
$u^2$
$u^{K-1}$
$u^K$
$d$
Forward Prop
Proposed Backprop
Jacobian-based Backprop


Proposed Backprop (JFB)
Jacobian-based Backprop
Test Accuracy %
95
90
85
80
75
0
250
500
750
1,000
Epoch


Proposed Backprop (JFB)
Jacobian-based Backprop
Test Accuracy %
95
90
85
80
75
0
10
20
30
40
50
Time (hr)

**MNIST**

| Method | Model size | Acc. |
|---|---|---|
| Explicit | 35K | 99.3% |
| Neural ODE† | 84K | 96.4% |
| Aug. Neural ODE† | 84K | 98.2% |
| MON ‡ | 84K | 99.2% |
| **FPN** | 35K | **99.4%** |

**SVHN**

| Method | Model size | Acc. |
|---|---|---|
| Explicit (ResNet) | 164K | 93.7% |
| Neural ODE† | 172K | 81.0% |
| Aug. Neural ODE† | 172K | 83.5% |
| MON (Multi-tier lg)‡ | 170K | 92.3% |
| **FPN (ours)** | 164K | **94.1%** |

**CIFAR-10**

| Method | Model size | Acc. |
|---|---|---|
| Explicit | 164K | 80.0% |
| Neural ODE† | 172K | 53.7% |
| Aug. Neural ODE† | 172K | 60.6% |
| MON (Single conv)‡ | 172K | 74.1% |
| **FPN (ours)** | 164K | **80.5%** |
| Explicit (ResNet-56)* | 0.85M | 93.0% |
| MON (Multi-tier lg)‡* | 1.01M | 89.7% |
| **FPN (ours)*** | 0.84M | **93.7%** |

# 4.
# Application
# Learn to Predict a Game

# Contextual Game

- $d$ represents the game contextual information, known to all the players
- We wish to predict game outcomes knowing only $d$
- Also train a player to play the game competitively

# Nash equilibrium

- $K$ self-interested players
- player $k$ chooses to do $x_k$, receives $u_k(x_k, x_{-k}, d)$

# Nash equilibrium

- $K$ self-interested players
- player $k$ chooses to do $x_k$, receives $u_k(x_k, x_{-k}, d)$
- NE is $(x_1, \ldots, x_K)$ if no player can improve their payoff by unilaterally deviating

# Nash equilibrium

- $K$ self-interested players
- player $k$ chooses to do $x_k$, receives $u_k(x_k, x_{-k}, d)$
- NE is $(x_1, \ldots, x_K)$ if no player can improve their payoff by unilaterally deviating
- Define game gradient $F \triangleq \left[\nabla_{x_1} u_1^\top \cdots \nabla_{x_K} u_K^\top\right]^\top$
- Define action set $\mathcal{C}$ or $\mathcal{C} = \mathcal{C}^1 \cap \mathcal{C}^2$

# NE as a Fixed Point

- Using operator splitting, an NE $x^\star$ satisfies

$$x_d^\star = P_{\mathcal{C}}\left(x_d^\star - F(x_d^\star; d)\right)$$

or

$$x_d^\star = P_{\mathcal{C}^1}(z_d^\star) \text{ where } z_d^\star = T(z_d^\star; d) \text{ and}$$
$$T(x; d) \triangleq x - P_{\mathcal{C}^1}(x) + P_{\mathcal{C}^2}\left(2P_{\mathcal{C}^1}(x) - x - F(P_{\mathcal{C}^1}(x); d)\right)$$

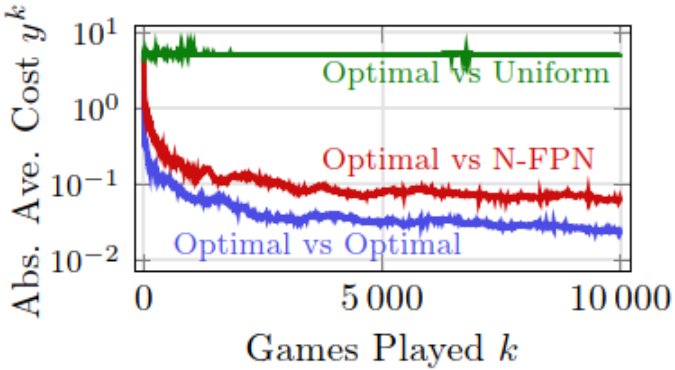
(Davis-Yin splitting)

# Observe many $d$, learn $F_\Theta$, thus $x^\star$

Rock, paper, scissors

- $d$ is payoff matrix
- $F_\Theta$: 2-layer N-FPN, 500 parameters
- Train $F_\Theta$
- Let one player (use learned $F_\Theta$ to take actions) to play with another with true $F_\Theta$

| | R | P | S |
|---|---|---|---|
| R | $0$ | $-\langle w^1, d\rangle$ | $\langle w^2, d\rangle$ |
| P | $\langle w^1, d\rangle$ | $0$ | $-\langle w^3, d\rangle$ |
| S | $-\langle w^2, d\rangle$ | $\langle w^3, d\rangle$ | $0$ |



# Contextual traffic routing

- $d$ represents weather, roadwork, etc., affects speed
- Each driver is selfish, minimizing commute time
- NE can be analytically computed
- Instead, we train a 3 layer fully connected N-FPN to predict NE given the road network, drivers, $d$
- $C$ = (network constraints) ∩ (nonnegativity)
- Compare to prediction to analytic solution

# Real city network tests

$$\mathsf{TRAFIX}(x_d^\circ, x_d^\star) \triangleq \frac{\#\left\{e \in E : |x_{d,e}^\circ - x_{d,e}^\star| < \varepsilon |x_{d,e}^\star|\right\}}{|E|}$$

| dataset | edges/nodes | OD-pairs | TRAFIX score |
|---|---|---|---|
| Sioux Falls | 76/24 | 528 | 0.94 |
| Eastern Mass. | 258/74 | 1113 | 0.97 |
| Berlin-Friedrichshain | 523/224 | 506 | 0.97 |
| Berlin-Tiergarten | 766/361 | 644 | 0.95 |
| Anaheim | 914/416 | 1406 | 0.95 |

# Summary

- Learning to optimize a new paradigm of optimization
- Useful at
  - Use data to improve modeling and method
  - Use data to find an optimization short cut
- Fixed-point network
  - Yields a consistent improvement in performance over finite-depth networks
  - Is surprisingly easy to train, end-to-end

# Co-authors, paper, and code

- Learning to Optimize: A Primer and A Benchmark. arXiv:2103.12828 by Tianlong Chen, Xiaohan Chen, Wuyang Chen, Zhangyang Wang (UT Austin), Jialin Liu (Alibaba US), Howard Heaton (UCLA)
- Fixed-point network: Howard Heaton (UCLA), Qiuwei Li (UCLA), Daniel McKenzie (UCLA), Samy Wu Fung (now at Colorado Sch of Mines)
- Codes
  - https://github.com/VITA-Group/Open-L2O
  - https://github.com/howardheaton/fixed_point_networks
  - https://github.com/howardheaton/nash_fixed_point_networks

# Thanks!

**Any questions?**