Learning to Optimize

Wotao Yin

Alibaba (US) Damo Academy

2022 CVPR Tutorial

1.
Background

Machine learning and Optimization

Answers are given as existing data

ML learns to give answers in the future

Induction

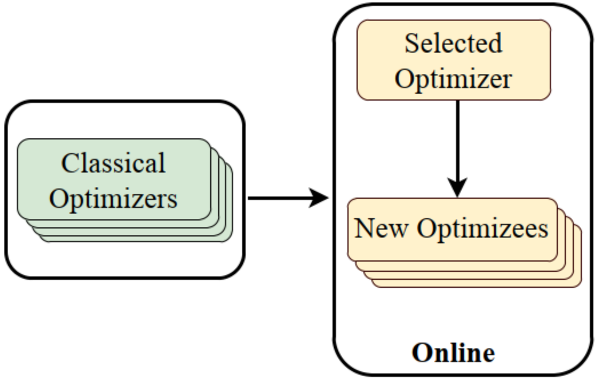
No answer is given; but we can evaluate answers.

OPT finds answers with best evaluations

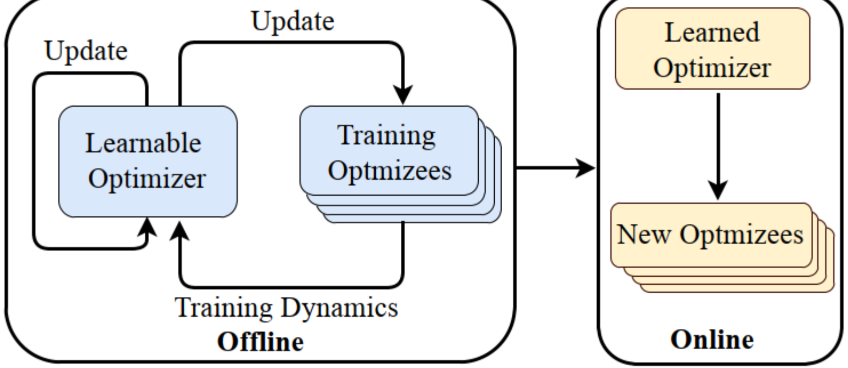
Prescription

L2O aims to “optimize faster” or “construct a better optimization model”, or both, in the future.

Classic optimization



Learning-to-optimize



model-based vs model-free

- g has a form of an existing method, called model.
- L2O searches for the **best values of some parameters**

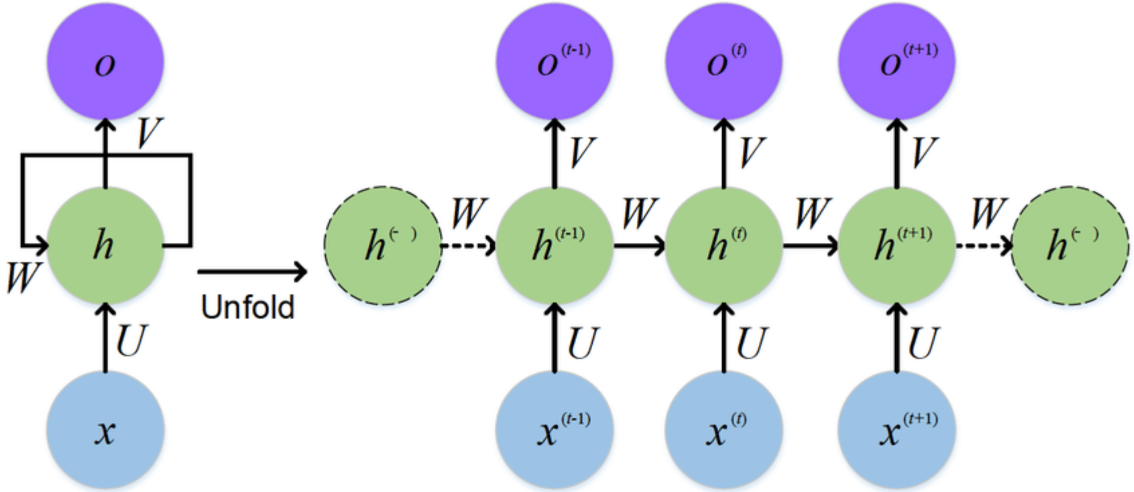
- g is based on universal approximators, e.g., recurrent neural networks.
- L2O is set to discover **new update rules** without referring to any existing updates (other than being iterative)

2.

Model-Free L2O

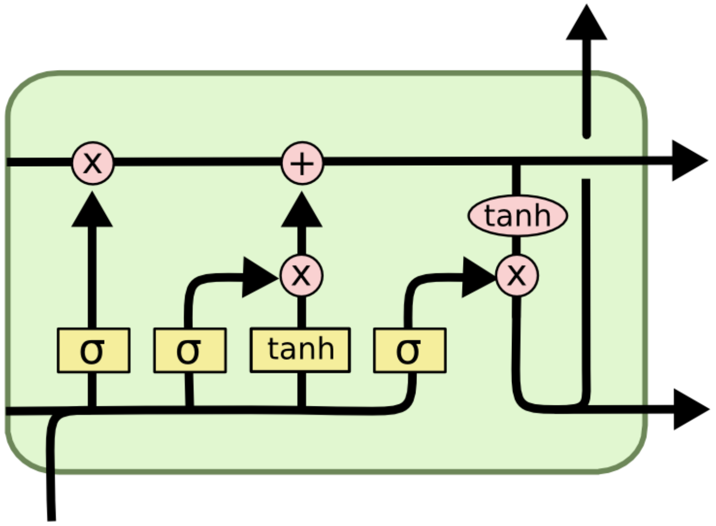
based on RNNs, especially LSTMs

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

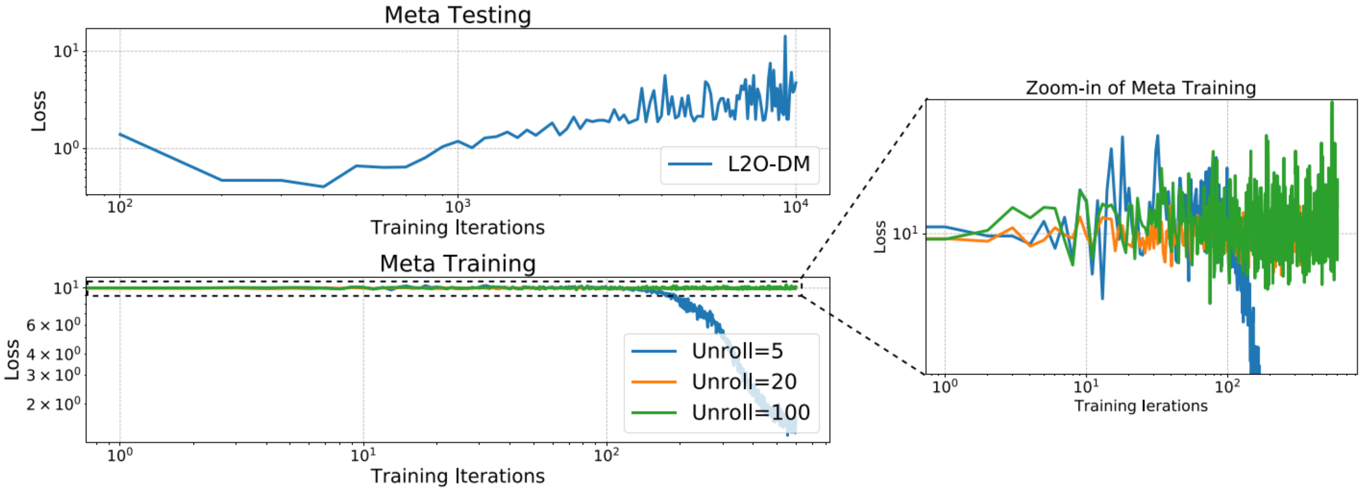


Andrychowicz et al'NIPS16; Chen et al'ICML17; Lv-Jiang-Li'17; Cao et al'NeurIPS19; Xiong-Hsieh'20

Optimizer Architecture	Input Feature	Meta Training Objective	Additional Technique	Evaluation Metric
LSTM	Gradient	Meta Loss	Transform input gradient ∇ into $\log(\nabla)$ and $\mathrm{sign}(\nabla)$	Training Loss
LSTM	Objective Value	Objective Value	N/A	Objective Value
LSTM	Gradient	Meta Loss	Random Scaling Combination with Convex Functions	Training Loss
Hierarchical RNNs	Scaled averaged gradients, relative log gradient magnitudes, relative log learning rate	Log Meta Loss	Gradient History Attention Nesterov Momentum	Training Loss
MLP	Gradient	Meta Loss	Unbiased Gradient Estimators	Training Loss 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, particle's velocity and attraction	Meta Loss and Entropy Regularizer	Sample- and Feature- Attention	Training Loss
RNN	Input Images, Input Gradient	Meta Loss	N/A	Standard and Robust Test Accuracies
LSTM	Input Gradient	Meta Loss	N/A	Training Loss and Robust Test Accuracy

Challenges: network depth

- Shallow networks may not reach the accuracy
- Deep networks have high training and memory costs



Other techniques

- Attention mechanisms and pointer networks
- Graph neural networks (GNNs)

3.

Model-Based L2O

3.1
Unrolling

Unrolling by example: LASSO

LASSO model:

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

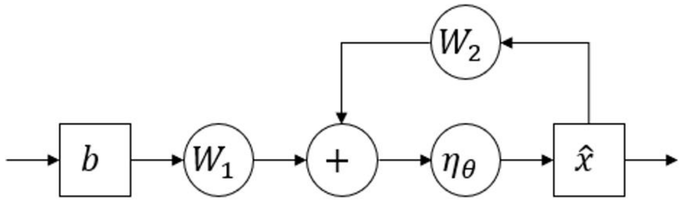
Iterative Shrinkage and Thresholding Algorithm (**ISTA**):

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

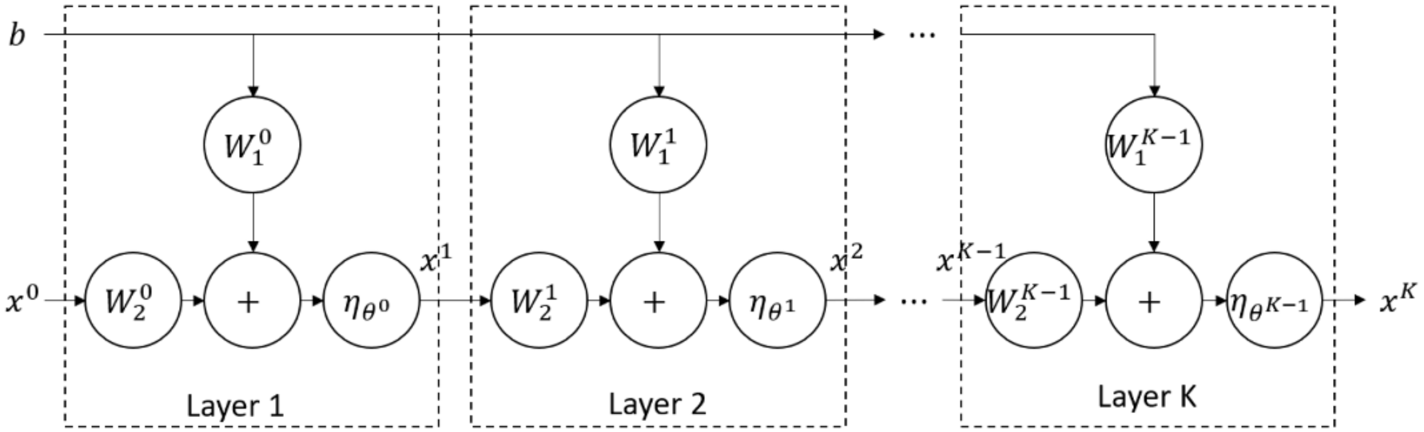
Rewrite ISTA as

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

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).
- ▷ Lack guarantees: when and what if L2O fails?
- ▷ **Open: solve more types of problems and add safeguards**

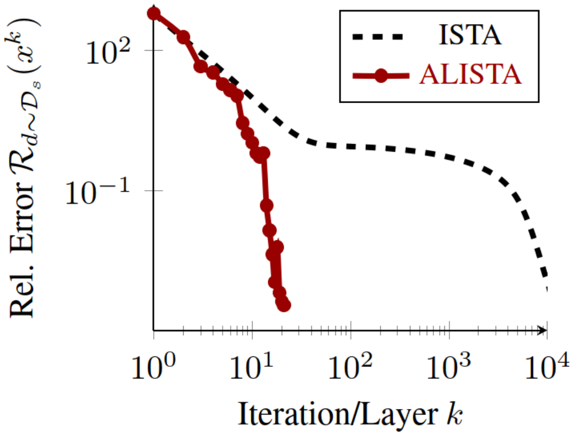
Safe-guarding

(Heaton et al.20) L2O convergence can be ensured by incorporating an “energy” *E*

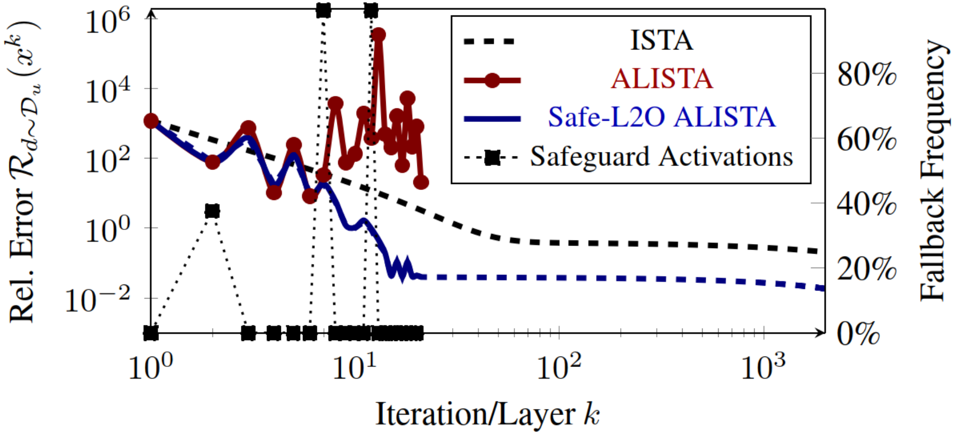
$$x^{k+1} = \begin{cases} \text{L2O update } z^k & \text{if } E^k(z^k) \leq E^k(x^k) \\ \text{classic update } T(x^k) & \text{otherwise} \end{cases}$$

When L2O fails to decrease the energy or exceeds *K*, the classic update T kicks in.

Apply learned ALISTA to problems seen and problems unseen



(a) Performance on seen distribution, i.e., $d \sim \mathcal{D}_s$



(b) Performance on *unseen* distribution, i.e., $d \sim \mathcal{D}_u$

3.2

Deep Equilibrium, or Fixed-Point Network

Deep Equilibrium (Fixed Point Network)

- Related to (Chen et al’NeurIPS18, Dupont et al’NeurIPS19) Neural ODE, based on black-box ODE solver for initial value problem
- (Bai et al’NeurIPS19, Winston&Kolter’NeurIPS20) Instead of finite iterations, use infinite iterations (in theory) to output a fixed point
- We can modify (i.e., train) the iterator or ODE model in an end-to-end manner

Example

Explicit network

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

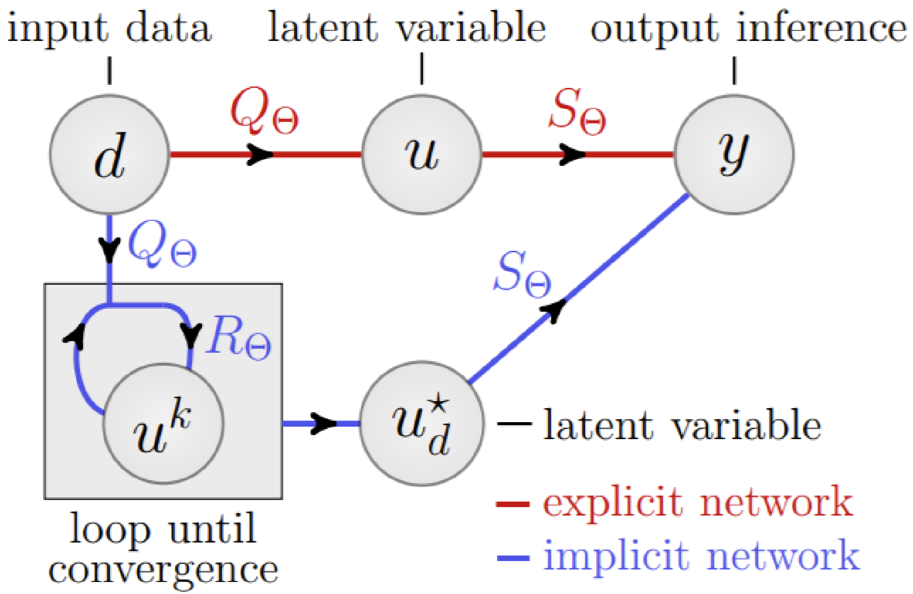
versus

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)

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



Back propagation

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

Back propagation

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

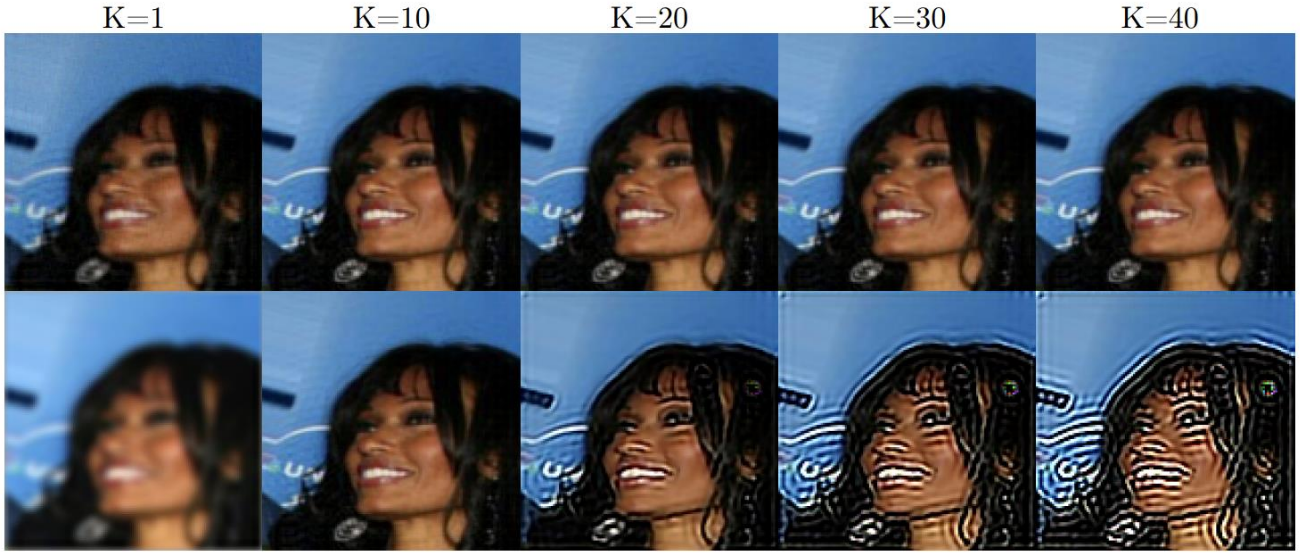
Back propagation

- Compute gradients w.r.t. parameters Θ
- Define $T_\Theta(u;d) \triangleq R_\Theta(u, Q_\Theta(d))$ $\tilde{u}_d = T_\Theta(\tilde{u}_d; d)$
- $\mathcal{J}_\Theta(u;d) \triangleq \mathrm{I} - \frac{\mathrm{d}T_\Theta}{\mathrm{d}u}(u;d)$ exists a.e. if T_Θ is Lipschitz and T_Θ is a contraction for each d
- Backprop through fixed-point mapping uses finite computation and storage

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

Fixed-Point Network

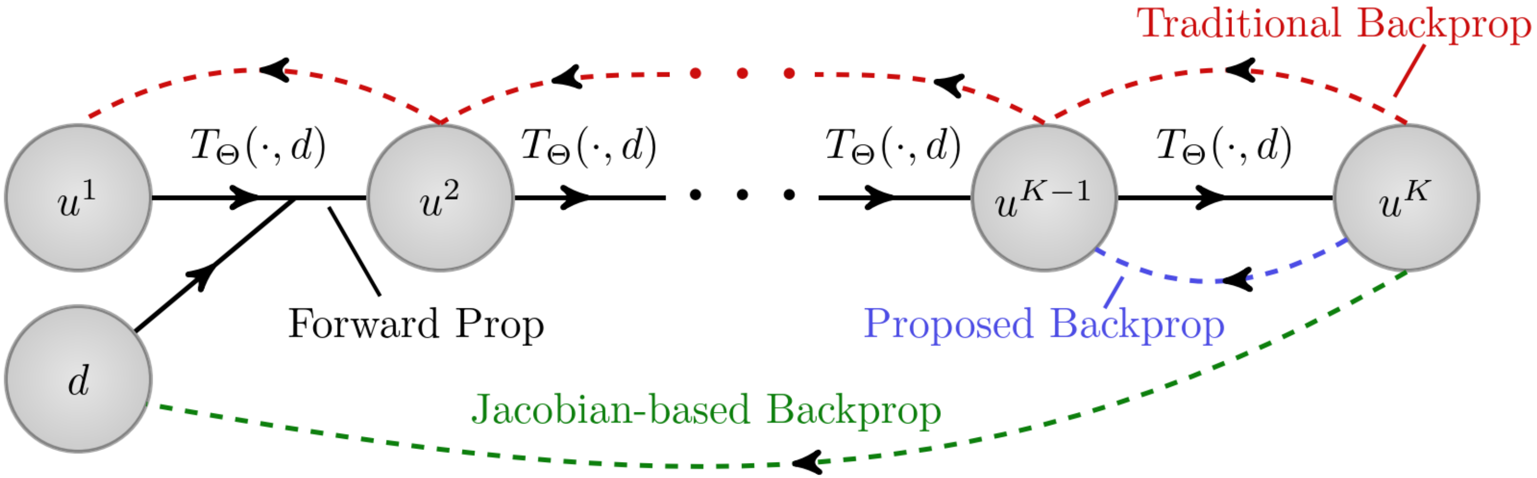
Unrolled Prox-Grad



Gilten, Ongie, and Willett'21

Jacobian-free back propagation

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



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

	Jacobian Free	Jacobian Based
MNIST	99.4% (21.3 s/epoch)	94.6 % (32.4 s/epoch)
SVHN	94.1% (37 s/epoch)	84.6% (65.1 s/epoch)
CIFAR10	93.7% (147 s/epoch)	unfinished

Issues that affect only Jacobian based methods:

- Batch normalization not applicable
- ReLU nonlinearity
- Non-convergence with CG and quasi-Newton solvers

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%**

Future directions

- Joint learning of network and optimizer
$$u_\Theta = T_{\Theta_1}(u_\Theta, Q_{\Theta_2}(x))$$
- Nonconvex optimization model
- Optimization as a middle layer, opposed to a final layer
- Need more real applications

3.3

Forward-by-Solver Backward-by-KKT

Predict then Optimize

- $Q_{\Theta_2}(x)$ predicts the input to optimization
- Then, $u_\Theta = T_{\Theta_1}(Q_{\Theta_2}(x))$ is the final prediction where T_{Θ_1} is a solver
- Often a solver is a blackbox, not differentiable
- So, for backpropagation, we use KKT condition

Challenges

- KKT conditions are nondifferentiable
- Solutions:
  - add $||\cdot||^2$ to objective
  - Log barrier
  - Gaussian smoothing
- Differentiable KKT conditions perform poorly (e.g., OptNet examples are small)

3.4
Plug-and-Play

Plug-and-Play: background

- Consider

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

- ADMM:

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

- For image recovery:
  - 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:

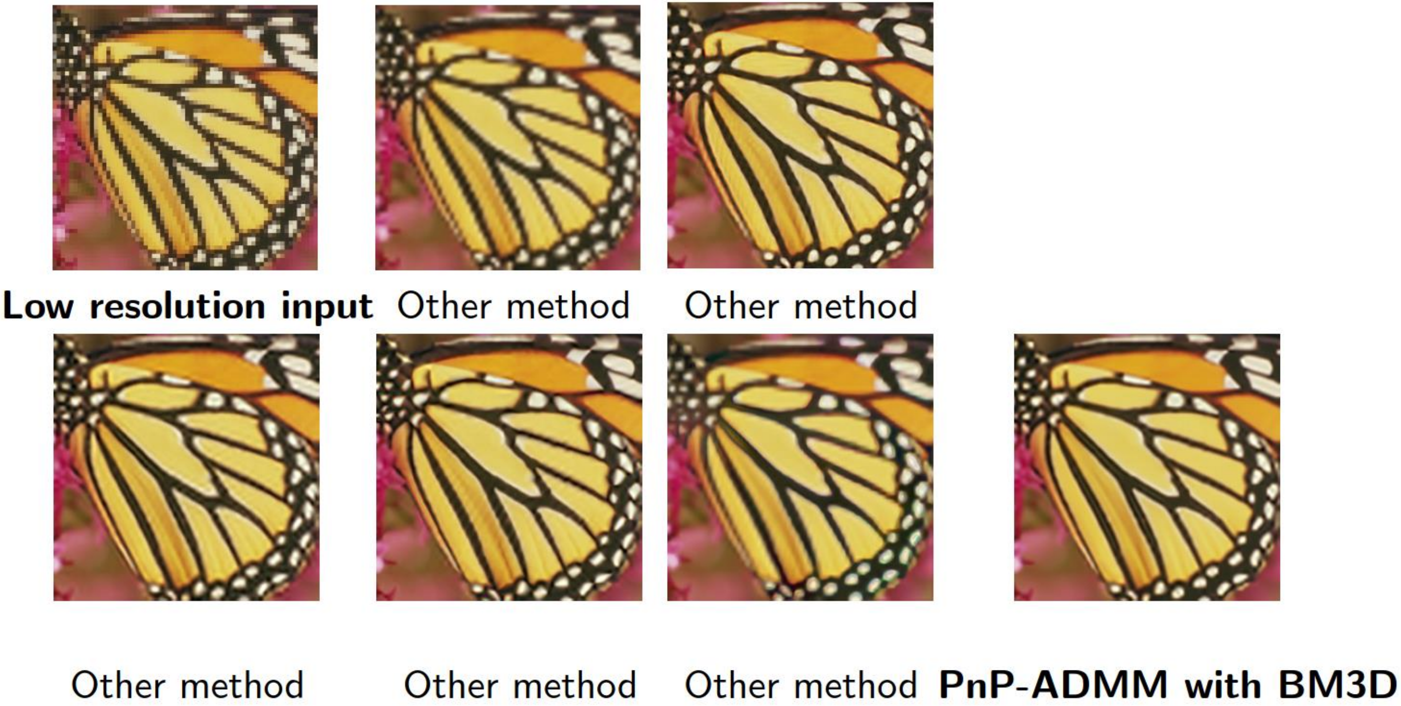
$$x^{k+1} = \text{Prox}_{\sigma^2 g}(y^k - u^k)$$
$$y^{k+1} = \text{Prox}_{\alpha f}(x^{k+1} + u^k)$$
$$u^{k+1} = u^k + x^{k+1} - y^{k+1}.$$

$$\Rightarrow$$

$$x^{k+1} = H_\sigma(y^k - u^k)$$
$$y^{k+1} = \text{Prox}_{\alpha f}(x^{k+1} + u^k)$$
$$u^{k+1} = u^k + x^{k+1} - y^{k+1}.$$

- Works surprisingly well!

Example: Super resolution



(Chen-Wang-Elgendy'17)

Strength:

- Good performance
- Can run for infinite iterations (Ryu et al'ICML19) if
  - $I - H_\sigma$ is Lipschitz, by spectrum normalization
  - f is strongly convex

Limitation:

- Denoise H_σ is pre-trained before plugged in (Training is not end-to-end)
- The good performance is difficult to explain

Other

- RED
- Neural network preconditioner

$$\min_{\Theta} ||y - Au(\Theta)||$$

3.5
Symobolic L2O with convergence guarantees

- Symbolic rules are easier to understand and verify, and generalize better
- Most works are still restricted to symbolic regressions (i.e., fitting formulas), but new papers appear every few days
- Open questions:
    - Better representations than a tree
    - Automatically get convergence proofs (e.g., use another net to model Lyapunov functions)

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 $(x_1, \dots, x_K)$ 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 $(x_1, \dots, x_K)$ if no player can improve their payoff by unilaterally c

$$F \triangleq \left[\nabla_{x_1} u_1^\top \cdots \nabla_{x_K} u_K^\top\right]^\top$$

- Define game gradient
- 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

$$x_d^\star = P_{\mathcal{C}^1}(z_d^\star) \text{ where } z_d^\star = T(z_d^\star; d) \text{ and}$$

$$T(x; d) \triangleq x - P_{\mathcal{C}^1}(x) + P_{\mathcal{C}^2}\left(2P_{\mathcal{C}^1}(x) - x - F(P_{\mathcal{C}^1}(x); d)\right)$$

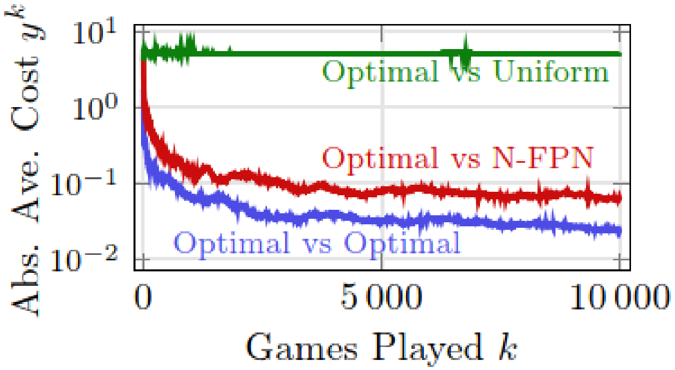
(Davis-Yin splitting)

Observe many d, learn F_Θ, thus $x^\star$

Rock, paper, scissors

- ▷ d decides a payoff matrix
- ▷ F_Θ: 2-layer N-FPN, 500 parameters
- ▷ Train F_Θ
- ▷ Let one player (use learned F_Θ to take actions) to play with another with true F_Θ

	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) $\cap$ (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

Thanks!

Any questions?

5.
Uncovered topics

- Meta-learning (learning to learn)
- Reinforcement learning based approaches
- Progress of solving MIPs, SATs, etc.
- Unrolling second-order (e.g., quasi-Newton) methods
- Use a classic solver, possibly with parameters, as a layer in a large network
- ......

6.

Open questions

- Lack of training data due to privacy or proprietary protections
- Unbalanced training data
- Un-safeguarded L2O methods may fail, unsuitable for critical scenarios
- Difficult to interpret, cannot do “what-if” analysis

Credits

Special thanks to all the students / collaborators who worked with us and released the code:

- Tianlong Chen, Xiaohan Chen, Wuyang Chen
- Howard Heaton
- Jialin Liu