Explainable Models via Data-Driven Optimization

Howard Heaton

Typal Academy

Overview

Setting

Solve problems where approximate optimization models can be hand-crafted

Learning to Optimize (L2O)

Make parameterized optimization model and use training data to tune it, *i.e.*

$$\text{(model output)} \triangleq \text{argmin (prior knowledge)} + \text{(data-driven terms)}$$

Goal for Today

Outline the tools needed for audience to create their own L2O models

Historic Comparison of Two Approaches

Machine Learning

$$\mathcal{N}_\Theta(d) = \sigma(W^m \cdot + b^m) \circ \cdots \circ \sigma(W^1 d + b^1)$$

- ✔ adapt to available data
- ✖ satisfy constraints / optimality
- ✔ expressive capacity
- ✔ flexible architectures

Traditional Optimization

$$\operatorname*{argmin}_{x \in \mathcal{C}} f(x)$$

- ✖ adapt to available data
- ✔ guaranteed optimality
- ✔ interpretable models
- ✔ scalable first-order algorithms

Learning fast approximations of sparse coding by Gregor and LeCun in 2010

Sparse codes can be modeled as solutions to the ℓ_1 regularized problem

$$\min_x \frac{1}{2}\|Ax - d\|^2 + \lambda\|x\|_1$$

Proximal gradient updates look like

$$x^{k+1} = \text{shrink}\left(x^k - \alpha A^\top(Ax^k - d),\ \alpha\lambda\right) = \underbrace{\text{shrink}\left(W_1 x^k + W_2 d,\ \alpha\lambda\right)}_{\text{feed forward layer}},$$

where $W_1 = I - \alpha A^\top A$, $W_2 = \alpha A^\top$, and $\text{shrink}(z,\ \beta) = \text{sign}(z) \cdot \max(|z| - \beta, 0)$

LISTA Idea: Treat x^K as network output and tune W_1 and W_2 from training data

Key L2O Steps

1. Make a network by paramaterizing an optimization problem to get

$$N_\Theta(d) \triangleq \underset{x}{\text{argmin}} \; f_\Theta(x;\, d)$$

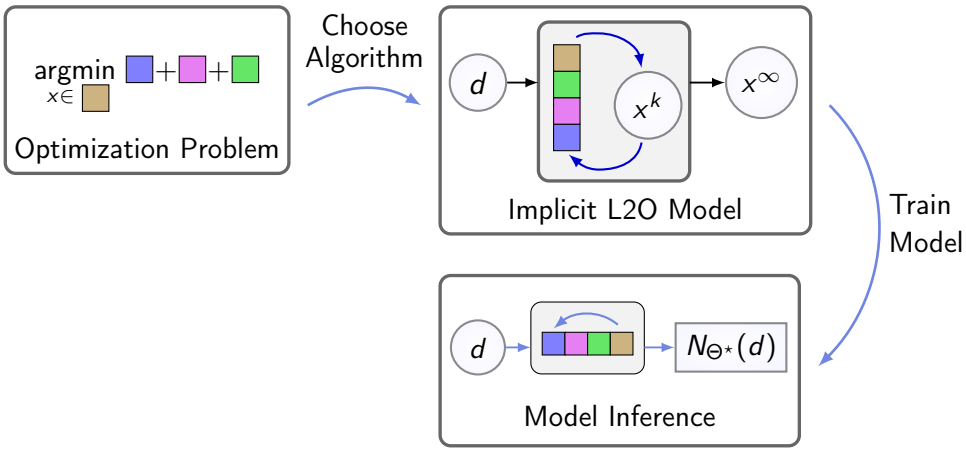
Note: Constraints can be included in this formulation

Note: More general L2O models can use black box and optimization layers

2. Forward prop consists of applying an apt first-order algorithm *until convergence*

3. Backprop consists of using built-in autograd *on last step of forward prop*

L2O via Building Blocks



Task

Recover a signal $x_d^\star$ from linear measurements $d = Ax_d^\star$

Key Knowledge

Signal $x_d^\star$ has low dimensional structure (but is *not* sparse)

L2O Model

For a "sparisfying matrix" K , we can estimate

$$x_d^\star \approx \underset{x}{\operatorname{argmin}} \|Kx\|_1 \text{ s.t. } Ax = d$$

Question: Even if this model is "right" for some choice of K, how do we find K?

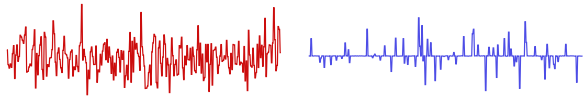


Figure 1: Applying the learned K sparsifies $x_d^\star$ (shown for test data d)

Fix weights $\Theta = K \in \mathbb{R}^{250\times 250}$, noting $x_d^\star \in \mathbb{R}^{250}$ and $d \in \mathbb{R}^{100}$, and let model $\mathcal{N}_\Theta$ be

$$\mathcal{N}_\Theta(d) \triangleq \underset{x}{\operatorname{argmin}} \|Kx\|_1 \text{ s.t. } Ax = d$$

For a distribution of measurement/signal pairs $(d, x_d^\star)$, train model by minimizing

$$\min_\Theta \mathbb{E}_d \left[\|x_d^\star - \mathcal{N}_\Theta(d)\|^2\right]$$

This ensures the weights Θ are tuned as well as possible for the task at hand

Toy Example – Results

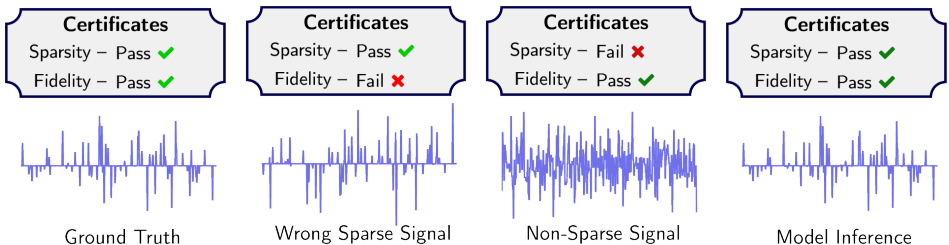


Figure 2: Example inferences for test data d with sparsified Kx of each inference x shown

```
x_fxd_pt = find_fixed_point(d)

pred = apply_opt_update(x_fxd_pt, d)

loss = criterion(pred, labels)

loss.backward()

optimizer.step()
```

Figure 3: Sample PyTorch code for backpropagation. Here `x_fxd_pt` is the *same* as `pred`, except that `pred` has gradients attached and `x_fxd_pt` does not

Informal Theorem:[1] Backpropping through the final step of a fixed point optimization algorithm (as shown above) yields a *preconditioned* gradient

[1]Wu Fung, *et al. JFB: Jacobian-Free Backpropagation for Implicit Networks*. 2022.

Prior Knowledge Embedding

Can use regularizer to promote sparsity, low-rank, smooth, laying on manifold, ...

Hard Constraints

Constraints can be put into model (*e.g.* simplex, linear system) and forward propagation can continue until these are met to given tolerance

L2O model can inherit all guarantees desired from traditional optimization

Outline

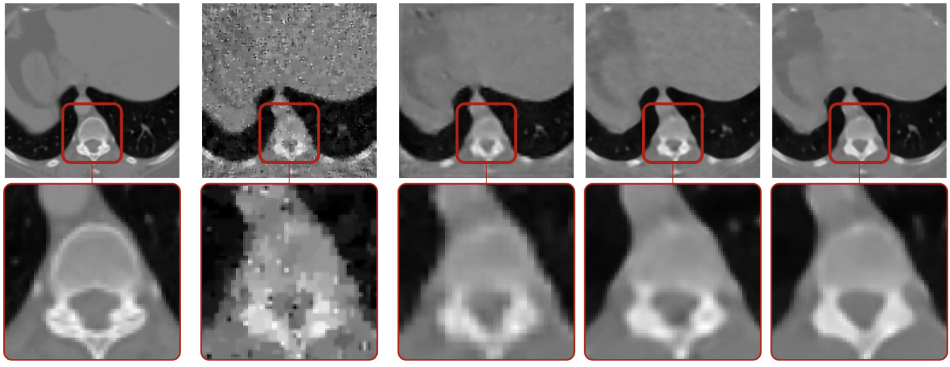


Figure 4: Comparison of techniques, ranging from traditional to fully data-driven

$$(\text{L2O Model}) = \mathcal{N}_\Theta(d) \triangleq \underset{x\in[0,1]^n}{\operatorname{argmin}} f_\Theta(Kx) \ \text{ s.t. } \|Ax - d\| \leq \delta$$

Traffic Routing

Task

Given contextual data *d* (*e.g.* weather, construction), predict traffic distribution

L2O Model

Assume drivers are self-interested and most likely behavior is a Nash equilibria,

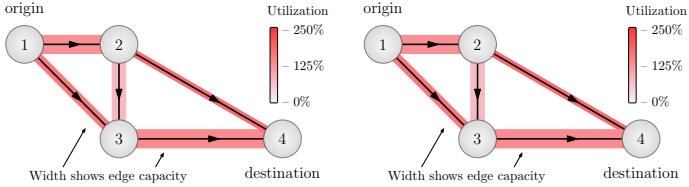
where game gradient is *unknown*

Constraints

Vertex-arc incidence matrix characterizes road network,

yielding large linear constraints to get valid traffic flows

Traffic Routing – Toy Example



(a) Rainy Day Prediction

(b) Rain Day True

(c) Sunny Day Prediction

(d) Sunny Day True

Figure 5: Predictions of toy traffic from context data *d*

Relevant Works

- Heaton, Wu Fung. *Explainable AI via Learning to Optimize*. 2022.
- McKenzie, Wu Fung, Heaton. *Faster Predict-and-Optimize with Three-Operator Splitting*. 2023.
- Wu Fung, *et al. JFB: Jacobian-free Backpropagation for Implicit Networks.* 2022
- Heaton, Wu Fung, Gibali. *Feasibiliy-based Fixed Point Networks*. 2021.
- Heaton, *et al. Learn to Predict Equilibria via Fixed Point Networks.* 2021.
- Gilton, Ongie, Willet. *Deep Equilibrium Architectures for Inverse Problems in Imaging*. 2022.
- Heaton, *et al.* Safeguarded Learned Convex Optimization. 2023.
- de Avila Belbute-Peres, *et al. End-to-End Differentiable Physics for Learning and Control.* 2018.

Summary

Setting

Solve problems where approximate optimization models can be hand-crafted

Learning to Optimize (L2O)

- Make parameterized optimization model

$$\text{(model output)} \triangleq \operatorname{argmin} \text{(prior knowledge)} + \text{(data-driven terms)}$$

- Training is “easy” using Jacobian-Free Backpropagation (JFB)
- Outputs leverage available data, have strong guarantees, and are interpretable

Outline

The toy problem can be generalized to a model of the form

$$\min_x f(Kx) + h(x) \text{ s.t. } \|Mx - d\| \leq \delta,$$

with proximable f and g, can be rewritten as

$$\min_{x,w,p} f(p) + h(x) + \delta_{B(d,\delta)}(w) \text{ s.t. } \begin{bmatrix} K \\ M \end{bmatrix} x - \begin{bmatrix} p \\ w \end{bmatrix} = 0,$$

where p and w are auxiliary variables and $\delta_{B(d,\infty)}$ is the indicator function that is 0 inside the Euclidean ball $B(d, \delta)$ and ∞ elsewhere

Using Linearized ADMM updates yields the iterative updates

$$
\begin{aligned}
p^{k+1} &= \text{prox}_{\lambda f}\left(p^k + \lambda(\nu_1^k + \alpha(Kx^k - p^k))\right) \\
w^{k+1} &= \text{proj}_{B(d,\delta)}\left(w^k + \lambda(\nu_2^k + \alpha(Mx^k - w^k))\right) \\
\nu_1^{k+1} &= \nu_1^k + \alpha(Kx^k - p^{k+1}) \\
\nu_2^{k+1} &= \nu_2^k + \alpha(Mx^k - w^{k+1}) \\
r^k &= K^\top\left(2\nu_1^{k+1} - \nu_1^k\right) + M^\top\left(2\nu_2^{k+1} - \nu_2^k\right) \\
x^{k+1} &= \text{prox}_{\beta h}\left(x^k - \beta r^k\right),
\end{aligned}
$$

which can be coded into the `apply_opt_update` function