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.*

(model output) , argmin (prior knowledge) + (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

$$N(d) = (W^m + b^m) \quad (W^1 d + b^1)$$

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

Traditional Optimization

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

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

The Origin – LISTA

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

Sparse codes can be modeled as solutions to the `$_1$ regularized problem

$$\min_x \frac{1}{2} kAx \quad dk^2 + \quad kxk_1$$

Proximal gradient updates look like

$$x^{k+1} = \text{shrink} \quad x^k \qquad A^{>}(Ax^k \quad d); \qquad = \text{shrink} \quad W_1 x^k + W_2 d; \qquad ;$$

feed forward layer

where $W_1 = I \qquad A^{>}A$, $W_2 = \quad A^{>}$, and shrink$(z; \quad) = \text{sign}(z) \quad \max(jzj \qquad ; 0)$

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

$$N \ (d) \ , \ \underset{x}{\text{argmin}} \ f \ (x; \ d)$$

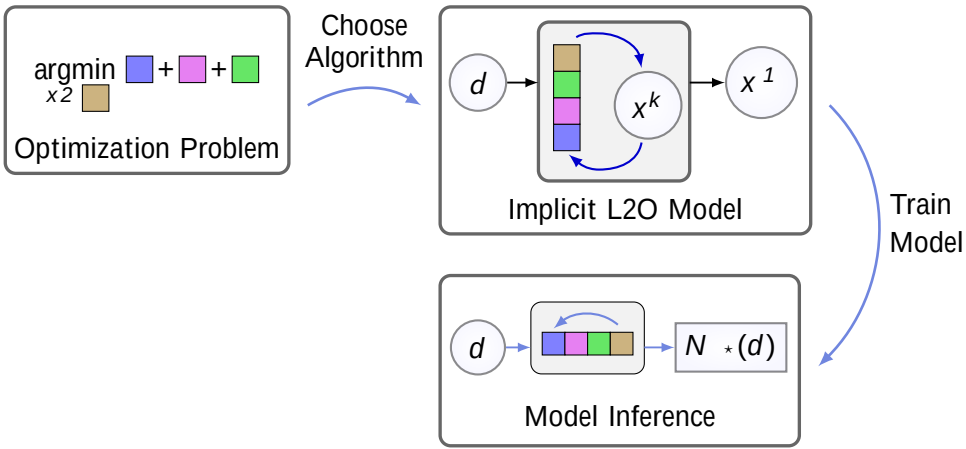
Note: Constraints can be included in this formulation

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



Toy Example – Setup

Task

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

Key Knowledge

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

L2O Model

For a "sparisfying matrix" K, we can estimate

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

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

Toy Example – Training

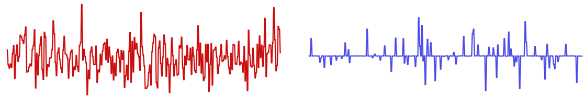


Figure 1: Applying the learned K sparsifies $x_d^\star$ (shown for test data d)

Fix weights $= K\ 2\ \mathbb{R}^{250\ \ 250}$, noting $x_d^?\ 2\ \mathbb{R}^{250}$ and $d\ 2\ \mathbb{R}^{100}$, and let model N be

$$N\ \ (d)\ ,\ \operatorname*{argmin}_x kKxk_1\ \text{s.t.}\ Ax = d$$

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

$$\min \mathbb{E}_d\ \ kx_d^?\ \ \ N\ \ (d)k^2$$

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

Toy Example – Results

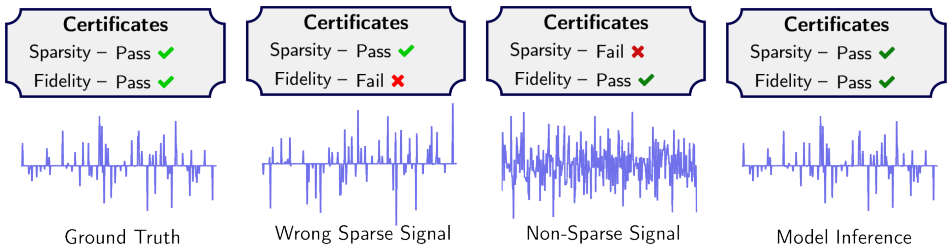


Figure 2: Example inferences for test data d with sparsified Kx of each inference x shown

```
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.

Model Output Guarantees

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

CT Image Reconstruction

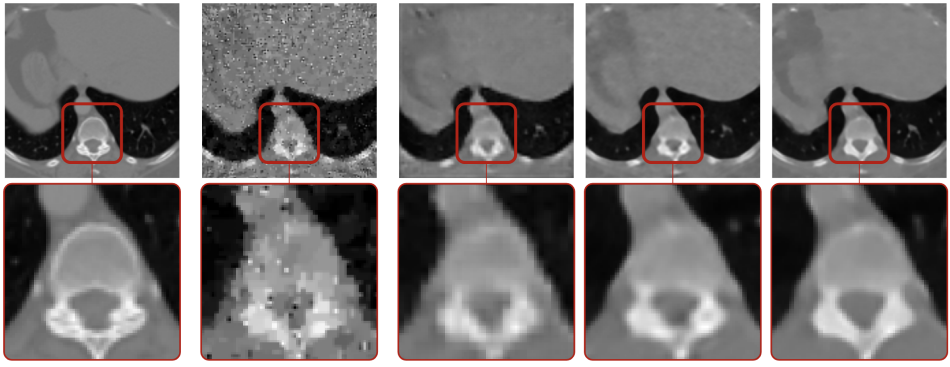


Figure 4: Comparison of techniques, ranging from traditional to fully data-driven

$$\text{(L2O Model)} = N\ (d)\ ,\ \underset{x\,2\,[0;1]^n}{\operatorname{argmin}} f\ (Kx)\ \text{s.t.}\ kAx\quad dk$$

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,

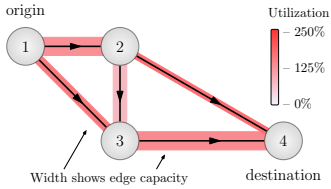
where game gradient is *unknown*

Constraints

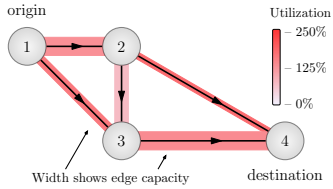
Vertex-arc incidence matrix characterizes road network,

yielding large linear constraints to get valid traffic flows

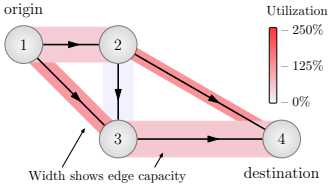
Traffic Routing – Toy Example



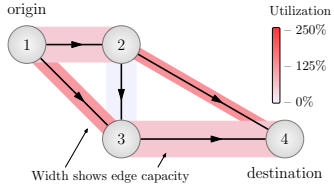
(a) Rainy Day Prediction



(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 Di erentiable 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

  (model output) , argmin (prior knowledge) + (data-driven terms)

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

Outline

Forward Prop for Toy Example

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

$$\min_{x} f(Kx) + h(x) \text{ s.t. } kMx \quad dk \quad ;$$

with proximable f and g, can be rewritten as

$$\min_{x;w;p} f(p) + h(x) + {}_{B(d;\ )}(w) \text{ s.t. } \begin{bmatrix} K \\ M \end{bmatrix} x \quad \begin{bmatrix} p \\ w \end{bmatrix} = 0;$$

where p and w are auxiliary variables and ${}_{B(d;\,1\,)}$ is the indicator function that is 0 inside the Euclidean ball $B(d;\ )$ and 1 elsewhere

Forward Prop for Toy Example

Using Linearized ADMM updates yields the iterative updates

$$
\begin{aligned}
p^{k+1} &= \text{prox}_{f}\left(p^{k} + \left(\lambda_1^{k} + (Kx^{k} - p^{k})\right)\right) \\
w^{k+1} &= \text{proj}_{B(d;\,)}\left(w^{k} + \left(\lambda_2^{k} + (Mx^{k} - w^{k})\right)\right) \\
\lambda_1^{k+1} &= \lambda_1^{k} + (Kx^{k} - p^{k+1}) \\
\lambda_2^{k+1} &= \lambda_2^{k} + (Mx^{k} - w^{k+1}) \\
r^{k} &= K^{\top}\left(2\lambda_1^{k+1} - \lambda_1^{k}\right) + M^{\top}\left(2\lambda_2^{k+1} - \lambda_2^{k}\right) \\
x^{k+1} &= \text{prox}_{h}\left(x^{k} - r^{k}\right);
\end{aligned}
$$

which can be coded into the `apply_opt_update` function