IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 27 October 2023, accepted 14 January 2024, date of publication 25 January 2024, date of current version 5 February 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3358724

RESEARCH ARTICLE

Teaching Networks to Solve Optimization Problems

XINRAN LIU[1], YUZHE LU[1], ALI ABBASI[1], MEIYI LI[2], (Graduate Student Member, IEEE), JAVAD MOHAMMADI[2], (Senior Member, IEEE), AND SOHEIL KOLOURI[1], (Senior Member, IEEE)
[1]Department of Computer Science, Vanderbilt University, Nashville, TN 37235, USA
[2]Department of Civil Architectural and Environmental Engineering, University of Texas at Austin, Austin, TX 78712, USA

Corresponding author: Soheil Kolouri (soheil.kolouri@vanderbilt.edu)

This work was supported in part by the Defense Advanced Research Projects Agency (DARPA) under Contract HR00112190135.

ABSTRACT Leveraging machine learning to facilitate the optimization process is an emerging field that holds the promise to bypass the fundamental computational bottleneck caused by classic iterative solvers in critical applications requiring near-real-time optimization. The majority of existing approaches focus on learning data-driven optimizers that lead to fewer iterations in solving an optimization. In this paper, we take a different approach and propose to replace the iterative solvers altogether with a trainable parametric set function, that outputs the optimal arguments/parameters of an optimization problem in a single feed forward. We denote our method as Learning to Optimize the Optimization Process ($\mathcal{LOOP}$). We show the feasibility of learning such parametric (set) functions to solve various classic optimization problems including linear/nonlinear regression, principal component analysis, transport-based coreset, and quadratic programming in supply management applications. In addition, we propose two alternative approaches for learning such parametric functions, with and without a solver in the $\mathcal{LOOP}$. Finally, through various numerical experiments, we show that the trained solvers could be orders of magnitude faster than the classic iterative solvers while providing near optimal solutions.

INDEX TERMS Machine learning for optimization, deep set learning.

I. INTRODUCTION

Optimization problems are ubiquitous in computational sciences and engineering. Classic solutions to optimization problems involve iterative algorithms often relying on predetermined first and second order methods like (sub)gradient ascent/descent, conjugate gradients, simplex basis update, among others. These methods often come with desirable theoretical convergence guarantees, but their iterative nature could be limiting in applications requiring near-real time inference. Moreover, these algorithms' performance remains the same regardless of the number of times a similar optimization problem is visited. Recently, there has been an emerging interest in leveraging machine learning to enhance the efficiency of optimization processes and address some of these shortcomings. The learning based solutions are often referred to as *Learning to Optimize* (L2O) methods in the literature.

While L2O methods do not come with theoretical guarantees, they hold the promise of: 1) reducing the number of iterations needed to arrive at a solution, and 2) improving over time as more optimization problems are visited. L2O allows for transferring recent advances in machine learning, e.g., self-supervised learning, meta-learning, and continual learning, to learn data-driven optimization algorithms that could improve over time. Most existing L2O methods aim to learn a function that receives the current loss or its gradient, and based on the memory of previous loss values (or gradients) provide an update for the optimization parameters. Hence, these methods do not eliminate the iterative nature of the

The associate editor coordinating the review of this manuscript and approving it for publication was Adnan Kavak.

solution but aim at improving the iterative solution to: 1) reduce the number of total iterations, and 2) leading to better solutions for non-convex problems.

In this paper, we consider an inherently different use-case of machine learning in solving optimization problems. We propose to replace the classic iterative solutions of an optimization problem with a trainable parametric (set) function that directly maps the input of the optimization problem to the optimal parameters in a single feed forward. This process, which we denote as *Learning to Optimize the Optimization Process* ($\mathcal{LOOP}$), is inspired by biological systems that are capable of solving complex optimization problems upon encountering the problem multiple times. By omitting the classic iterative solutions, $\mathcal{LOOP}$ overcomes one of the major optimization bottlenecks enabling near-real-time optimization in a wide range of critical applications.

$\mathcal{LOOP}$ is particularly suitable when one needs to perform a certain type of optimization (e.g., linear/quadratic programming) over a specific distribution of input data (e.g., sensors data collection) repeatedly. These problems abound in practice, with examples being cyber-physical infrastructures, autonomous vehicle networks, sensor networks monitoring a physical field, financial markets, and supply chains. For example, the resiliency and cost-effectiveness of our cyber-physical energy system relies on finding optimal energy dispatch decisions in near-real-time. This is a prime example of an optimization required to be repeatedly solved over the distribution of electricity demands on the power grid. Another example is traffic flow management in transportation networks, where traffic control systems need to determine traffic lights' status based on the traffic measurements continuously.

At a first glance, the use of neural networks for solving frequently solved optimization problems may seem inefficient. However, such paradigm shift would allow us to leverage recent advances in deep learning, in particular, deep learning on edge-devices, continual learning, and transfer learning to improve the performance of an optimizer over time, even for a fixed given computational budget. Below we enumerate our specific contributions.

1) Providing a generic framework, $\mathcal{LOOP}$, for replacing the classic iterative optimization algorithms with a trainable parametric (set) function that outputs the optimal arguments/parameters in a single feed forward.
2) Proposing two generic approaches for training parametric (set) functions to solve a certain type of optimization problem over a distribution of input data.
3) Demonstrating the success of our $\mathcal{LOOP}$ framework in solving various types of optimization problems including linear/nonlinear regression, principal component analysis, the optimal transport-based coreset, and the quadratic programming in supply management application.

II. RELATED WORK

One of the classic applications of machine learning in optimization has been in predicting proper hyper-parameters to solve an optimization problem. Such hyper-parameters could include learning rate, momentum decay, and regularization coefficients, etc. The existing literature on learning to predict hyper-parameters include approaches based on sequential model-based Bayesian optimization (SMBO) [1], [2], [3], and gradient-based methods [4], [5], [6]. At their core, these methods instantiate different variations of the same optimization algorithm, e.g., stochastic gradient descent (SGD), by selecting different hyper-parameters.

More recently, a large body of work has focused on leveraging machine learning to improve the optimization process by replacing the engineered optimizers with learnable ones. These methods, referred to as Learning to Optimize (L2O) approaches, are based on learning a parametric function, often in the form of a recurrent neural network, that receives the current loss (or its gradient) as input and outputs the parameter updates [7], [8], [9], [10], [11]. Such methods are effective in optimizing a wide range of optimization problems by reducing the number of iterations and often achieve better solutions for non-convex optimization problems. Chen et al. [12] provide a comprehensive review of these approaches and their numerous applications. Unlike the hyper-parameter search methods that instantiate different variations of the same optimization algorithm (e.g., SGD), L2O approaches effectively search over an expansive space of optimization algorithms to find an optimal algorithm. The optimal algorithm (i.e., the learned optimizer) fits input data distribution for a specific optimization problem (e.g., linear/quadratic programming); hence, it can lead to better performance than generic algorithms.

In this paper, our focus is entirely different from both hyper-parameter optimization approaches, and *L2O* approaches discussed above. Instead of searching in the space of possible optimizers, our goal is to replace the optimization algorithm with a parametric (set) function that directly maps the optimization's input data to the optimal arguments/parameters. The motivation behind such transition is to: 1) discard iterations altogether, 2) have an optimizer that improves over time and encounters more optimization problems of a specific type. More importantly, the proposed framework allows one to leverage some of the core machine learning concepts, including continual/lifelong learning, transfer learning, domain adaptation, few/one/zero-shot learning, model compression (through sparse training and/or training), and many others into the improving the optimization process.

Several recent papers in the literature leverage deep neural networks to approximate the output of an optimization algorithm, which is in essence similar to our proposed framework, $\mathcal{LOOP}$. In VoxelMorph, for instance, Balakrishnan et al. [13] trained a convolutional neural network to register medical images; image registration is a non-convex optimization problem often solved through time-consuming iterative and multi-scale solvers. In an entirely different application, Pan et al. [14] trained a neural network to predict the set of independent operating variables (e.g., energy dispatch decisions) for optimal power flow (OPF) optimization problems, denoted as DeepOPF. They showed that DeepOPF requires a fraction of the time used by conventional solvers while resulting in competitive performance. More recently, Knyazev et al. [15] trained a neural network to directly predict the parameters of an input network (with unseen architecture) to solve the CIFAR-10 and ImageNet datasets. $\mathcal{LOOP}$ is the common theme behind these seemingly unrelated works. In this paper, we establish $\mathcal{LOOP}$ as a generic alternative framework to the classic optimization algorithms, as well as, the L2O approaches, and show that many optimization problems can be directly solved through training neural networks.

III. METHOD

We start by considering unconstrained optimization problems of the following type:

$$u^* = \arg\min_u f(\mathcal{X}, u) \tag{1}$$

where $\mathcal{X} = \{x_n \in \mathbb{R}^d\}_{n=1}^N$ is the set of inputs to the distribution, $u \in \mathbb{R}^l$ is the optimization parameters, and $f(\mathcal{X}, u)$ is the objective function with respect to parameters u and inputs $\mathcal{X}$. To replace this optimization with a set function approximator, we propose two approaches as in Figure 1.

Solver in the $\mathcal{LOOP}$– In our first formulation, during the training, we use the classic solvers to obtain u^* and use it as the ground truth. Then we pose the problem as a supervised learning problem. Our training objective is shown below:

$$\begin{aligned} &\arg\min_\theta \quad \mathbb{E}_{\mathcal{X}\sim P_\mathcal{X}}[d(\phi_\theta(\mathcal{X}), u^*)] \\ &s.t.\ u^* = \arg\min_u f(\mathcal{X}, u) \end{aligned} \tag{2}$$

where $d(\cdot,\cdot) : \mathbb{R}^l \times \mathbb{R}^l \to \mathbb{R}_+$ is a discrepancy/distance defined in $\mathbb{R}^l$, and ϕ_θ denotes our set neural network, and $P_\mathcal{X}$ is a set distribution.

Without Solver– The use of a solver in our first formulation could be limiting, as such solvers are often computationally expensive turning the training excruciatingly slow. More importantly, in non-convex problems the calculated u^* for input $\mathcal{X}$ is not unique (e.g., due to different initialization), which leads to solving a regression problem with changing targets. To avoid these problems, in our second formulation, we directly optimize the objective function and with a slight abuse of the term call it a "self-supervised" formulation:

$$\arg\min_\theta \quad \mathbb{E}_{\mathcal{X}\sim P_\mathcal{X}}[f(\mathcal{X}, \phi_\theta(\mathcal{X}))] \tag{3}$$

where the expected objective value over the distribution of the input sets is minimized. Note, for constrained problems (depending on the use case) we leverage different optimization techniques. For instance, we can enforce simple constraints (e.g., $u \geq 0$) into our model (i.e., the set function) using Rectified Linear Unit (ReLU) activations in the output layer of our network. Also, we can use the Lagrange dual function and absorb the constraints into our objective function as penalty terms. Next we describe the different optimization problems we consider in this paper.

A. PROBLEM 1: LINEAR/NONLINEAR REGRESSION

We start by the simple and yet routine problem of regression. Let $\mathcal{X}_i = \{(x_n^i \in \mathbb{R}^d, y_n^i \in \mathbb{R})\}_{n=1}^{N_i}$ where the goal is to learn a parametric function $\rho_u : \mathbb{R}^d \to \mathbb{R}$. Here, index i refers to the i'th regression problem of interest. In linear regression, $\rho_u(x) = u^T x$ (we absorbed the bias into x for simplicity of notation). For nonlinear regression $\rho_u(x) = u^T\psi(x)$, $\psi : \mathbb{R}^d \to \mathbb{R}^l$ is a nonlinear mapping to a feature space (i.e., the kernel space). The optimization problem is then as follows:

$$u^* = \arg\min_u \frac{1}{2}\sum_{n=1}^N \|\rho_u(x_n) - y_n\|_2^2 + \lambda\Omega(u) \tag{4}$$

where $\Omega(u)$ is the regularization term (e.g., ℓ_2 or ℓ_1 norm), and λ is the regularization coefficient. Our goal is then to learn a network that can solve the regression problem for unseen input data.

B. PROBLEM 2: PRINCIPAL COMPONENT ANALYSIS

Next, we consider the principle components analysis (PCA) problem, a common technique to project high-dimensional samples into a lower dimensional space while maximizing the variation of the data. Let $\mathcal{X}_i = \{x_n^i \in \mathbb{R}^d\}_{n=1}^{N_i}$, then PCA seeks an orthornormal set of k vectors, $\{w_l\}_{l=1}^k$ such that:

$$w_l = \arg\max_w \quad w^T S_i w \quad s.t.\ w_j^T w_l = \begin{cases} 1 & j = l \\ 0 & j < l \end{cases}$$

where $S_i = \frac{1}{N_i}\sum_{n=1}^{N_i}(x_n^i - \bar{x}^i)(x_n^i - \bar{x}^i)^T$ is the covariance matrix of the data, and $\bar{x}^i = \frac{1}{N_i}\sum_{n=1}^{N_i} x_n^i$ is the mean. Deriving the closed-form-solution for this problem involves calculation of the eigenvectors of the covariance matrix, i.e., $S_i w_l^* = \lambda_l w_l^*$. Here λ_l and w_l^* are the l'th eigenvalue and eigenvector, respectively. This optimization problem can be presented as a set-function that receives a set of d-dimensional points, $\mathcal{X}^i$ with cardinality $|\mathcal{X}_i| = N_i$, and returns $U^* = [w_1^*, w_2^*, \ldots, w_k^*]$. Using this representation, $\mathcal{LOOP}$ approximates the discussed set-function and outputs the top k principle components for the input set. Hence, we aim to find a ϕ_θ, such that $\phi_\theta(\mathcal{X}) \approx U^*$ for $\mathcal{X} \sim P_\mathcal{X}$.

C. PROBLEM 3: OPTIMAL TRANSPORT-BASED CORESET

For our third problem, we consider the optimal transport-based coreset problem. The notion of coreset originates from computational geometry [16] and has been widely used in machine learning tasks. Constructing a coreset from a large dataset is an optimization problem of finding a smaller set to best approximate the original dataset on a certain measure. Claici et al. [17] leveraged optimal transport theory and introduced Wasserstein measure to calculate the coreset. Their work aims to minimize the Wasserstein distance of the coreset from a given input data distribution. In this paper we consider

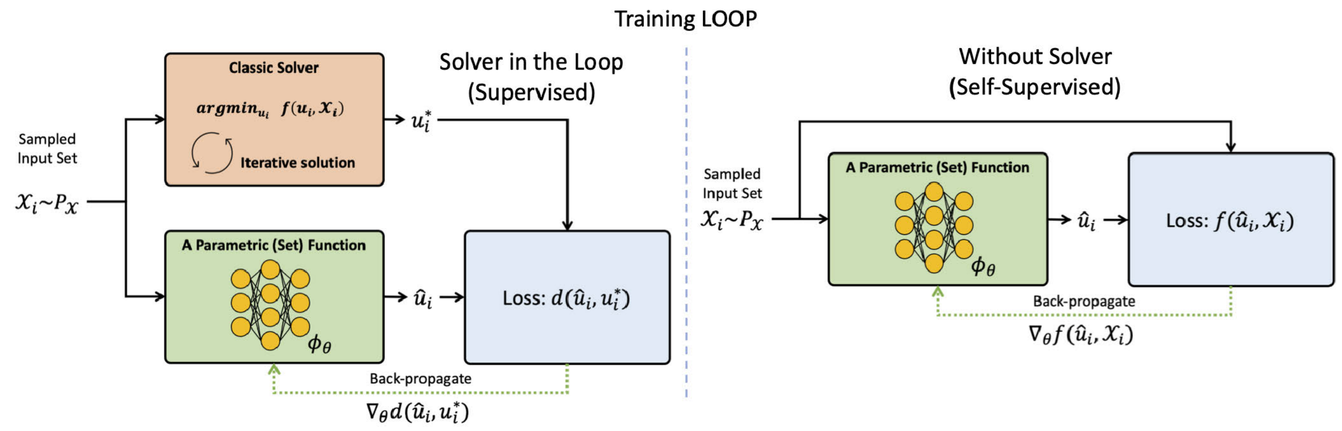
Training LOOP
Solver in the Loop (Supervised)
Without Solver (Self-Supervised)
Sampled Input Set
$\mathcal{X}_i \sim P_\mathcal{X}$
Classic Solver
$argmin_{u_i}\ f(u_i, \mathcal{X}_i)$
Iterative solution
u_i^*
A Parametric (Set) Function
ϕ_θ
$\hat{u}_i$
Loss: $d(\hat{u}_i, u_i^*)$
Back-propagate
$\nabla_\theta d(\hat{u}_i, u_i^*)$
Loss: $f(\hat{u}_i, \mathcal{X}_i)$
$\nabla_\theta f(\hat{u}_i, \mathcal{X}_i)$

FIGURE 1. Our two proposed approaches for training $\mathcal{LOOP}$: 1) with solver in the loop (left), and 2) without solver in the loop and by directly minimizing the objective function (right).

this transport-based coreset problem with respect to a fixed size output.

Let $\mathcal{X} = \{x_n \in \mathbb{R}^d\}_{n=1}^N$ be an input set. We assume that elements of each set are i.i.d. samples from an underlying distribution. Our sets are represented as empirical distributions, i.e., $p(x) = \frac{1}{N}\sum_{n=1}^N \delta(x - x_n)$. Given a size M ($M \ll N$), we seek a set $\mathcal{U}^* = \{\mu_m \in \mathbb{R}^d\}_{m=1}^M$ with the empirical distribution $q_\mathcal{U}(x) = \frac{1}{M}\sum_{m=1}^M \delta(x - \mu_m)$, such that

$$\mathcal{U}^* = \arg\min_{\mathcal{U}} W_2(p, q_\mathcal{U}) \tag{5}$$

where $W_2(\cdot,\cdot)$ denotes the 2-Wasserstein distance. Existing approaches to this optimization problem rely on iterative linear programming to compute optimal transports in each iteration. We replace this costly process with a parametric set function ϕ_θ such that $\phi_\theta(\mathcal{X}) \approx \mathcal{U}^*$ for $\mathcal{X} \sim P_\mathcal{X}$ (Figure 2). The optimal transport-based coreset problem is equivalent to the free-support Wasserstein barycenter problem [18] when there is only one input distribution.

D. PROBLEM 4: SUPPLY MANAGEMENT IN CYBER-PHYSICAL SYSTEMS

Lastly, we utilize $\mathcal{LOOP}$ to solve the fundamental problem of supply management in Cyber-Physical Systems (CPS). The electric power grid is an example of a CPS that is increasingly facing supply-demand issues [19], [20], [21]. Power networks are large-scale systems spanning multiple cities, states, countries, and even continents and are characterized as a complex interconnect of multiple entities with diverse functionalities. The grid of the future will differ from the current system by the increased integration of decentralized generation, distributed storage, and communications and sensing technologies. These advancements, combined with climate change concerns, resiliency needs, and electrification trends, are resulting in a more distributed and interconnected grid, requiring decisions to be made at scale and in a limited time window. At its basic form, the energy supply-demand problem seeks to find the most cost-effective power production for meeting the end-users' needs and can be formulated as,

$$\begin{aligned} &\arg\min_u \sum_{n=1}^N C_n(u_n) \\ &s.t. \sum_{n=1}^N u_n = \sum_{m=1}^M x_m, \quad \underline{u}_n \le u_n \le \overline{u}_n \end{aligned} \tag{6}$$

where u_n is the produced electric power from source n and C_n is its corresponding cost, which is a quadratic function. Given that u_n represents the power output, it is bounded by physical limitation of the resource n, i.e., $\overline{u}_n$ and $\underline{u}_n$. In this setup, x_m refers to the hourly electric demand in node m (where the term 'node' identifies an end-user/consumer). Note, the values of x_m are positive. The equality constraint ensures the supply-demand balance. In practice, this problem is solved on an hourly basis to serve the predicted electric demand for the next hour. We aim to approximate this process with a parametric set function, such that $\phi_\theta(\mathcal{X}) \approx U^*$ for $\mathcal{X} \sim P_\mathcal{X}$.

IV. EXPERIMENTS

In this section, we demonstrate the application of $\mathcal{LOOP}$ on problems enumerated in Section III and compare it to classic solvers. Throughout this section, GT refers to the Ground Truth and Solver refers to the results obtained from using commercial solvers to solve optimization problems of interest. For each problem and for each model architecture, we repeat the training of our $\mathcal{LOOP}$ models five times, and we test the performance on a set of 100 problems per model. We then report the mean and standard deviations of all experiments over the five models and the 100 test sets. We start by laying out the specifics of our models and then discuss the implementation details for each problem.

A. MODELS

Given that the inputs to our optimization problems are all sets, we pose these problems as learning permutation invariant deep

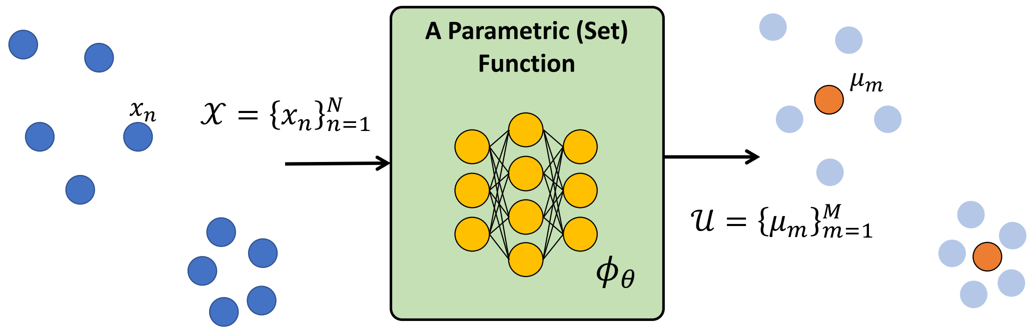
A Parametric (Set) Function
x_n
$\mathcal{X} = \{x_n\}_{n=1}^N$
ϕ_θ
μ_m
$\mathcal{U} = \{\mu_m\}_{m=1}^M$

FIGURE 2. For an input set $\mathcal{X} = \{x_n \in \mathbb{R}^d\}_{n=1}^N$, $\mathcal{LOOP}$ returns a coreset $\mathcal{U} = \{\mu_m \in \mathbb{R}^d\}_{m=1}^M$ that minimizes the Wasserstein distance between the empirical distributions $p(x) = \frac{1}{N}\sum_{n=1}^N \delta(x - x_n)$ and $q_{\mathcal{U}}(x) = \frac{1}{M}\sum_{m=1}^M \delta(x - \mu_m)$.

neural networks on set-structured data. To that end, we use Deep Sets [22] with different pooling mechanisms and the Set Transformer [23].

Deep Sets are permutation invariant neural architectures (i.e., the output remains unchanged under any permutation of input set's elements), which consist of: 1) a multi-layer perceptron (MLP) encoder, 2) a global pooling mechanism (e.g., average pooling), and 3) a MLP decoder that projects the pooling representation to the output; $\phi(\mathcal{X}) = \psi(pool(\{\eta(x_1), \cdots, \eta(x_n)\}))$. Here, η is the encoder that extracts features from each element of $\mathcal{X}$ independently, resulting in a permutation equivariant function on the input set, and ψ is the decoder that generates final output after a pooling layer ($pool$). To achieve a permutation invariance set function, the pooling mechanisms must be a permutation invariance operator (e.g., average pooling, or more advanced methods like Pooling by sliced-wasserstein embedding (PSWE) [24]). Specifically, we use global average pooling (GAP) and Sliced-Wasserstein Embedding (SWE) [24], [25] respectively as the pooling layer.

Set Transformer follows a similar blueprint of permutation equivariant encoder, permutation invariant pooling, and permutation equivariant decoder as Deep Sets. However, while the encoder in the Deep Sets model acts on each set element independently, Set Transformers use attention to pass information between elements in the encoder. This allows the encoder to model relations between elements, which can be crucial to approximate a parametric (set) function in some learning tasks.

More precisely, the encoder is a stack of multiple trainable (Induced) Set Attention Blocks (SAB and ISAB) [23] that perform self-attention operations on a set and produce output containing information about pairwise relations between elements. Note that these blocks are permutation equivariant, that is, for any permutation π of elements in $\mathcal{X} = \{x_i\}_{i=1}^n$, $block(\pi\mathcal{X}) = \pi block(\mathcal{X})$. As a composition of permutation equivariant blocks, the encoder is also permutation equivariant and captures higher-order interaction features. The decoder aggregates features by a learnable pooling layer, Pooling by Multihead Attention (PMA) [23], and send them through a SAB to get output. Since PMA is a permutation invariant operator, and the rest of the operators (SAB or ISAB) are all permutation equivariant, Set Transformer becomes a permutation invariant architecture.

B. PROBLEM 1: LINEAR/NONLINEAR REGRESSION

1) DATASET

We follow a generative model $y = w^T\phi(x) + \epsilon$, where $\phi(\cdot)$ is the feature map, w contains the ground truth parameters of our regression problem, and ϵ denotes noise. For feature maps, in the linear case we have $\phi(x) = [1, x]^T$ and in the nonlinear case, we select $\phi(x) = [\rho(x-\mu_1), \ldots, \rho(x-\mu_M)]$ with $\rho(x)$ being a radial basis function and $\{\mu_m\}_{m=1}^M$ form a grid in a predefined interval. To generate each dataset $\mathcal{X}_i$, we first sample the set cardinality N_i uniformly from a predefined interval. Then, we sample w, $\{\epsilon_n\}_{n=1}^{N_i}$, and $\{x_n\}_{n=1}^{N_i}$, and generate our (x_n^i, y_n^i) pairs (train and test).

For each model architecture and each learning setting (i.e., with and without solver in the $\mathcal{LOOP}$) we train our $\mathcal{LOOP}$ model 5 times and report the test MSE of our model, the solver, and the ground truth. Figure 4 shows sample qualitative results of our nonlinear regression experiments with ground truth, solver, and $\mathcal{LOOP}$ results overlayed on the observed noisy data. In addition, for the Set Transformer architecture, we report the test performance of our trained $\mathcal{LOOP}$ model and the solver as a function of the number of training samples (Figure 3). We see that while for all architectures $\mathcal{LOOP}$ is able to perform comparable with the solver, for the Set Transformer architecture the gap between $\mathcal{LOOP}$ and the solver is the smallest.

C. PROBLEM 2: PRINCIPAL COMPONENT ANALYSIS

1) DATASET

We used the MNIST [26] dataset to sample train and test sets. MNIST contains 60,000 train and 10,000 test images of handwritten digits. The size of a single image is 28 × 28. During training, we first select pairs of random digits to sample images from. Then a random number of data ranging from 500 to 1000 is uniformly sampled from the two digits to form the input set.

Given an input set $\mathcal{X}_i$, our network aims to predict the top $K = 5$ eigenvectors of the input data. In "solver in

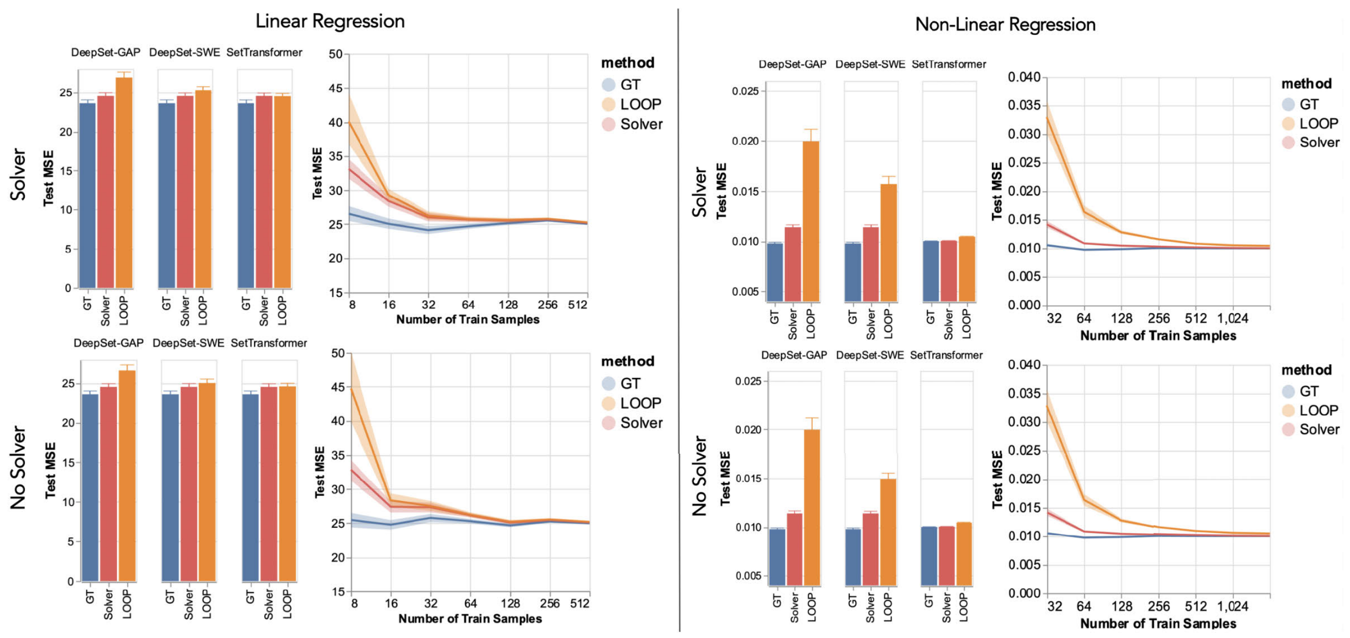

FIGURE 3. Performance comparison between $\mathcal{LOOP}$ and the solver for three different model architectures (left) and for the two proposed learning settings (with or without the solver in the $\mathcal{LOOP}$) for the linear regression (a) and nonlinear regression (b) problems. The plots on the right shows the performance of the Set Transformer network and the solver as a function of training samples.

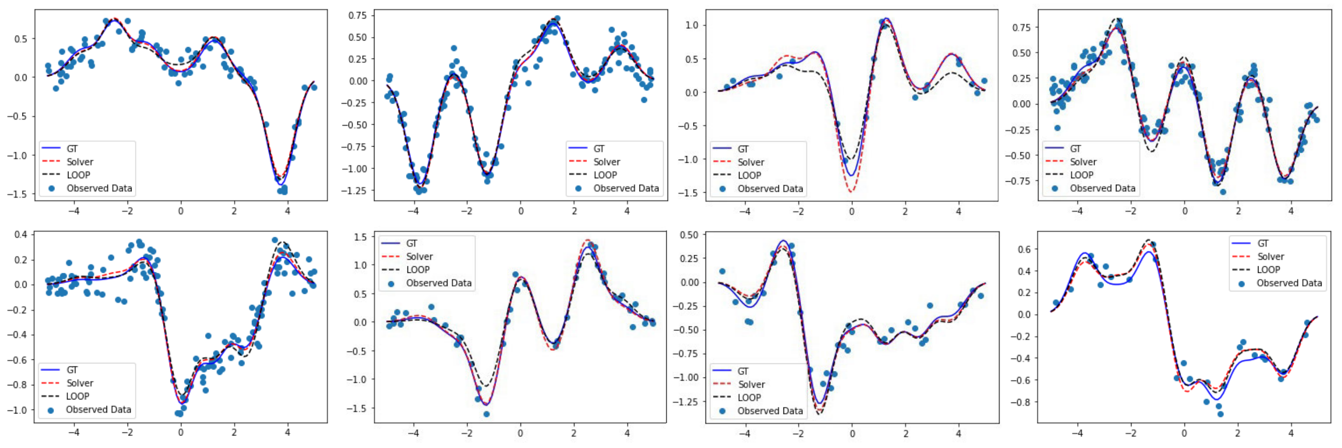

FIGURE 4. Sample qualitative results for our nonlinear regression problem compared with the Ground Truth (GT) and the solver's result. Note that the input set cardinality is a random variable and it varies among these plots.

the $\mathcal{LOOP}$'' approach, the top $K = 5$ eigenvectors are calculated by the solver. Then our set transformer [23] is trained to maximize the cosine similarities between the ground-truth eigenvectors and the predicted ones. In our ``no solver'' approach, the set transformer maximizes the area under the curve of the captured variances along the predicted eigenvectors. We train 5 different models for this experiment and evaluated each model on 100 different test problems. For our metric, we calculate the cosine similarities between the predicted vectors and the principle components obtained from the solver. The mean and standard deviation of the cosine similarities for each eigenvector is depicted in Figure 5 (left). We also show the performance of the trained model as a function of different number of training samples from 128 to 2048 (on the right). Results of ``solver in the $\mathcal{LOOP}$'' and ``no solver'' training are shown for the Set Transformer model in 5. Our network is able to effectively predict the top principle components in all experiments, while having a higher fidelity for the ones with larger eigenvalues.

D. PROBLEM 3: TRANSPORT-BASED CORESET

1) DATASET

We generate datasets by drawing samples from random 2D Gaussian Mixture Models (GMMs). We start by initializing a random number of Gaussians with random means but a fixed

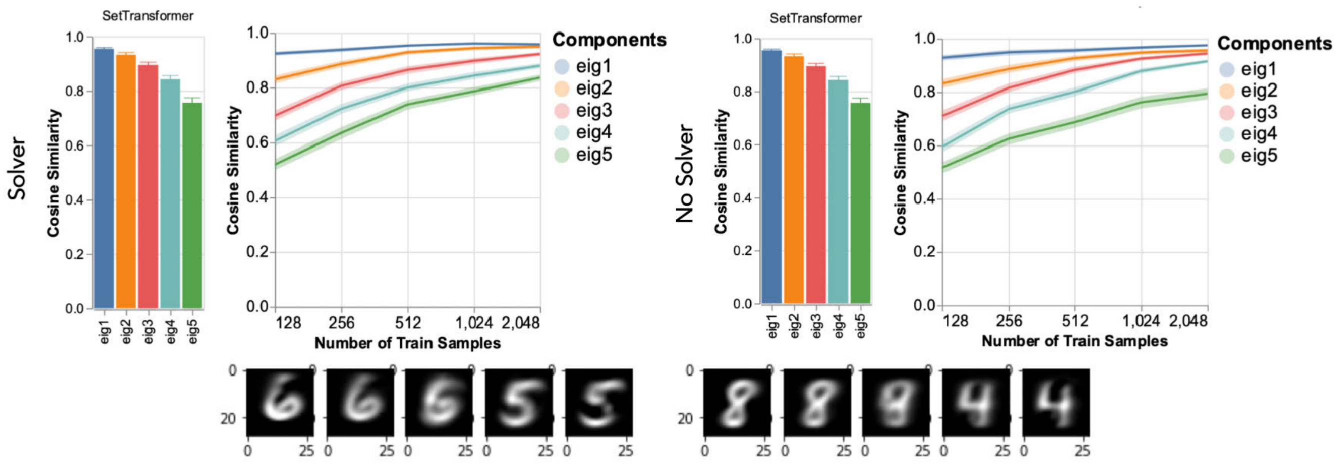

FIGURE 5. $\mathcal{LOOP}$'s performance in predicting the principle components as measured by the cosine similarity between the solver's and our model's outputs. We also provide the performance of the network as a function of the input data cardinality. On the bottom is the visualization of the first eigenvector calculated by our $\mathcal{LOOP}$ model on four different problems with two random pairs of digits. We can see that the network's output is quantitatively and qualitatively aligned with the first principle component.

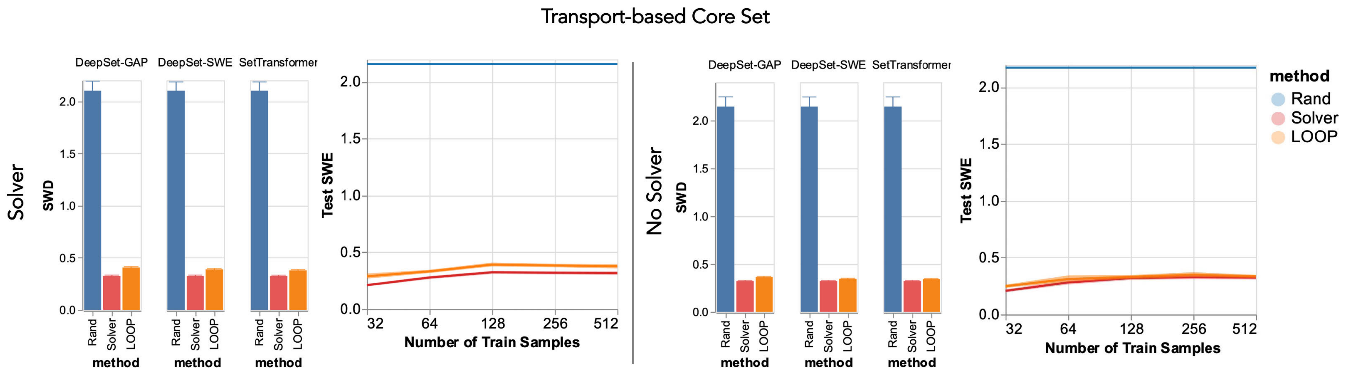

FIGURE 6. Performance comparison between $\mathcal{LOOP}$ and the solver for our three different models (left) under the two proposed learning settings. The y-axis represents the average Wasserstein distance between the input distribution p and the coreset distribution $q_{\mathcal{U}}$, when the coreset are: 1) random samples from the uniform distribution, 2) the output of the solver, and 3) the output of our $\mathcal{LOOP}$ model. The plots on the right of each column show the performance of the Set Transformer and the solver as a function of the number of training samples.

covariance matrix. Then, we draw random number of samples from each of these randomly initialized Gaussians to generate our input sets $\mathcal{X}^i$.

2) RESULTS

Results of the two training approaches "solver in the $\mathcal{LOOP}$" and "no solver" using three different models are shown in 6. Since the problem is equivalent to the free-support Wasserstein barycenter problem, we used the solver from the Python Optimal Transport package [27] as the solver. To compare the output of our $\mathcal{LOOP}$ model with the solver, we use $W_2(p, q_{\mathcal{U}})$ as our metric (the lower the better). Also, to provide a reference for comparison, we also consider the Wasserstein distance between the input distribution, p, and a uniform distribution in the input domain, $\bar{q}$, which we refer to as Rand (equivalent to chance). We used the Sliced-Wasserstein distance (SWD) [28] as the objective function in the "no solver" training, as SWD is significantly faster to compute than the Wasserstein distance. Finally, we compare the performance of $\mathcal{LOOP}$ and the solver as a function of number of training samples in Figure 6.

E. PROBLEM 4: SUPPLY MANAGEMENT IN CYBER-PHYSICAL SYSTEMS

1) DATASET

We use the publicly available IEEE 2000-bus system data set [29] as the seed infromation to generate hourly energy data for one week. We use different load profiles for weekdays and weekends and randomly scale the original data. The scaling coefficient lies between 0.95 and 1.05. This process results in 24×7 data points. We use the data of odd hours for training and that of even hours for testing. The IEEE 2000-bus system is a 2,000 nodes graph representing a realistic large scale electric grid. This network consists of 1,125 demand nodes and 544 supply nodes.

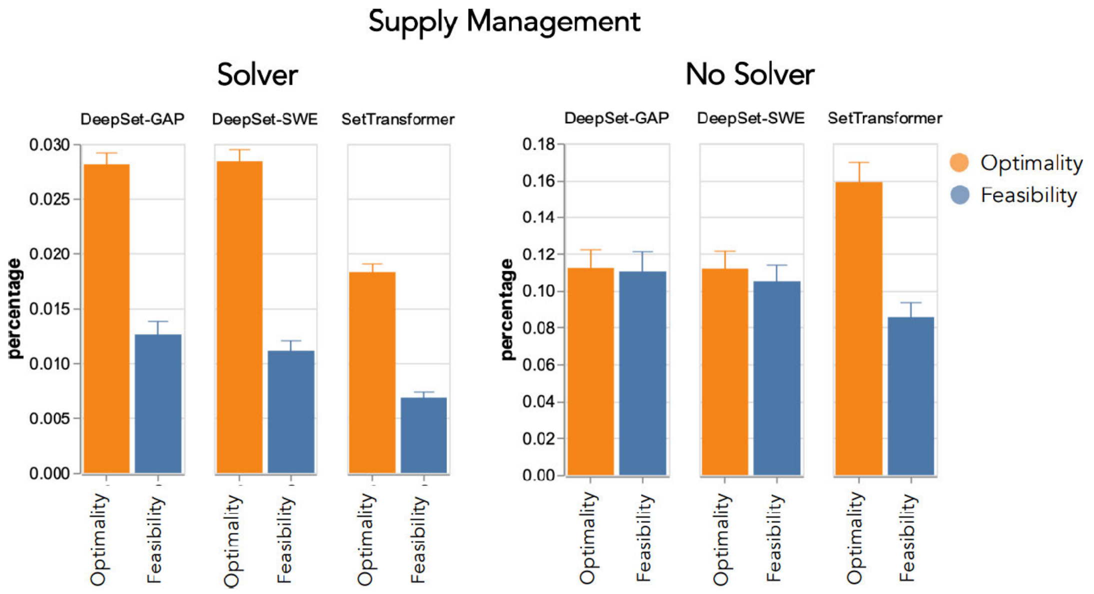

FIGURE 7. $\mathcal{LOOP}$ **'s performance measured by the distance of its output from the optimal solution and the feasible set We define the optimality distance as $\sum_{n=1}^{N} |u_n - u_n^*| / \sum_{n=1}^{N} u_n^*$, where u_n^* and u_n refer to the solver's and $\mathcal{LOOP}$'s outputs. The feasibility distance $\sum_{n=1}^{N} \left|u_n - u_n^{proj}\right| / \sum_{n=1}^{N} u_n^{proj}$ where u_n^{proj} denotes projection of u_n onto the feasible set.**

2) RESULTS

For solver in the $\mathcal{LOOP}$, we use the mean squared error as the loss function, i.e., $L = \frac{\sum_{n=1}^{N}(u_n - u_n^*)^2}{n}$. Here $\{u_n^*\}_{n=1}^{N}$ are the solver's output. We use the quadratic programming (QP) solver of CVXPY library [30] as our solver. In our second learning setting, i.e., with no solver in the $\mathcal{LOOP}$, we include the optimization constraints in our objective as penalty terms. Therefore, the loss function will consist of three terms,

$$L = \sum_{n=1}^{N} C_n(u_n) + \lambda_1 \left(\sum_{n=1}^{N} u_n - \sum_{m=1}^{M} x_m \right)^2 + \lambda_2 \left[(ReLU(\underline{u}_n - u_n))^2 + (ReLU(u_n - \overline{u}_n))^2 \right] \tag{7}$$

where λ_is are the penalty coefficients. We use $\lambda_1 = 0.001$, $\lambda_2 = 10$ in our experiments.

In 7, the first term represents the cost of electricity production. The second term ensures the equality of supply and demand, and the third term enforces the supply to be bounded. We bound the output according to inequality constraints for testing. To quantify the performance of our $\mathcal{LOOP}$ model, we report two metrics: 1) optimality, which measures how far we are from the solver's output, and 2) feasibility, which measures the distance of $\mathcal{LOOP}$'s output from the feasible set. We measure feasibility distance by projecting the network's output onto the feasible set and measuring the distance between the original and project solutions.

Figure 7 shows the results for two $\mathcal{LOOP}$ approaches (solver in the $\mathcal{LOOP}$and no solver) using different models. The gap between our two learning settings for this problem is more significant than previous unconstrained ones. This is because the $\mathcal{LOOP}$ model with no solver minimizes a combination of the objective function and the penalty terms. Unlike the solver in the $\mathcal{LOOP}$ model (which could leverage optimality information), the $\mathcal{LOOP}$ model with no solver does not establish any relation between feasibility and optimality. We also present results of different penalty parameters for the $\mathcal{LOOP}$ model with no solver in the supplemental materials, where the gap between the two $\mathcal{LOOP}$ approaches is reduced by more careful tuning of penalty terms (λ_1 and λ_2). Moreover, Table 1 presents the average computing time of different solvers (ECOS [31], CVXOPT [32], OSQP [33], Matpower 7.1 [34]), and $\mathcal{LOOP}$ over ten runs. Using GPU, three $\mathcal{LOOP}$ approaches outperform all solvers. On CPU, the $\mathcal{LOOP}$ model of Set Transformer performs on par with the today's solvers while other $\mathcal{LOOP}$ models are noticeably faster.

F. CONTINUAL LEARNING ON $\mathcal{LOOP}$

A promising capability of $\mathcal{LOOP}$ is to continually train the network to adjust to the change in the input distribution. Note, $\mathcal{LOOP}$ (specifically in the no solver in the $\mathcal{LOOP}$ setting) is ripped for continual learning. In short, the $\mathcal{LOOP}$ agent can evaluate the quality of its prediction (i.e., by measuring the objective value or by checking the feasibility) and perform continual learning if the prediction quality is degraded. We perform a continual learning experiment on $\mathcal{LOOP}$, while the agent is tasked to solve nonlinear regression problems where the frequency spectrum of the input data drifts from Task 1 to Task 2. To overcome catastrophic forgetting, we use memory replay [35] as one of the core bio-inspired mechanisms for overcoming catastrophic forgetting [36]. Table 2 demonstrates the application of $\mathcal{LOOP}$ in continual

TABLE 1. Average computing time of using different solvers and $\mathcal{LOOP}$ approaches over ten runs.

	Method	Time(s)		Method	Time(s)
Solvers	ECOS Solver (CPU)	0.1237	$\mathcal{LOOP}$	SetTransformer(CPU)	0.1284
	CVXOPT Solver(CPU)	1.8277		DeepSet-SWE(CPU)	0.0600
	OSQP Solver(CPU)	0.1421		DeepSet-GAP(CPU)	0.0538
	ECOS Solver(GPU)	0.1005		SetTransformer(GPU)	0.0057
	CVXOPT Solver(GPU)	1.6288		DeepSet-SWE(GPU)	0.0070
	OSQP Solver(GPU)	0.1218		DeepSet-GAP(GPU)	0.0022
	Matpower 7.1 Solver	0.9609			

TABLE 2. Test MSE for both tasks after each training phase using $\mathcal{LOOP}$ (left) and $\mathcal{LOOP}$ with memory replay (right). The models are trained on task 1 in phase 1 and on task 2 in phase 2.

	Task 1	Task 2
test MSE after phase 1	0.091	4.805
test MSE after phase 2	0.287	0.130

	Task 1	Task 2
test MSE after phase 1	0.091	4.805
test MSE after phase 2	0.088	0.136

learning of non-linear regression under domain shift. We see that memory replay enables $\mathcal{LOOP}$ to learn the new task (Task 2) while achieving positive backward transfer on Task 1. More details are provided in the supplementary materials. Lastly, the application of other continual learning mechanisms like regularization-based approaches [37] and gradient projection approaches [38], [39], opens up an exciting research direction for future work.

V. CONCLUSION

This paper presents a novel alternative for existing iterative methods to solve optimization problems. Specifically, this paper introduces $\mathcal{LOOP}$ (Learning to Optimize Optimization Process) framework, which approximates the optimization process with a trainable parametric (set) function. Such a function maps optimization inputs to the optimal parameters in a single feed forward. We proposed two approaches for training $\mathcal{LOOP}$; using a classic solver for providing ground truth (supervised learning) and without a solver in the $\mathcal{LOOP}$ (self-supervised learning). The performance of the proposed methods is showcased in the contexts of diverse optimization problems; (i) linear and non-linear regression, (ii) principal component analysis, (iii) transport-based coreset, and (iv) supply management in cyber-physical setups. We used three separate models in our experiments, namely deep sets with global average pooling, deep sets with sliced-Wasserstein Embedding, and Set Transformers. Our results supports that replacing optimization problem with a single forward mapping yields outputs within a reasonable distance from commercial solvers' solutions. $\mathcal{LOOP}$ holds the promise for the next generation of optimization

APPENDIX A
OPTIMAL TRANSPORT

We present additional results of applying our proposed $\mathcal{LOOP}$ framework to the optimal transportation problem with respect to a fixed reference. Let $\mathcal{X} = \{x_n \in \mathbb{R}^d\}_{n=1}^N$ be an input set and let $\mathcal{Y} = \{y_m \in \mathbb{R}^d\}_{m=1}^M$ represent our fixed reference set. As in transport-based coreset problem, we assume that the elements of each set are i.i.d. samples

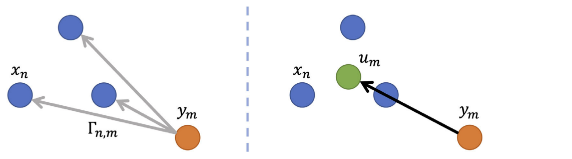

FIGURE 8. Under the transportation plan, Γ, the particle at y_m splits and goes to multiple x_ns (on the left), while the barycentric projection (on the right) captures the center of mass where y_m is transported to.

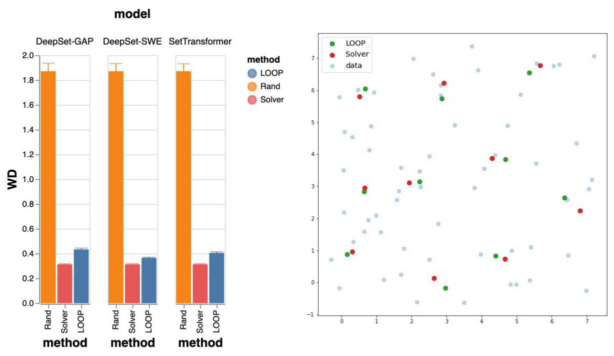

FIGURE 9. Quantitative (left) and qualitative (right) performance comparison between $\mathcal{LOOP}$ and the solver. The y-axis in the quantitative results (left) represents the average Wasserstein distance between the input distribution $P_{\mathcal{X}}$ and the approximated Monge coupling U.

from an underlying distribution and represent our sets as empirical distributions, $p(x) = \frac{1}{N}\sum_{n=1}^N \delta(x - x_n)$ and $q(y) = \frac{1}{M}\sum_{m=1}^M \delta(y - y_m)$. Then the optimal transport problem seeks the optimal transportation plan, $\Gamma \in \mathbb{R}^{N\times M}$, such that,

$$\arg\min_{\Gamma} \sum_{n=1}^{N}\sum_{m=1}^{M} d(x_n, y_m)\Gamma_{n,m}$$
$$s.t. \sum_{n=1}^{N}\Gamma_{n,m} = \frac{1}{M}, \sum_{m=1}^{M}\Gamma_{n,m} = \frac{1}{N} \quad (8)$$

where $d(\cdot,\cdot) : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}_+$ is the transportation cost. The optimization in Eq. (8) is a linear programming problem, and, generally speaking, is $\mathcal{O}(N^3)$ to solve. The transportation plan

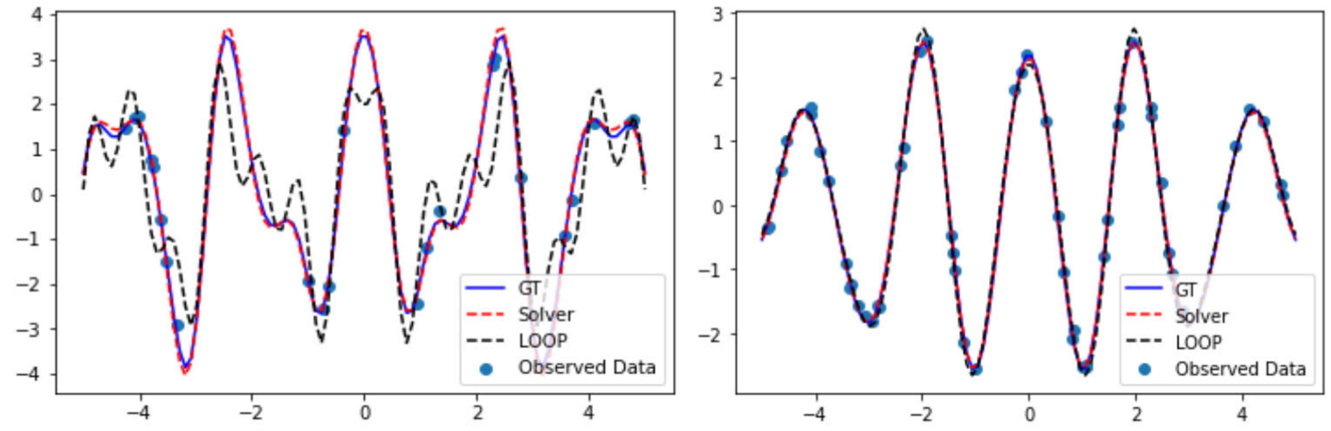

FIGURE 10. Performance on Task 1 after training on Task 2 using $\mathcal{LOOP}$ (left) and $\mathcal{LOOP}$ with memory replay (right).

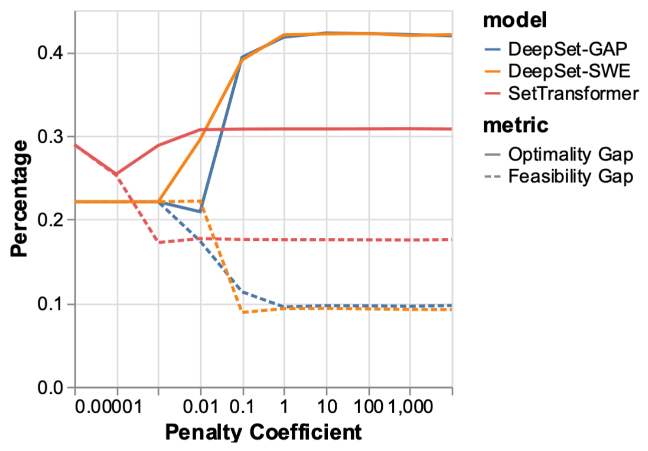

FIGURE 11. Performance of $\mathcal{LOOP}$ on the supply management problem using different penalty parameters($\lambda_1 = \lambda_2$). The y-axis is the relative error when compared to solutions from the solver (lower the better).

(i.e., the Kantorovich plan), $\Gamma_{n,m}$, represents how much mass is split and transported from x_n to y_m. An approximate Monge coupling can be attained through barycentric projection of the optimal transportation plan through:

$$u_m = M \sum_{n=1}^{N} \Gamma_{n,m} x_n \tag{9}$$

Figure 8 demonstrates the concept of the transportation plan and its barycentric projection. Matrix $U = [u_1, \ldots, u_M] \in \mathbb{R}^{d\times M}$ is related to the K-Means problem with $K = M$ and is extensively used in the literature in the context of linear optimal transport [40] and Wasserstein embeddings [41].

The optimization in Eq. (8) followed by the barycentric projection in Eq. (9) can be thought as a process that receives $\mathcal{X}$ and for a fixed $\mathcal{Y}$ returns matrix $U \in \mathbb{R}^{d\times M}$. Then, our goal is to approximate this process with a parametric set function, such that $\phi_\theta(\mathcal{X}) \approx U^*$ for $\mathcal{X} \sim P_{\mathcal{X}}$.

We use the same GMM data generator as in Section IV-D and show the results of the training approach "solver in the $\mathcal{LOOP}$" using three different models in 9.

APPENDIX B
CONTINUAL LEARNING ON $\mathcal{LOOP}$ DETAILS

For datasets, we follow the same generative model $y = \omega^T\phi(x) + \epsilon$ as in the nonlinear case of Section IV-B, but with different feature map. We select $\phi(x) = [\cos(b_1 x), \cos(b_2 x), \cdots, \cos(b_M x)]$, where the frequencies $\{b_m\}_{m=1}^M$ form a grid in a predefined interval. For Task 1, the frequency interval is set to be $[\frac{2\pi}{3}, 2\pi]$, while in Task 2, it is increased to $[\frac{4\pi}{5}, 4\pi]$. Specifically, we use the Set Transformer architecture with solver in the $\mathcal{LOOP}$ to fit the data. We train $\mathcal{LOOP}$ for 4 times with and without the memory replay and report the test MSE of both methods. Figure 10 visualizes the test results for Task 1 after training phase 2.

APPENDIX C
SUPPLY MANAGEMENT IN CYBER-PHYSICAL SYSTEMS

We present a sensitivity study of different penalty parameters for the $\mathcal{LOOP}$ model with no solver, where λ_1 is the penalty parameter for equality constraint and λ_2 is the penalty parameter for inequality constraint. As we can see from the plot, when we increase the penalty parameters, the output from $\mathcal{LOOP}$ becomes closer to the feasible set but deviates from the optimal solution.

REFERENCES

[1] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Sequential model-based optimization for general algorithm configuration," in *Proc. Int. Conf. Learn. Intell. Optim.* Cham, Switzerland: Springer, 2011, pp. 507–523.

[2] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 24, 2011.

[3] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 25, 2012.

[4] Y. Bengio, "Gradient-based optimization of hyperparameters," *Neural Comput.*, vol. 12, no. 8, pp. 1889–1900, Aug. 2000.

[5] D. Maclaurin, D. Duvenaud, and R. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *Proc. Int. Conf. Mach. Learn.*, 2015, pp. 2113–2122.
[6] Y. Wei, P. Zhao, and J. Huang, "Meta-learning hyperparameter performance prediction with neural processes," in *Proc. Int. Conf. Mach. Learn.*, 2021, pp. 11058–11067.
[7] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *Proc. 27th Int. Conf. Mach. Learn. (ICML)*, Jun. 2010, pp. 399–406.
[8] K. Li and J. Malik, "Learning to optimize," 2016, *arXiv:1606.01885*.
[9] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, 2016, pp. 3981–3989.
[10] O. Wichrowska, N. Maheswaranathan, M. W. Hoffman, S. G. Colmenarejo, M. Denil, N. Freitas, and J. Sohl-Dickstein, "Learned optimizers that scale and generalize," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 3751–3760.
[11] Y. Chen, M. W. Hoffman, S. G. Colmenarejo, M. Denil, T. P. Lillicrap, M. Botvinick, and N. Freitas, "Learning to learn without gradient descent by gradient descent," in *Proc. Int. Conf. Mach. Learn.*, 2017, pp. 748–756.
[12] T. Chen, X. Chen, W. Chen, H. Heaton, J. Liu, Z. Wang, and W. Yin, "Learning to optimize: A primer and a benchmark," 2021, *arXiv:2103.12828*.
[13] G. Balakrishnan, A. Zhao, M. R. Sabuncu, J. Guttag, and A. V. Dalca, "VoxelMorph: A learning framework for deformable medical image registration," *IEEE Trans. Med. Imag.*, vol. 38, no. 8, pp. 1788–1800, Aug. 2019.
[14] X. Pan, M. Chen, T. Zhao, and S. H. Low, "DeepOPF: A feasibility-optimized deep neural network approach for AC optimal power flow problems," 2020, *arXiv:2007.01002*.
[15] B. Knyazev, M. Drozdzal, G. W. Taylor, and A. R. Soriano, "Parameter prediction for unseen deep architectures," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 34, 2021.
[16] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan, "Geometric approximation via coresets," *Combinat. Comput. Geometry*, vol. 52, nos. 1–30, pp. 1–23, 2005.
[17] S. Claici, A. Genevay, and J. Solomon, "Wasserstein measure coresets," 2018, *arXiv:1805.07412*.
[18] M. Cuturi and A. Doucet, "Fast computation of Wasserstein barycenters," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 685–693.
[19] M. Mohammadi, J. Thornburg, and J. Mohammadi, "Towards an energy future with ubiquitous electric vehicles: Barriers and opportunities," *Energies*, vol. 16, no. 17, p. 6379, Sep. 2023.
[20] M. Mohammadi and A. Mohammadi, "Empowering distributed solutions in renewable energy systems and grid optimization," in *Distributed Machine Learning and Optimization: Distributed Machine Learning and Optimization: Theory and Applications*. Springer, 2023, pp. 1–17. [Online]. Available: http://www.wikicfp.com/cfp/servlet/event.showcfp?eventid=165754
[21] K. Nelson, P. Moura, and J. Mohammadi, "EVs and ERCOT: Foundations for modeling future adoption scenarios and grid implications," in *Proc. 11th Workshop Model. Simul. Cyber-Phys. Energy Syst. (MSCPES)*, 2023, pp. 1–6.
[22] M. Zaheer, S. Kottur, S. Ravanbhakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola, "Deep sets," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, 2017, pp. 3394–3404.
[23] J. Lee, Y. Lee, J. Kim, A. Kosiorek, S. Choi, and Y. W. Teh, "Set transformer: A framework for attention-based permutation-invariant neural networks," in *Proc. Int. Conf. Mach. Learn.*, 2019, pp. 3744–3753.
[24] N. Naderializadeh, J. F. Comer, R. W. Andrews, H. Hoffmann, and S. Kolouri, "Pooling by sliced-Wasserstein embedding," in *Proc. Adv. Neural Inf. Process. Syst.*, A. Beygelzimer, Y. Dauphin, P. Liang, and J. W. Vaughan, Eds., 2021, pp. 1–12. [Online]. Available: https://openreview.net/forum?id=1z2T01DKEaE
[25] Y. Lu, X. Liu, A. Soltoggio, and S. Kolouri, "SLOSH: Set locality sensitive hashing via sliced-Wasserstein embeddings," 2021, *arXiv:2112.05872*.
[26] Y. LeCun. (1998). *The Mnist Database of Handwritten Digits*. [Online]. Available: http://yann.lecun.com/exdb/mnist/
[27] R. Flamary et al., "POT: Python optimal transport," *J. Mach. Learn. Res.*, vol. 22, no. 78, pp. 1–8, Mar. 2021. [Online]. Available: http://jmlr.org/papers/v22/20-451.html
[28] S. Kolouri, K. Nadjahi, U. Simsekli, R. Badeau, and G. K. Rohde, "Generalized sliced Wasserstein distances," 2019, *arXiv:1902.00434*.
[29] T. Xu, A. B. Birchfield, K. M. Gegner, K. S. Shetye, and T. J. Overbye, "Application of large-scale synthetic power system models for energy economic studies," *Proc. 50th Hawaii Int. Conf. Syst. Sci.*, 2017.
[30] S. Diamond and S. Boyd, "CVXPY: A Python-embedded modeling language for convex optimization," *J. Mach. Learn. Res.*, vol. 17, no. 83, pp. 1–5, 2016.
[31] A. Domahidi, E. Chu, and S. Boyd, "ECOS: An SOCP solver for embedded systems," in *Proc. Eur. Control Conf. (ECC)*, Jul. 2013, pp. 3071–3076.
[32] L. Vandenberghe. (2010). *The CVXOPT Linear and Quadratic Cone Program Solvers*. [Online]. Available: http://cvxopt.org/documentation/coneprog.pdf
[33] B. Stellato, G. Banjac, P. Goulart, A. Bemporad, and S. Boyd, "OSQP: An operator splitting solver for quadratic programs," *Math. Program. Comput.*, vol. 12, no. 4, pp. 637–672, Dec. 2020.
[34] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Trans. Power Syst.*, vol. 26, no. 1, pp. 12–19, Feb. 2011.
[35] G. M. van de Ven, H. T. Siegelmann, and A. S. Tolias, "Brain-inspired replay for continual learning with artificial neural networks," *Nature Commun.*, vol. 11, no. 1, pp. 1–14, Aug. 2020.
[36] D. Kudithipudi, M. Aguilar-Simon, J. Babb, M. Bazhenov, D. Blackiston, J. Bongard, A. P. Brna, S. Chakravarthi Raja, N. Cheney, and J. Clune, "Biological underpinnings for lifelong learning machines," *Nature Mach. Intell.*, vol. 4, no. 3, pp. 196–210, 2022.
[37] M. De Lange, R. Aljundi, M. Masana, S. Parisot, X. Jia, A. Leonardis, G. Slabaugh, and T. Tuytelaars, "A continual learning survey: Defying forgetting in classification tasks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 44, no. 7, pp. 3366–3385, Jul. 2022.
[38] G. Saha, I. Garg, and K. Roy, "Gradient projection memory for continual learning," in *Proc. Int. Conf. Learn. Represent.*, 2020.
[39] A. Abbasi, P. Nooralinejad, V. Braverman, H. Pirsiavash, and S. Kolouri, "Sparsity and heterogeneous dropout for continual learning in the null space of neural activations," 2022, *arXiv:2203.06514*.
[40] W. Wang, D. Slepčev, S. Basu, J. A. Ozolek, and G. K. Rohde, "A linear optimal transportation framework for quantifying and visualizing variations in sets of images," *Int. J. Comput. Vis.*, vol. 101, no. 2, pp. 254–269, Jan. 2013.
[41] S. Kolouri, N. Naderializadeh, G. K. Rohde, and H. Hoffmann, "Wasserstein embedding for graph learning," in *Proc. Int. Conf. Learn. Represent.*, 2021.

XINRAN LIU received the B.S. degree in mathematics and applied mathematics from Chongqing University, in 2019, and the M.S. degree in mathematics from Vanderbilt University, in 2021, where she is currently pursuing the Ph.D. degree in computer science. Her research interests include machine learning applications of computational optimal transport, especially in computer vision and signal processing.

YUZHE LU received the B.A. degree in mathematics and computer science from Vanderbilt University, in 2022. He is currently pursuing the master's degree in machine learning with Carnegie Mellon University. His research interests include machine learning, computer vision, natural language processing, and optimal transport.

ALI ABBASI received the B.Sc. degree in electrical engineering from the University of Tehran, in 2020. He is currently pursuing the Ph.D. degree in computer science with the MINT Laboratory, Vanderbilt University, Nashville, TN, USA. His research interests include continual learning and bio-inspired neural networks, exploring their capabilities to effectively memorize, and selectively forget their past experiences.

MEIYI LI (Graduate Student Member, IEEE) received the B.S. and M.S. degrees in electrical engineering from Shanghai Jiao Tong University, Shanghai, China, in 2017 and 2020, respectively, and the Ph.D. degree from Carnegie Mellon University, Pittsburgh, PA, USA, in 2021. She is currently pursuing the Ph.D. degree in electrical engineering with The University of Texas at Austin, Austin, TX, USA, with a focus on power and energy systems optimization.

Her research interests include learning to optimize and distributed optimization within the power and energy domain. She has received numerous accolades, including the Prestigious ‘‘Best of the Best’’ Paper Award at the 2019 IEEE Power and Energy Society General Meeting. Her impressive academic record includes the National Scholarship for Outstanding Academic Achievements, in 2018, the Carnegie Institute of Technology Dean’s Fellowship, in 2020, and the Distinguished Alumni Endowed Graduate Fellowship at The University of Texas at Austin, in 2022.

JAVAD MOHAMMADI (Senior Member, IEEE) received the Ph.D. degree from the Electrical and Computer Engineering Department, Carnegie Mellon University (CMU), in 2016.

He is currently an Assistant Professor with the Department of Civil, Architectural, and Environmental Engineering, The University of Texas at Austin. Before joining UT, he was a Faculty Member of the Electrical and Computer Engineering Department, CMU. His research interests include distributed decision-making in networked cyber-physical systems, including energy networks and electrified transportation systems. AFOSR, ARPA-E, and the Department of Energy support his grid modernization efforts. His research on building energy management has received financial support from local governments and institutions, such as the Sloan Foundation. As a Graduate Student, he was a recipient of the Innovation Fellowship from the Swartz Center for Entrepreneurship.

SOHEIL KOLOURI (Senior Member, IEEE) received the Ph.D. degree from the Biomedical Engineering Department, Carnegie Mellon University (CMU), in 2015. Currently, he is an Assistant Professor with the Department of Computer Science, Vanderbilt University, Nashville, TN, USA, where he leads the Machine Intelligence and Neural Technologies (MINT) Laboratory. Prior to joining Vanderbilt University, he was a Research Scientist with HRL Laboratories, LLC., Malibu, CA, USA, where he led multiple DARPA programs on AI in autonomy. His research interests include pattern recognition in medical images, applied mathematics, machine learning, and computer vision, with a specific focus on computational optimal transport and geometry. He is an Active Member of the Association for Computing Machinery (ACM). He was a recipient of the Bertucci Fellowship Award from the College of Engineering, in 2014, for outstanding graduate students, and the Outstanding Dissertation Award from the Biomedical Engineering Department, in 2015. His work has been recognized by numerous international awards, including the Best Paper Award at the 2022 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP).