A Curriculum Learning Approach to Optimization with Application to Downlink Beamforming

Jeremy Johnston, Xiao-Yang Liu, *Graduate Student Member, IEEE*, Shixun Wu, and Xiaodong Wang, *Fellow, IEEE*

***Abstract*—We investigate neural networks' ability to approximate the solution map of certain classes of beamforming optimization problems. The model is trained in an unsupervised manner to map a given channel realization to a near-optimal point of the corresponding optimization problem instance. Training is offline so that online optimization requires only the feedforward computation, the complexity of which is orders of magnitude less than state-of-the-art optimization algorithms. In order to obtain a near-optimal channel-beamformer mapping, either of two curriculum learning strategies is required: The *reward curriculum* employs a sequence of learning objectives of increasing complexity. The *subspace curriculum* employs a sequence of training data distributions restricting the data to linear subspaces of increasing dimension. For the MISO beamforming problem, the learned optimizer achieves near-optimal objective value (sum rate or minimum rate) across a wide range of signal-to-noise ratios. In the MIMO and relay scenarios, the learned optimizer is on par with and in some cases far exceeds performance of suboptimal beamforming strategies.**

***Index Terms*—Deep learning, downlink beamforming, nonconvex optimization, curriculum learning.**

I. Introduction

REAL-time applications in communications and control require an agent/controller to solve optimization problem instances generated by an environment over time. For example, in MIMO communications a base station must repeatedly update its transmit beamformers as the user channels vary, which occurs typically in intervals on the order of milliseconds. Therefore computational cost must play a central role in the algorithm design. In principle, there exists a deterministic mapping from the problem data space to the solution space; iterative convex optimization algorithms approximate this mapping to desired accuracy via recursive application of an analytically-derived operator. The thrust of *learning to optimize* (L2O) [1], [2], [3] and *amortized optimization* [4] is to replace such operator with a neural network and learn its parameters through data, with the goal of obtaining near-optimal solutions with much lower computational complexity.

Manuscript received 12 May 2023; revised 15 September 2023 and 1 November 2023; accepted 3 November 2023. Date of current version 6 December 2023. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Xiao Fu. *(Corresponding author: Jeremy Johnston.)*

Jeremy Johnston, Xiao-Yang Liu, and Xiaodong Wang are with the Department of Electrical Engineering, Columbia University, New York, NY 10027 USA (e-mail: j.johnston@columbia.edu; xl2427@columbia.edu; xw2008@columbia.edu).

Shixun Wu is with the Department of Computer Science and Engineering, University of California, Riverside, CA 92507 USA (e-mail: sw3511@columbia.edu).

Digital Object Identifier 10.1109/TSP.2023.3334396

A. Learning to Optimize and Amortized Optimization

The L2O/amortized optimization framework introduces notions of an *optimizee* and *optimizer*. The optimizee is an optimization objective family, each member of which is uniquely specified by problem data. The optimizer is a parametric function (e.g., neural network) to be trained to produce an approximate optimum for each instance of the optimizee. The optimizer may be trained in either a supervised or unsupervised fashion with a suitable loss function such as the mean-squared error if supervised, or the average optimizee value if unsupervised. Thus the cost of optimization is "amortized" over the training distribution, shifting the computational burden from online optimization to offline learning [4]. At the inference stage, feedforward computations of the learned optimizer are used to obtain an approximate solution to any sample instance of the optimizee, offering orders of magnitude speed-ups compared to state-of-the-art optimization algorithms.

As originally presented in [1] and [2], L2O was used for learning an optimizer for training neural networks. Their method adopts the algorithmic structure of vanilla gradient descent, but uses a recurrent neural network (RNN) to process the gradient and output the next step direction. For example, the optimizee could be the classification error of some neural network classifier, and the optimizer is an RNN trained such that it can optimize the classification error over the classifier's parameters. The optimizer's training objective is to minimize the expected cumulative classification (training set) error incurred over the optimizer's trajectory. Even though the optimizer may be trained for a classifier with a certain model architecture, the optimizer was shown to generalize to previously unseen model architectures.

See [3] and [4] for thorough surveys of prior work and applications. Towards a theoretical foundation, the statistical complexity of optimal solution mappings for linear and quadratic programs is analyzed in [5]. L2O-like methods can also be applied to combinatorial optimization problems where the decision variable belongs to a large but finite set. An optimizer may be trained to iteratively construct and/or modify a candidate decision until convergence to an optimum [6]. Learning-based methods can be also used to augment rather than supplant

conventional optimization methods. For example, in mixed integer linear programming, a learnable function aids conventional heuristics to determine which variables to branch at each iteration [7].

B. Curriculum Learning

Curriculum learning is an umbrella term for neural network training techniques that rely on the assumption that learning is more efficient when simple concepts are learned before more complex ones. The term was coined in [8] where "the basic idea is to start small, learn easier aspects of the task or easier sub-tasks, and then gradually increase the difficulty level." For example, a curriculum may involve modification of the training objective over the course of training, such that the objective becomes more complex as the training progresses. Similarly, training may begin with easier (e.g., noiseless) examples and proceed to harder (e.g. noisy) examples. In sequential learning tasks, one may gradually increase the sequence length; in L2O, for example, the optimizer trajectory length may be increased over the course of training [9].

C. Downlink Beamforming

In this paper, we consider three classes of optimization problems arising from *downlink beamforming*, a fundamental technology in multiuser wireless communication that allows simultaneous transmission of multiple data streams from a base station (BS) to multiple users using the same time-frequency resource [10], [11]. The BS is equipped with an antenna array whose complex amplitudes are to be configured so that the transmitted signals add constructively in certain spatial directions and destructively in others, thereby enabling spatial multiplexing of user data streams. A particular configuration of amplitudes is called a *beamformer*. Each beamformer has sidelobes that interfere with other users, therefore the beamformers should be optimized jointly so as to maximize a performance function that quantifies desired system behavior. For example, the minimum user rate is an objective that promotes fairness among users; to achieve the best overall system performance, we may consider the weighted sum of the user rates. Most functions of interest (e.g., sum rate), however, lead to optimization problems that cannot be solved in real time, thus suboptimal heuristic beamformers (e.g., based on mean-squared error criterion) prevail in practice [10].

D. Contribution and Outline

As far as we know, prior work has neither sought nor achieved sum rate optimal beamformers via deep learning; suboptimal heuristics are the only benchmarks considered therein. The major contributions of this work are as follows:

- We develop two curriculum learning techniques for learning to optimize beamformers: The *reward curriculum* prescribes a sequence of training objectives of increasing complexity, employing the mean-squared error (MSE) criterion. The *subspace curriculum* prescribes a sequence of training data distributions by restricting the training data to a linear subspace of increasing dimension.
- Our learning approach accommodates various beamforming problem scenarios with different optimization objectives. For MISO beamforming we consider sum rate and min rate maximization. In addition, we consider sum rate maximization for MIMO beamforming and relay beamforming.
- With the proposed curriculum learning techniques, the learned optimizers obtain nearly optimal beamformers for MISO sum rate and MISO min rate scenarios. For MIMO, our learned optimizer far outperforms the block diagonalization and MMSE methods, particularly in the overloaded case.
- For the relay scenario we jointly learn two optimizers mimicking block coordinate optimization. This divide-and-conquer strategy breaks the learning problem into two smaller subproblems which yields better performance than if we had attempted to learn a single mapping for the full problem. The learned optimizers are on par with baseline methods but require significantly less computation.

The remainder of the paper is organized as follows. First, Section II formalizes our learning approach. Section III presents two curriculum learning methods which are key to learning the optimal solution mapping. In Sections IV and V we introduce three beamforming scenarios, MISO, MIMO, and relay, to which we will apply the L2O approach. Of the three, an optimal solution is known only for the MISO case, therefore for MIMO and relay our goal is to achieve user performance as high as possible beyond the baseline heuristic methods. Finally, in Section VI we present simulation results and compare and contrast existing methods that employ deep learning for beamforming design.

II. Learning to Optimize

Consider a family of optimization problems

$$\mathcal{P} = \{\underset{x\in\mathbb{R}^n}{\text{maximize}}\ R(s,x) \mid s \in \mathbb{R}^m\}$$

where $R : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$ is the objective, $s \in \mathbb{R}^m$ is a parameter that specifies the problem instance and $x \in \mathbb{R}^n$ is the decision variable. We are interested in applications in which instances of problems in $\mathcal{P}$ must be rapidly solved in order to enable some real-time application, such as communication or control. In this regime, iterative algorithms are often preferred owing to their low complexity. Iterative optimization entails application of a sequence of functions $h_t : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^m$, $t = 1, 2, \ldots$, where each h_t maps s and a candidate point x_t to a new point $x_{t+1} := h_t(s, x_t)$ and is designed such that the sequence $\{x_t\}$ converges to a maximizer of $R(s, x)$ for all s. The recurrence is a composite mapping designed to approach or approximate a solution mapping,

$$x^\star(s) = \underset{x}{\text{argmax}}\ R(s,x). \tag{1}$$

In learning to optimize (L2O), the goal is to learn an operator $F_\theta : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}^n$, referred to as an *optimizer*, that approximates the optimal mapping (1) for all s, thereby embedding in

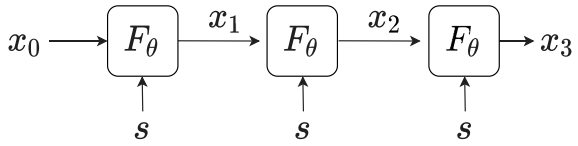


Fig. 1. Block diagram for learning iterative optimization. Shown here is an example with $T = 3$ iterations. At each iteration t, the learnable mapping F_θ is applied to the input (s, x_{t-1}) to produce x_t.

F_θ the set of all solutions of the problem family $\mathcal{P}$. Typically, F_θ is a neural network with learnable parameters θ. We seek to approximate the mapping (1) through recursive application of F_θ. Given s and a starting point x_0, we apply F_θ for T steps, yielding $x_t := F_\theta(s, x_{t-1}), t = 1, 2, \ldots, T$. A block diagram of this scheme is shown in Fig. 1 for $T = 3$ iterations. To measure the performance of a given trajectory $\{x_t\}$, we consider the sum of the objective value over the trajectory, $\sum_{t=1}^{T} R(s, x_t)$. If s has distribution p_s, we consider

$$J(\theta) := \mathbb{E}_{s\sim p_s}\left[\sum_{t=1}^{T} R(s, x_t)\right] \tag{2}$$

and the learning problem

$$\underset{\theta}{\text{maximize}}\ J(\theta)$$

to which we may apply gradient ascent. Training samples (i.e., problem instances specified by s that belong to $\mathcal{P}$) may be obtained through simulation or measurement. If p_s is known, then we may generate arbitrarily many samples. The starting point x_0 is chosen either heuristically or randomly for each s. Although the quantity of interest is the final objective value, $R(s, x_T)$, such a metric would ignore the intermediate values; summation over the entire trajectory, on the other hand, encourages each step to improve the objective.

Learning θ in effect "amortizes" the computational cost of optimization across all problem instances, shifting the computational burden from online optimization to offline learning [4]. In deployment of the learned model, an optimization problem is instantiated by s, then s is fed to the model which outputs a near-optimal solution for that problem instance. Since the model execution entails just feedforward computation of the model, optimization can be done repeatedly and rapidly.

The overall procedure is summarized in Algorithm 1. The target family $\mathcal{P}$ is specified by an objective function R with parameter $s \in \mathbb{R}^m$. In each epoch, the iteration $x_t = F_\theta(s, x_{t-1})$ is carried out for $t = 1, 2, \ldots, T$ and the cumulative objective value J is computed. Finally, a gradient step updates θ. At test time, for a given problem instantiated by s, we simply apply $x_t = F_\theta(s, x_{t-1})$ for $t = 1, 2, \ldots, T$ and return the final iterate x_T.

Analogous to the common practice of varying the hyperparameters of an iterative optimization algorithm as the iterations progress, the parameter θ may in general be allowed to vary with t. When θ is free to vary with respect to t, we refer to the optimizer as "untied"; the above presentation considers a "tied" optimizer [12]. Intuitively, the optimal parameters at $t = 1$ need not be the same as some later step, say $t = 5$, and furthermore, allowing the parameters to vary grants more expressive capacity. For an untied optimizer, Algorithm 1, we replace F_θ with a sequence of optimizers $F_{\theta_t}, t = 1, \ldots, T$, hence step 15 becomes $x_t^{(i)} = F_{\theta_t}(s^{(i)}, x_{t-1}^{(i)})$.

Algorithm 1: Learning to Optimize

1 **Input**:
2 R, objective function
3 s, problem instance parameter
4 x, decision variable
5 T, number of steps
6 η, learning rate
7 N_b, batch size.
8 F_θ, learnable optimizer
9 **Train**:
10 **for** epoch $l = 1, 2, \ldots$ **do**
11 Obtain batch of samples $\{s^{(i)} \mid i = 1, \ldots, N_\mathrm{b}\}$
12 **for** $i = 1, \ldots, N_\mathrm{b}$ **do**
13 Choose $x_0^{(i)}$ (heuristically or randomly)
14 **for** $t = 1, 2, \ldots, T$ **do**
15 $x_t^{(i)} = F_\theta(s^{(i)}, x_{t-1}^{(i)})$
16 $J(\theta) = \sum_i \sum_{t=1}^{T} R(s^{(i)}, x_t^{(i)})$
17 $\theta \leftarrow \theta + \eta \nabla_\theta J(\theta)$
18 **Evaluate** on test set $\mathcal{S}$:
19 **for** $s \in \mathcal{S}$ **do**
20 Choose x_0
21 **for** $t = 1, 2, \ldots, T$ **do**
22 $x_t = F_\theta(s, x_{t-1})$
23 Output x_T

A. Block Coordinate Optimization

L2O can be extended to problems where block coordinate optimization is appropriate. The block coordinate method splits the optimization variable to subsets and iteratively optimizes over each subset while holding the others fixed. If $x = (y, z)$ is a partition of the optimization variable, the subproblems at iteration t have the form

$$y_t = \underset{y}{\text{argmax}}\ R(s, y, z_{t-1}) \tag{3}$$

$$z_t = \underset{z}{\text{argmax}}\ R(s, y_t, z). \tag{4}$$

This approach is employed by ADMM [13] to efficiently solve convex problems. Even if R is nonconvex, the subproblems may be tractable or admit closed-form solutions, thus providing an efficient means of finding a local optimum.

Inspired by block coordinate optimization, we may learn two optimizers to learn the optimal mappings in (3) and (4). The rationale is twofold: a nonconvex objective may become simplified when certain coordinates are held constant; and splitting the optimization variable reduces the dimension of the output space of each optimizer which, by mitigating the curse of dimensionality, reduces the desired mapping's complexity.

Algorithm 2: Learning a Block Coordinate Optimizer

Input:
  R, objective function
  s, problem instance parameter
  $x = (y, z)$, decision variable
  T, number of steps
  η, learning rate
  N_b, batch size.
  F_θ, G_ϕ, learnable optimizers
Train:
for epoch $l = 1, 2, \ldots$ **do**
  Obtain batch of samples $\{s^{(i)} \mid i = 1, \ldots, N_\mathrm{b}\}$
  for $i = 1, \ldots, N_\mathrm{b}$ **do**
    Choose $y_0^{(i)}, z_0^{(i)}$
    for $t = 1, \ldots, T$ **do**
      $y_t^{(i)} = F_\theta(s, y_{t-1}^{(i)}, z_{t-1}^{(i)})$
      $z_t^{(i)} = G_\phi(s, y_t^{(i)}, z_{t-1}^{(i)})$
    end
  end
  $J(\theta, \phi) = \sum_i \sum_{t=1}^{T} R(s^{(i)}, y_t^{(i)}, z_t^{(i)})$
  if $l \mod 2 = 0$ **then**
    $\theta \leftarrow \theta + \eta \nabla_\theta J(\theta, \phi)$
  else
    $\phi \leftarrow \phi + \eta \nabla_\phi J(\theta, \phi)$
  end
end
Evaluate on test set $\mathcal{S}$:
for $s \in \mathcal{S}$ **do**
  Choose y_0, z_0
  for $t = 1, \ldots, T$ **do**
    $y_t = F_\theta(s, y_{t-1}, z_{t-1})$
    $z_t = G_\phi(s, y_t, z_{t-1})$
  end
  Output $x = (y_T, z_T)$
end

The procedure is summarized in Algorithm 2. Let F_θ and G_ϕ denote optimizers which output candidate points y and z, respectively. We consider the following alternating scheme that iteratively computes y_t and z_t for $t = 1, \ldots, T$:

$$y_t = F_\theta(s, y_{t-1}, z_{t-1}) \tag{5}$$
$$z_t = G_\phi(s, y_t, z_{t-1}). \tag{6}$$

A block diagram of this scheme is shown in Fig. 2 for $T = 3$ iterations. As in the single-variable case, we consider the average sum of the objective values obtained over the iterations,

$$J(\theta, \phi) := \mathbb{E}_{s \sim p_s} \left[\sum_{t=1}^{T} R(s, y_t, z_t) \right]. \tag{7}$$

We employ (7) as a training objective function in order to learn the optimizer parameters θ and ϕ, which may be optimized via block coordinate optimization as well. That is, at epoch 1 we perform a gradient step for ϕ, at epoch 2 we update θ, at epoch 3 we update ϕ, and so on. To obtain a warm start for each optimizer, F_θ (or G_ϕ) may be separately pretrained via (3) (or (4)) by fixing z (or y).

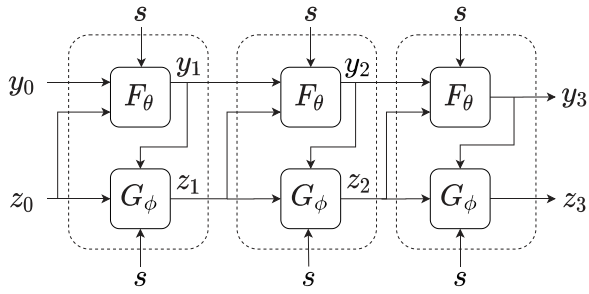


Fig. 2. Block diagram for learning block coordinate optimization. Shown here is an example with $T = 3$ iterations. At each iteration t, the learnable mapping F_θ is applied to the input (s, y_{t-1}, z_{t-1}) to obtain y_t and G_ϕ is applied to the input (s, y_t, z_{t-1}) to obtain z_t.

B. Iterative Optimization as a Markov Decision Process

Iterative optimization may be viewed as a Markov Decision Process [2] where the policy is F_θ, the state at iteration $t = 0, 1, \ldots$ is the pair (s, x_t), the action is $x_{t+1} = F_\theta(s, x_t)$, and the reward for action x is $R(s, x)$. Reinforcement learning (RL) [14] seeks to maximize with respect to the policy parameter θ the discounted cumulative episode reward $\mathbb{E}\left[\sum_{t=1}^{\infty} \gamma^t R(s, x_t)\right]$, $0 < \gamma < 1$, where expectation is over the start state (s, x_0) and all possible trajectories. In the case where the policy, transition dynamics, and x_0 are all deterministic, every trajectory is determined by the start state, so the RL objective becomes $\mathbb{E}_{s \sim p_s}\left[\sum_{t=1}^{\infty} \gamma^t R(s, x_t)\right]$ where p_s is the start state distribution. Setting $\gamma = 1$ and keeping only the first T terms of the summation yields the aforementioned training objective J. In principle, off-the-shelf RL algorithms may be applied in L2O [2]. However, in L2O we assume that the reward function R is perfectly known, which is generally not the case in RL; indeed, much of the effort in the development of RL algorithms goes toward approximating the value function since the reward is either unknown or computationally costly to compute. Similarly, there is a correspondence between block coordinate optimizers and multi-agent reinforcement learning (MARL) [15]. In the foregoing presentation, we may view F_θ and G_ϕ as the policies of two agents acting in an environment. Equation (7) corresponds to the expected cumulative reward of the agents' actions.

C. Relation To Existing Methods

Several optimization-inspired deep learning methods can be obtained as particular cases of the above framework, depending on the architecture of the optimizer network (F_θ), the training objective function ($J(\theta)$), and the particular training algorithm. *Deep unfolding* [16] is strategy for designing F_θ drawing upon pre-existing iterative optimization algorithms for inspiration. A given optimization algorithm is viewed as a function whose learnable parameters correspond to the tunable hyperparameters of the given algorithm; for example, when the target optimization family is ℓ_1-regularized linear regression, ADMM [17] and ISTA [18] consist of a sequence of linear and nonlinear operations and thus are readily converted to neural networks.

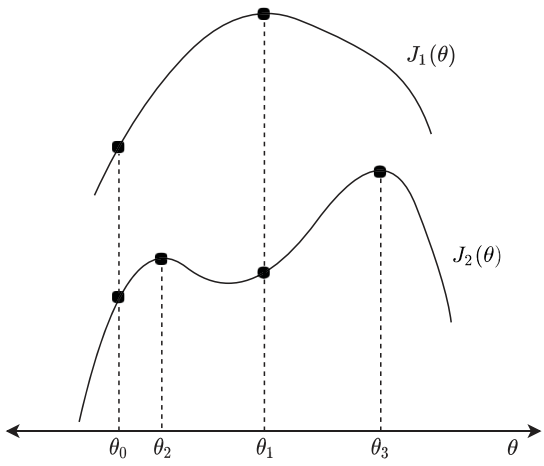


Fig. 3. Reward curriculum intuition. The primary task has loss surface J_2 and the subordinate task has J_1. Starting from initial point θ_0, training with J_1 converges to local optimum θ_1; training with J_2 converges to θ_2. Starting from θ_1, training continues with J_2 and converges to θ_3, a local optimum better than θ_2.

A related set of approaches exploits the algorithmic structure of gradient descent [1], [2]. Here the optimizer has the form $F_\theta(s, x_t, H_t) := x_t + G_\theta(s, H_t)$, where here G_θ is a learnable function that outputs a step direction based on a history H_t containing previous values of x_t and gradients $\nabla_x R(s, x)|_{x=x_t}$. Although this method exploits the structure of vanilla gradient descent, the architecture of $G_\theta(s, H_t)$ is a generic network architecture (e.g. MLP) and thus imposes a weaker prior than typical of deep unfolding. Furthermore, the trajectory of x_t may be viewed as an MDP with state H_t and the optimizer F_θ is trained via deep reinforcement learning algorithms [2].

From a theoretical perspective, the universal approximation theorem [19] suggests there exists a single-layer neural network can approximate certain iterative algorithms arbitrarily well. Neural networks can approximate multiplication and division arbitrarily well, therefore a neural network can in principle approximate arbitrarily well any algorithm that is a composition of multiplications and divisions [20]. In special cases it can be shown, again by invoking the universal approximation theorem, that a neural network can achieve zero duality gap as the network size approaches infinity [21]. A more precise characterization of L2O must overcome the inherent difficulties of neural network analysis and is an active area of research [4].

III. Curriculum Learning

The ethos of curriculum learning is to train a model on a sequence of tasks of increasing difficulty. Each task is defined by a particular training objective function and a particular training data distribution. The sequence of tasks should increase in difficulty [8], in that the loss functions should become successively more complex, and the entropy of the data distributions should increase as the curriculum progresses. Doing so may encourage exploration and act as a sort of regularization, as illustrated in Fig. 3. Rather than solely learn the primary task we first learn an easier subordinate task, and resuming training from such point, training may converge to a point in parameter space unreachable had we trained solely on the primary task.

Most curriculum learning strategies are based on intuition or justified by analogy to the way humans learn. There have been attempts to precisely define and rigorously analyze the concept in simplified settings, for instance in the case of convex loss functions [22]. The import of such analyses is questionable since neural network loss functions have many local minima. It is an open question whether we can rigorously characterize the mechanism by which a given curriculum allows the network to avoid traps and reach favorable regions of parameter space.

We found that a straightforward implementation of Algorithm 1 may be insufficient to learn the optimal mapping. Curriculum learning techniques are required. We propose two curricula that can be used within Algorithms 1 and 2. The first, subspace curriculum, uses a fixed training objective and prescribes a sequence of training data distributions of increasing complexity. The second, reward curriculum, uses a fixed training distribution and prescribes a sequence of training objectives of increasing complexity.

A. Subspace Curriculum

This curriculum curates the training data (problem instances) seen by the optimizer over the course of training. At each stage we sample from distributions of increasing entropy and therefore learn tasks of increasing difficulty [8, §3]. Observe that the distribution p_s affects the difficulty of maximizing (2). For a zero-entropy distribution $p_s(s) = \delta(s - s_0)$ where $s_0 \in \mathbb{R}^m$ is known, we only have to learn a single output, namely $\operatorname{argmax}_x R(s_0, x)$.

The subspace curriculum prescribes that during each stage of the curriculum the training data is restricted to a linear subspace of the state space $\mathbb{R}^m$. That is, we train the optimizer on a sequence of tasks corresponding to the problem families

$$\mathcal{P}_d = \{\underset{x \in \mathbb{R}^m}{\text{maximize}}\ R(s, x) \mid s \in S_d\}$$

for $d = 1, 2, \ldots, m$ where $S_d \subseteq \mathbb{R}^m$ is a d-dimensional subspace. Since $S_1 \subset S_2 \subset \cdots \subset \mathbb{R}^m$, we have $\mathcal{P}_1 \subset \mathcal{P}_2 \subset \cdots \subset \mathcal{P}_m$, so intuitively the complexity of the problem family increases with d. The entropy of the distribution increases with d, since the dimension of the sample space increases with d; for example, with $p_{\alpha,d} = \mathcal{N}(0, \boldsymbol{I})$, the entropy is given by $d(1 + \log 2\pi)/2 + (\log d)/2$, which is increasing in d. The subspaces are generated via a particular orthonormal basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_m\} \subseteq \mathbb{R}^m$ chosen prior to training, such that $S_d := \operatorname{span}(\{\mathbf{b}_1, \ldots, \mathbf{b}_d\})$. If the subspace dimension is d, training samples are generated via $s = \sum_{i=1}^{d} \alpha_{i,d} \mathbf{b}_i$, where the coefficients $\alpha_{i,d} \in \mathbb{R}$ are sampled from a chosen distribution $p_{\alpha,d}$.

Algorithm 3 demonstrates the application of the subspace curriculum in a general learning environment. We are given a training objective $J(\theta)$ where θ denotes the learnable model parameters. Before training begins, we generate a random orthonormal basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_m\}$ which induces the subspaces of $\mathbb{R}^m$ in which training samples will reside. We initialize the subspace dimension $d = 1$ and we increment the subspace dimension every N epochs. Thus, for the first N epochs all

Algorithm 3: Subspace Curriculum

Input:
  Δd: subspace dimension increment
  N: number of epochs in each stage of curriculum
  $\{\mathbf{b}_1, \dots, \mathbf{b}_n\}$: orthonormal basis
  $\{p_{\alpha,d} \mid d = 1, \dots, m\}$: coefficients distributions
  N_{b}: batch size
  $J(\theta)$: training objective
Initialize $d = 1$
for epoch $l = 1, 2, \dots$ **do**
  Generate batch $\{s^{(i)} = \sum_{k=1}^{d} \alpha_{k,d}^{(i)} \mathbf{b}_k \mid \alpha_{k,d}^{(i)} \sim p_{\alpha,d}, i = 1, \dots, N_{\mathrm{b}}\}$
  Compute $J(\theta)$ and update θ
  if $d < n$ and $l = 0 \mod N$ **then**
    update $d \leftarrow d + \Delta d$

training samples lie on the line induced by basis vector $\mathbf{b}_1$, i.e., the samples are generated according to $s = \alpha_{1,1}\mathbf{b}_1$ where $\alpha_{1,1} \sim p_{\alpha,1}$. After N epochs, we increase the subspace dimension to $d = 2$, so that in the following N epochs training samples are generated via $s = \alpha_{1,2}\mathbf{b}_1 + \alpha_{2,2}\mathbf{b}_2$ where $\alpha_{1,2}, \alpha_{2,2} \sim p_{\alpha,2}$. After $N(m-1)$ epochs the subspace dimension is $d = m$, at which point the training samples span $\mathbb{R}^m$. Since the subspace curriculum affects only the training data, it can in principle be incorporated into various learning algorithms. For example, in Algorithm 1, one would only need to modify the training sample generation (step 11).

B. Reward Curriculum

Suppose we have two objective functions $R_1 : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$ and $R_2 : \mathbb{R}^m \times \mathbb{R}^n \to \mathbb{R}$ corresponding to two distinct tasks such that the point $\operatorname{argmax}_x R_1(s, x)$ obtains a reasonably good objective value R_2 for all s. We may first learn θ for the family of problems

$$\mathcal{P}_1 = \{\underset{x \in \mathbb{R}^m}{\text{maximize}}\ R_1(s, x) \mid s \in \mathbb{R}^m\},$$

using the training objective $J_1(\theta) := \mathbb{E}_{s \sim p_s}\left[\sum_{t=1}^{T} R_1(s, x_t)\right]$. If the optimizer converges to a point θ_1, then starting from θ_1 we continue training with primary task objective $J_2(\theta) := \mathbb{E}_{s \sim p_s}\left[\sum_{t=1}^{T} R_2(s, x_t)\right]$ until convergence. Generally, θ_1 will be suboptimal with respect to $\mathcal{P}_2$, but nonetheless may provide a warm start that ultimately leads to a point superior to that which would be obtained via training solely with J_2. Moreover, if R_1 is a relatively simple function (e.g., quadratic in x), then it stands to reason that the mapping $\operatorname{argmax}_x R_1(s, x)$ will be simpler and θ will converge quickly. This intuition is illustrated in Fig. 3 and supported by the experimental results in Section VI. The idea of using a sequence of objective functions for nonconvex optimization traces back to *continuation methods*: a nonsmooth objective is replaced by a surrogate objective that is relatively smooth (i.e., easier to optimize) and parameterized by a scalar such that, as the scalar increases, the surrogate becomes less smooth and converges to the desired objective [8].

Algorithm 4: Reward Curriculum

Input:
  $\{J_k\}_{k=1}^{K_c}$: task objective functions
  θ: model parameters
  N: number of epochs in each stage of curriculum
Initialize $k = 1$
for epoch $l = 1, 2, \dots$ **do**
  Generate batch $\{s^{(i)}\}$
  Compute $J_k(\theta)$ and update θ
  if $k < K_c$ and $l = 0 \mod N$ **then**
    update $k \leftarrow k + 1$

TABLE I
SYMBOL CORRESPONDENCE BETWEEN SECTION II AND VARIOUS BEAMFORMING SCENARIOS

	MISO	MIMO	Relay
s	$\mathbf{H}$	$\{\mathbf{H}_k\}$	$(\mathbf{H}, \mathbf{G})$
x	$\mathbf{W}$	$\{\mathbf{W}_k\}$	$(\mathbf{F}, \mathbf{W})$
R	$R(\mathbf{H}, \mathbf{W})$	$R(\{\mathbf{H}_k\}, \{\mathbf{W}_k\})$	$R(\mathbf{H}, \mathbf{G}, \mathbf{F}, \mathbf{W})$

Algorithm 4 demonstrates the application of a reward curriculum in a general learning environment. It is assumed that there are K_c task objective functions $\{J_k\}_{k=1}^{K_c}$ such that the task difficulty increases with k and the desired task is represented by the final objective function J_{K_c}. Training begins with $k = 1$ and thus J_1 serves as the training objective. In each epoch, we obtain a batch of samples and then perform a gradient update using J_1. After N epochs, the task index is incremented to $k = 2$ and J_2 is used as the training objective function. After $N(K_c - 1)$ epochs we have $k = K_c$ and the final task objective J_{K_c} is used for the remainder of training. The reward curriculum is readily applied to Algorithm 1; the training objective in step 16 is modified according to the curriculum over the course of training.

Next we apply the proposed methods to three beamforming scenarios: MISO, MIMO and relay. This requires four main steps: (1) design the optimizer neural network architecture, (2) define a problem family with objective function R corresponding to a system performance criterion, (3) formulate an iterative scheme that outputs beamformers for given channels where the iterative operator is a learnable optimizer, (4) devise a training procedure to learn the optimizer parameters such that the iterative scheme approaches the optimal channel-beamformer mapping. For reference, Table I contains the symbol mapping for each application scenario.

IV. Application to MISO Downlink Beamforming

A. System Model

Consider an N-antenna BS that communicates with K single-antenna users. The BS applies transmit beamformer $\mathbf{w}_i \in \mathbb{C}^N$ to the ith user data stream, so that, if $x_i \in \mathbb{C}$ is the symbol intended for user i, the transmitted signal is $\sum_{i=1}^{K} x_i \mathbf{w}_i$. Let $\mathbf{h}_k$ denote the channel between the BS and user k, so that

user k receives the signal

$$y_k = \mathbf{h}_k^{\mathsf{H}} \sum_{i=1}^{K} x_i \mathbf{w}_i + n_k, \ k = 1, \ldots, K \tag{8}$$

where $n_k \sim \mathcal{CN}(0, \sigma^2)$ is additive noise. User k's signal-to-interference-plus-noise ratio is

$$\mathrm{SINR}_k = \frac{|\mathbf{h}_k^{\mathsf{H}} \mathbf{w}_k|^2}{\sigma^2 + \sum_{i \neq k} |\mathbf{h}_k^{\mathsf{H}} \mathbf{w}_i|^2}.$$

The MISO beamforming problem is formulated as

$$\begin{aligned} \underset{\mathbf{w}_1, \ldots, \mathbf{w}_K}{\text{maximize}} \quad & f(r_1, \ldots, r_K) \\ \text{subject to} \quad & \sum_{k=1}^{K} \|\mathbf{w}_k\|_2^2 \leq P \end{aligned} \tag{9}$$

where $f : \mathbb{R}^K \to \mathbb{R}$ is the system performance function and $r_k := \log_2(1 + \mathrm{SINR}_k)$ is the *achievable information rate* of user k. For many common choices of f, (9) can be solved via specialized algorithms. However, the computational complexity is often prohibitive; in practice sub-optimal linear beamformers prevail.

1) Sum-Rate Maximization: For sum-rate maximization we define $f(r_1, \ldots, r_K) := \sum_{k=1}^{K} r_k$. In this case (9) is nonconcave and has many local maxima, hence general purpose solvers are not guaranteed to obtain the global optimum. Nonetheless, a global optimum can be found via the branch-reduce-bound (BRB) algorithm [23]. BRB iteratively refines a set of bounding boxes that contain the Pareto frontier. At each iteration the bounds are improved by solving a sequence of convex programs. Arbitrary small solution error can be achieved in a finite number of iterations, at the cost of solving an exponential (in K) number of convex programs.

2) Min-Rate Maximization: For min-rate maximization we define $f(r_1, \ldots, r_K) := \min_k r_k$. In this case (9) is equivalent to

$$\begin{aligned} \underset{\mathbf{w}_1, \ldots, \mathbf{w}_K, r}{\text{maximize}} \quad & r \\ \text{subject to} \quad & r_k \geq r, \ \forall k \\ & \sum_{k=1}^{K} \|\mathbf{w}_k\|_2^2 \leq P, \end{aligned} \tag{10}$$

which is quasiconvex and solvable via the bisection method [23, Theorem 2.10].

3) Linear MMSE Beamformers: The well-known linear MMSE beamformers [10] are given by

$$\mathbf{w}_k = \sqrt{p_k} \frac{(\sigma^2 \mathbf{I}_N + \sum_{i=1}^{K} \frac{P}{K} \mathbf{h}_i \mathbf{h}_i^{\mathsf{H}})^{-1} \mathbf{h}_k}{\left\| (\sigma^2 \mathbf{I}_N + \sum_{i=1}^{K} \frac{P}{K} \mathbf{h}_i \mathbf{h}_i^{\mathsf{H}})^{-1} \mathbf{h}_k \right\|_2} \tag{11}$$

where $\{p_k\}$ are transmit powers. We assume equal power allocation, $p_k = P/K$.

B. Proposed Learning Algorithm

We apply Algorithm 1 as follows. Let $s = [\mathbf{h}_1 \ \cdots \ \mathbf{h}_K]^{\mathsf{H}} := \mathbf{H} \in \mathbb{C}^{K \times N}$ and $x = [\mathbf{w}_1 \ \cdots \ \mathbf{w}_K] := \mathbf{W} \in \mathbb{C}^{N \times K}$. The optimization problem class is

$$\mathcal{P} = \left\{ \begin{array}{l} \underset{\mathbf{W} \in \mathbb{C}^{N \times K}}{\text{maximize}} \ R(\mathbf{H}, \mathbf{W}) \\ \|\mathbf{W}\|_F^2 = P \end{array} \right\},$$

where $R : \mathbb{C}^{N \times K} \times \mathbb{C}^{N \times K} \to \mathbb{R}$ is the performance criterion (f in (9)). The learnable optimizer is denoted $F_\theta : \mathbb{C}^{N \times K} \times \mathbb{C}^{N \times K} \to \mathbb{C}^{N \times K}$ with learnable parameters θ. The beamformers are iteratively computed via

$$\mathbf{W}_{t+1} := F_\theta(\mathbf{H}, \mathbf{W}_t). \tag{12}$$

and the training objective is

$$J(\theta) = \mathbb{E}_{\mathbf{H} \sim p_{\mathbf{H}}} \left[\sum_{t=1}^{T} R(\mathbf{H}, \mathbf{W}_t) \right] \tag{13}$$

where $p_{\mathbf{H}}$ is the channel distribution. Algorithm 1 essentially performs stochastic gradient ascent with objective $J(\theta)$. In each epoch, a batch of channels is generated with SNR $\sigma_{\mathbf{H}}^2$ (the SNR cam be made to vary from sample to sample so that the optimizer is trained on a range of SNRs; see Section VI for details). For each sample in the batch, we carry out the iteration (12) for T steps, compute the corresponding cumulative loss (13) and perform a gradient step for θ.

As previously mentioned, we found that Algorithm 1 by itself is insufficient. At least one of the following curriculum learning techniques is required in order to obtain near-optimal beamformers.

1) Reward Curriculum Learning: The MMSE problem is well-suited for the role of a subordinate task in a reward curriculum: the MMSE beamformers achieve a reasonably good sum rate and min rate, and the MSE objective is simply quadratic in $\mathbf{W}$. The MSE is defined as

$$\mathrm{MSE}(\mathbf{H}, \mathbf{W}) := \mathbb{E}\left[\|\mathbf{H}\mathbf{W}\mathbf{x} + \mathbf{n} - \mathbf{x}\|_2^2 \right], \tag{14}$$

where expectation is with respect to the noise $\mathbf{n} \sim \mathcal{CN}(0, \sigma^2 \mathbf{I})$ and data vector $\mathbf{x} \sim \mathcal{CN}(0, \mathbf{I})$. The MMSE optimization problem is

$$\begin{array}{l} \underset{\mathbf{W} \in \mathbb{C}^{N \times K}}{\text{minimize}} \ \mathrm{MSE}(\mathbf{H}, \mathbf{W}). \\ \|\mathbf{W}\|_F^2 = P \end{array}$$

To compute the MSE, we may either empirically evaluate the objective via sampling i.i.d. vectors $\mathbf{x}$ and $\mathbf{n}$, or we may use the analytical expression $\mathrm{MSE}(\mathbf{H}, \mathbf{W}) = \|\mathbf{H}\mathbf{W}\|_F^2 - 2\mathrm{Re}\{\mathrm{trace}(\mathbf{H}\mathbf{W})\}$. In terms of Algorithm 4, our proposed reward curriculum defines the task loss $R_1 := \mathrm{MSE}$ and R_2 is the desired performance criterion (e.g., sum rate, min rate).

2) Subspace Curriculum Learning: Assuming $p_{\mathbf{H}} \sim \mathcal{CN}(0, 1)$, we can generate arbitrarily many training samples and curate the training data according to the subspace curriculum. To implement subspace curriculum learning, we must specify the orthonormal basis $\{\mathbf{b}_1, \ldots, \mathbf{b}_n\}$ of $\mathbb{R}^n$ and the distributions $\{p_{\alpha,d} \mid d = 1, \ldots, m\}$ which are used to generate samples in a given subspace. For the MISO problem, the full channel space is $\mathbb{R}^{2NK}$, hence $n = 2NK$. The basis vectors are randomly generated. We set $p_{\alpha,d} = \mathcal{CN}(0, 1)$ for all d.

C. Toy Example

Each beamforming method can be written as a function

$$\mathbf{W}_f^\star(\mathbf{H}) := \underset{\mathbf{W}}{\mathrm{argmax}} \, f(\mathbf{W}, \mathbf{H}),$$

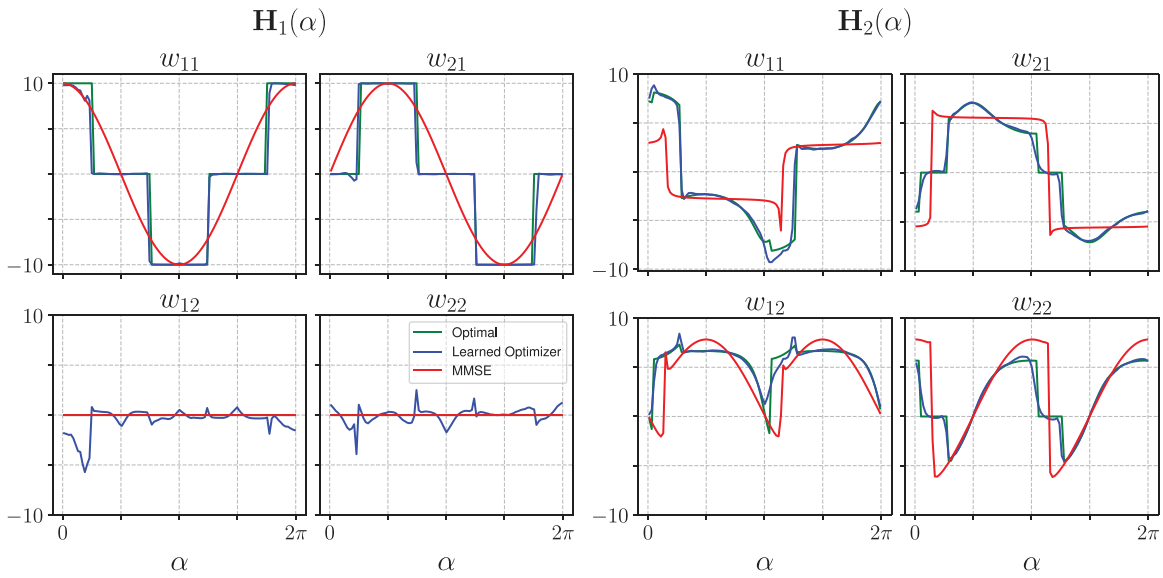


Fig. 4. Solution mapping for optimal (green), learned optimizer (blue), and MMSE (red) beamformers. On the left, the user channels are 2-dimensional, on the right they are 3-dimensional. The optimal solution map for the 2-dimensional channels is relatively more complex than that of the 3-dimensional channels. Also, the MMSE solution map is a smoothed version of the optimal solution map. Since the solution map is the function we wish to approximate, these observations suggest that the subspace and reward curricula reduce the difficulty of the learning task. The learned optimizer approximates the optimal map.

referred to as the *solution map* corresponding to the objective f. For sum-rate maximization, f is the sum rate; for linear MMSE, f is the negative MSE. Our learning approach in essence attempts to approximate the mapping $\mathbf{W}_f^\star(\mathbf{H})$ for a specified f. Therefore the difficulty of the learning task is tied to the complexity of $\mathbf{W}_f^\star(\mathbf{H})$.

The following toy example is meant to elucidate how the two proposed curriculum strategies (reward and subspace) deform the target solution map $\mathbf{W}_f^\star(\mathbf{H})$. To facilitate visualization, we consider the case $N = K = 2$, so that $\mathbf{W}_f^\star(\mathbf{H}) \in \mathbb{R}^{2\times 2}$, and channels of the form

$$\mathbf{H}_1(\alpha) = \begin{bmatrix} \cos\alpha & 0 \\ \sin\alpha & 0 \end{bmatrix} \in \mathbb{R}^{2\times 2},\ \mathbf{H}_2(\alpha) = \begin{bmatrix} \cos\alpha & 1 \\ \sin\alpha & 0.5 \end{bmatrix} \in \mathbb{R}^{2\times 2}$$

parameterized by $\alpha \in [0, 2\pi]$. It can be verified that the set $\{\mathrm{vec}(\mathbf{H}_1(\alpha)) : \alpha \in [0, 2\pi]\}$ spans a 2-dimensional subspace of $\mathbb{R}^4$, while $\{\mathrm{vec}(\mathbf{H}_2(\alpha)) : \alpha \in [0, 2\pi]\}$ spans a 3-dimensional subspace of $\mathbb{R}^4$. In Fig. 4, each set of plots shows the entries of $\mathbf{W}_f^\star(\mathbf{H}_i(\alpha))$, $i = 1, 2$, $\alpha \in [0, 2\pi]$, for the cases: (a) f is the sum rate (green), computed via the optimal BRB algorithm, (b) when f is the negative MSE (red) given by equation (11) in the manuscript. Also shown are the beamformer entries output by a neural network trained (via the proposed framework) to approximate the sum rate solution map. We make the following observations:

- *The MSE solution map is simpler than the sum rate solution map:* In all plots, the red curve is a smoothed version of the green curves. This suggests that the $\mathbf{W}_f^\star(\mathbf{H})$ is simpler when f is the MSE, relative to when f is the sum rate. We conclude that learning the MMSE solution map will be easier than learning the sum rate solution map.
- *Solution map complexity increases with channel subspace dimension:* The solution maps for the channels $\mathbf{H}_1(\alpha)$ are qualitatively smoother than those for $\mathbf{H}_2(\alpha)$. This suggests that the training objective becomes simpler when the channels lie on a low-dimensional subspace. As the subspace dimension increases, the desired solution map becomes more complex.

V. Application to MIMO and Relay Beamforming

A. MIMO Beamforming

In the MIMO case, each user is equipped with a receive antenna array capable of receive beamforming, or spatially filtering the impinging waveform, and hence may perform additional interference cancellation, thereby relaxing the transmit beamforming requirements. Suppose the BS has N transmit antennas and user k has m_k receive antennas. The symbol $\mathbf{x}_k \in \mathbb{C}^{m_k}$ is intended for user k and the BS applies the beamformers $\mathbf{W}_k \in \mathbb{C}^{N\times m_k}$, so that the transmitted signal is $\sum_{k=1}^K \mathbf{W}_k\mathbf{x}_k$. If $\mathbf{H}_k \in \mathbb{C}^{N\times m_k}$ is user k's channel matrix, then user k's array measurement $\mathbf{y}_k = [y_{k,1}\ \cdots\ y_{k,m_k}]^\mathsf{T} \in \mathbb{C}^{m_k}$ has the form $\mathbf{y}_k = \mathbf{H}_k^\mathsf{H} \sum_{i=1}^K \mathbf{W}_i\mathbf{x}_i + \mathbf{n}_k$, or

$$\mathbf{y}_k = \mathbf{H}_k^\mathsf{H}\mathbf{W}_k\mathbf{x}_k + \mathbf{H}_k^\mathsf{H}\sum_{i\neq k}^K \mathbf{W}_i\mathbf{x}_i + \mathbf{n}_k \tag{15}$$

Define $\mathbf{\Sigma}_k \in \mathbb{C}^{m_k\times m_k}$, the covariance matrix of the interference-plus-noise term,

$$\mathbf{\Sigma}_k = \sigma^2\mathbf{I} + \mathbf{H}_k^\mathsf{H}\left(\sum_{i\neq k}^K \mathbf{W}_i\mathbf{W}_i^\mathsf{H}\right)\mathbf{H}_k. \tag{16}$$

Then the rate of user k is

$$r_k = \log_2\left|\mathbf{I} + \mathbf{\Sigma}_k^{-1}\mathbf{H}_k^\mathsf{H}\mathbf{W}_k\mathbf{W}_k^\mathsf{H}\mathbf{H}_k\right| \tag{17}$$

and the beamformer design problem becomes

$$\begin{array}{ll}\underset{\mathbf{W}_1,\ldots,\mathbf{W}_K}{\text{maximize}} & f(r_1,\ldots,r_K)\\ \text{subject to} & \sum_{i=1}^{K}\|\mathbf{W}_i\|_F^2 \le P.\end{array} \tag{18}$$

(18) is nonconvex and no known method guarantees a solution.

The *block diagonalization beamforming (BDBF)* [24] method uses the singular value decomposition to choose transmit beamformers that lie in the null space of the matrix

$$\tilde{\mathbf{H}}_k^{\mathsf{T}} \triangleq [\mathbf{H}_1 \ \cdots \mathbf{H}_{k-1} \ \mathbf{H}_{k+1} \ \cdots \ \mathbf{H}_K]^{\mathsf{T}} \in \mathbb{C}^{(m-m_k)\times N},$$

thereby guaranteeing zero inter-user interference. We refer the reader to [24, §4.1] for more details.

1) Proposed Learning Algorithm: The class of resource allocation problems under consideration is

$$\mathcal{P} = \left\{ \begin{array}{ll}\underset{\mathbf{W}_k\in\mathbb{C}^{N\times m_k}}{\text{maximize}} & R(\{\mathbf{H}_k\},\{\mathbf{W}_k\})\\ \sum_k \|\mathbf{W}_k\|_F^2 = P\end{array} \right\},$$

where $R : \mathbb{C}^{N\times(m_1+m_2+\cdots+m_K)} \times \mathbb{C}^{N\times(m_1+m_2+\cdots+m_K)} \to \mathbb{R}$ is the performance criterion and $\mathbf{H}_k \in \mathbb{C}^{N\times m_k}$. In this case we may straightforwardly apply Algorithm 1 by defining $s := (\mathbf{H}_1,\ldots,\mathbf{H}_K)$, $x := (\mathbf{W}_1,\ldots,\mathbf{W}_K)$. We consider only the sum rate

$$R(\{\mathbf{H}_k\},\{\mathbf{W}_k\}) = \sum_{k=1}^{K}\log_2\left|\mathbf{I} + \mathbf{\Sigma}_k^{-1}\mathbf{H}_k^{\mathsf{H}}\mathbf{W}_k\mathbf{W}_k^{\mathsf{H}}\mathbf{H}_k\right|.$$

B. Relay Beamforming

We consider a scenario where there is an M-antenna relay station (RS) between the N-antenna BS and the single-antenna users. Let $\mathbf{G} \in \mathbb{C}^{M\times N}$ be the MIMO channel matrix between the BS and the RS and let $\mathbf{h}_k \in \mathbb{C}^M$ be the channel vector between the RS and user k. The transmitted signal from the BS is $\sum_{k=1}^K x_k\mathbf{w}_k$. Denote the BS beamformers $\mathbf{W} = [\mathbf{w}_1 \ \cdots \ \mathbf{w}_k] \in \mathbb{C}^{N\times K}$. The received signal at the RS is given by $\mathbf{z} = \mathbf{G}\sum_{k=1}^K x_k\mathbf{w}_k + \mathbf{v} \in \mathbb{C}^M$ where $\mathbf{v} \sim \mathcal{CN}(0,\sigma_r^2)$. The RS then employs a transmit beamforming matrix $\mathbf{F} \in \mathbb{C}^{M\times M}$ and forwards the signal $\mathbf{Fz} \in \mathbb{C}^M$ to the users. The received signal at user k can be written as $y_k = \mathbf{h}_k^{\mathsf{H}}\mathbf{Fz} + n_k$, or

$$y_k = \mathbf{h}_k^{\mathsf{H}}\mathbf{FG}\mathbf{w}_k x_k + \sum_{l\neq k}\mathbf{h}_k^{\mathsf{H}}\mathbf{FG}\mathbf{w}_l x_l + \mathbf{h}_k^{\mathsf{H}}\mathbf{Fv} + n_k,$$

where $n_k \sim \mathcal{CN}(0,\sigma^2)$. The SINR of user k is

$$\mathrm{SINR}_k = \frac{|\mathbf{h}_k^{\mathsf{H}}\mathbf{FG}\mathbf{w}_k|^2}{\sum_{l\neq k}|\mathbf{h}_k^{\mathsf{H}}\mathbf{FG}\mathbf{w}_l|^2 + \|\mathbf{h}_k^{\mathsf{H}}\mathbf{F}\|_2^2\sigma_r^2 + \sigma^2}. \tag{19}$$

and the user rates are given by $r_k := \log_2(1+\mathrm{SINR}_k)$. The goal is to choose beamformers $\mathbf{W}$ and $\mathbf{F}$ to maximize a performance function f subject to transmit power constraints at the BS and RS:

$$\begin{array}{ll}\underset{\mathbf{W},\mathbf{F}}{\text{maximize}} & f(r_1,\ldots,r_K)\\ \text{subject to} & \|\mathbf{W}\|_F^2 \le P_b\\ & \|\mathbf{F}\|_F^2 \le P_r.\end{array} \tag{20}$$

To obtain a suboptimal solution, we may plug (20) into a generic solver (e.g., sequential least-squares quadratic programming), or we may attempt to find a suboptimal point of the MMSE problem

$$\min_{\mathbf{W},\mathbf{F}} \mathbb{E}\left[\sum_{k=1}^{K}|y_k - x_k|^2\right] \ \text{s.t.}\ \|\mathbf{W}\|_F^2 = P_b,\ \|\mathbf{F}\|_F^2 = P_r. \tag{21}$$

via block coordinate descent. If $\mathbf{F}$ is fixed, then (21) reduces to a MISO MMSE problem and we may use (11) with the channel replaced with the effective channel $\mathbf{HFG}$. If $\mathbf{W}$ is fixed, then we may compute the unconstrained MMSE estimate for $\mathbf{F}$ in closed form via $\mathbf{F} = (\mathbf{HH}^{\mathsf{H}})^{-1}\mathbf{HW}^{\mathsf{H}}\mathbf{G}^{\mathsf{H}}(\mathbf{GWW}^{\mathsf{H}}\mathbf{G}^{\mathsf{H}} + \sigma_r^2\mathbf{I})^{-1}$ and then normalize such that $\|\mathbf{F}\|_F^2 = P_r$.

1) Proposed Learning Algorithm: The class of resource allocation problems under consideration is

$$\mathcal{P} = \left\{ \begin{array}{ll}\underset{\mathbf{F}\in\mathbb{C}^{M\times M},\mathbf{W}\in\mathbb{C}^{N\times K}}{\text{maximize}} & R(\mathbf{H},\mathbf{G},\mathbf{F},\mathbf{W})\\ \|\mathbf{W}\|_F^2 = P_b, \|\mathbf{F}\|_F^2 = P_r\end{array} \right\},$$

where $\mathbf{H} \in \mathbb{C}^{K\times M}$, $\mathbf{G} \in \mathbb{C}^{M\times N}$ and R is the performance criterion. We consider the sum rate $R(\mathbf{H},\mathbf{G},\mathbf{F},\mathbf{W}) = \sum_{k=1}^K \log_2(1+\mathrm{SINR}_k)$ where SINR_k is given by (19).

Applying Algorithm 2, we define two optimizers F_θ and G_ϕ and alternate between selection of variables $y := \mathbf{W}$ and $z := \mathbf{F}$ given $s := (\mathbf{H},\mathbf{G})$. Observe that when $\mathbf{F}$ is fixed, the task of selecting $\mathbf{W}$ is equivalent to that of the MISO downlink beamforming problem in Section IV with the channel matrix set equal to the effective channel $\tilde{\mathbf{H}} := \mathbf{HFG}$. When $\mathbf{W}$ is fixed, the task is to choose relay beamformers $\mathbf{F}$ given the BS-relay channel $\mathbf{H}$ and the relay-user effective channel $\mathbf{GW}$. The beamformers are computed by alternating between the two optimizers for $t = 1,\ldots,T$:

$$\mathbf{W}_{t,0} = \mathbf{W}_{t-1,T_W} \tag{22}$$
$$\mathbf{F}_{t,0} = \mathbf{F}_{t-1,T_F} \tag{23}$$
$$\tilde{\mathbf{H}}_t = \mathbf{HF}_{t,0}\mathbf{G} \tag{24}$$
$$\mathbf{W}_{t,k} = F_\theta(\tilde{\mathbf{H}}_t, \mathbf{W}_{t,k-1}), k = 1,\ldots,T_W \tag{25}$$
$$\mathbf{F}_{t,k} = G_\phi(\mathbf{GW}_{t,T_W}, \mathbf{H}, \mathbf{F}_{t,k-1}), k = 1,\ldots,T_F, \tag{26}$$

where we have unrolled F_θ and G_ϕ for T_F and T_G steps, respectively. We set $\mathbf{F}_{0,T_F} = \mathbf{I}/\sqrt{M}$ and $\mathbf{W}_{0,T_W} = \mathbf{HG}/\|\mathbf{HG}\|_F$. The input to F_θ is analogous to that of the MISO optimizer defined in Section IV, where $\tilde{\mathbf{H}}_t$ serves as the BS-user channel, but a key difference is that here the channel varies with t since it depends on the choice of $\mathbf{F}_t$. The inputs of G_ϕ include $\mathbf{GW}_t$ (the effective channel between the relay and the users) along with $\mathbf{H}$ and $\mathbf{F}_t$. The training objective is defined as

$$J(\theta,\phi) = \mathbb{E}\left[\sum_{t=1}^{T}\sum_{k=1}^{T_W} R(\mathbf{H},\mathbf{G},\mathbf{F}_{t,0},\mathbf{W}_{t,k}) \right. \tag{27}$$
$$\left. + \sum_{l=1}^{T_F} R(\mathbf{H},\mathbf{G},\mathbf{F}_{t,l},\mathbf{W}_{t,T_W})\right]. \tag{28}$$

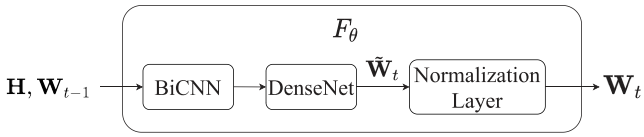


Fig. 5. Block diagram of our proposed optimizer network architecture for the MISO beamforming problem. An arbitrary iteration t is depicted. The input is the state $(\mathbf{H}, \mathbf{W}_{t-1})$. The output of the DenseNet module is $\tilde{\mathbf{W}}_t$ which is normalized to produce the output $\mathbf{W}_t$ with $\|\mathbf{W}\|_F^2$ satisfying the transmit power constraint.

Since for fixed $\mathbf{F}$ the problem has the same structure has the MISO scenario, F_θ may be pre-trained as a MISO beamforming optimizer (Section IV) so that it outputs the optimal $\mathbf{W}$ for any given $\tilde{\mathbf{H}}$. Having initialized F_θ, we then apply Algorithm 2.

VI. Results

The following experiments show that the learned optimizers significantly outperform the baseline state-of-the-art beamforming methods and in some scenarios achieve optimality. A combination of several training techniques are required in order to do so. These techniques are especially important for larger problem sizes and higher SNRs, where the learning task becomes more challenging and simple training methods fail. The first set of results show the performance of our best learned optimizer trained using the full set of training techniques. Then we perform an ablation study to probe the influence of each training technique.

A. Neural Network Architecture

The algorithms in Section II are valid for an optimizer with arbitrary neural network architecture. For the beamforming applications, we hand-designed a model architecture that exploits the structure of the state and action. The proposed neural network architecture consists of three modules (Fig. 5): biconvolutional neural network (BiCNN), DenseNet, and normalization layer. This model is used in all experiments. For $N = 4$ there are approximately 5×10^5 learnable parameters overall and for $N = 8$, approximately 4×10^6 learnable parameters.

1) BiCNN Module: The first module is a bilateral convolutional neural network (BiCNN) which performs feature extraction on the network input, lifting the input to a high dimensional space. Convolution exploits the translation invariance of the input along both user and antenna dimensions, since for any ordering of input channel vectors the optimal beamformers are the same (up to a permutation of the user indices) and vice versa. For the MISO optimizer, the input is an array with shape $(4, K, N)$ comprising the concatenation of the real and imaginary parts of $\mathbf{H}$ and $\mathbf{W}$. The module consists of two subnetworks each with two 2-D convolution layers. One subnetwork filters over the user axis while the other filters over the antenna axis. The overall output is the sum of the subnetwork outputs.

2) DenseNet Module: The DenseNet module consists of two linear layers with shortcut connections as in [25]. The DenseNet's shortcut connections allow for more sophisticated features using fewer parameters [25]. This module transforms the high-dimensional BiCNN output to a vector of $2NK$ real numbers which form the beamformer matrix in $\mathbb{C}^{N\times K}$.

3) Normalization Layer: Each beamforming scenario includes a constraint on the transmit power. This constraint is easily enforced by appending a normalization layer. In particular, suppose $\tilde{\mathbf{W}} \in \mathbb{C}^{N\times K}$ is the beamformer matrix produced by the DenseNet module. Then, for a given transmit power P, the normalization layer outputs $\mathbf{W} = \sqrt{P}\tilde{\mathbf{W}}/\|\tilde{\mathbf{W}}\|_F$ so that the output satisfies $\|\mathbf{W}\|_F^2 = P$.

TABLE II
Wall Clock Run Times for MISO Beamforming Methods. "Ours" Is the Per-Step Feedforward Inference Time on an A100 GPU (e.g., for $T = 5$ Total Steps in the Case $N = K = 4$, the Total Inference Time Is Around 14 Milliseconds). The Optimal Methods Are Described in Section IV and Implemented on a 3.7 GHz CPU. The Run Times of "Ours" and MMSE Are Independent of the Channel Realization. The Optimal Method Run Time Is Generally Different for Each Realization; We Report the Average Over 100 Samples

	$N = K = 4$	$N = K = 8$
Ours (per step)	2.8 ms	4.0 ms
Optimal (sum rate)	20 minutes	6 hours
Optimal (min rate)	4 seconds	6 seconds
MMSE	0.6 ms	0.8 ms

B. MISO Downlink Beamforming

Here we present results for the MISO beamforming scenario of Section IV. Training samples are generated $[\mathbf{H}]_{ij} \overset{\text{i.i.d.}}{\sim} \mathcal{CN}(0, \frac{1}{NK}\sigma_{\mathbf{H}}^2)$. The SNR is

$$\text{SNR} := \frac{\mathbb{E}[\|\mathbf{H}^{\text{H}}\mathbf{W}\|_F^2]}{\sigma^2} = \frac{\sigma_{\mathbf{H}}^2 P}{\sigma^2}$$

where we have used the constraint $\|\mathbf{W}\|_F^2 = P$. Without loss of generality, we fix $P = 1$ and $\sigma^2 = 1$ so that $\text{SNR} = \sigma_{\mathbf{H}}^2$. For each sample $\sigma_{\mathbf{H}}^2$ is drawn uniformly from $\{10\text{ dB}, 15\text{ dB}, 20\text{ dB}\}$, so that the network is trained on a variety of samples. In Section VI.E.4 we further investigate the effect of the training distribution properties.

Fig. 6 shows the results for sum rate (left) and min rate (right) versus SNR, including the cases $N = K = 4$ (solid) and $N = K = 8$ (dashed). The L2O policies were trained using Algorithm 1 combined with the MSE reward curriculum (Section IV.B.1). The true optimal beamformers are computed as described in Section IV.A.1. For each SNR test case, the learned optimizer achieves a significantly higher objective value than MMSE, and in the $N = K = 4$ case achieves virtually optimal sum rate. The single step optimizer without curriculum learning (green), on the other hand, performs worse than MMSE, showing that the naive implementation is inadequate.

The wall clock run times of each method are reported in Table II. We report the per-step execution time of the learned optimizer (i.e., for the case $T = 1$). Compared to the optimal sum rate and min rate methods, the learned optimizer run time

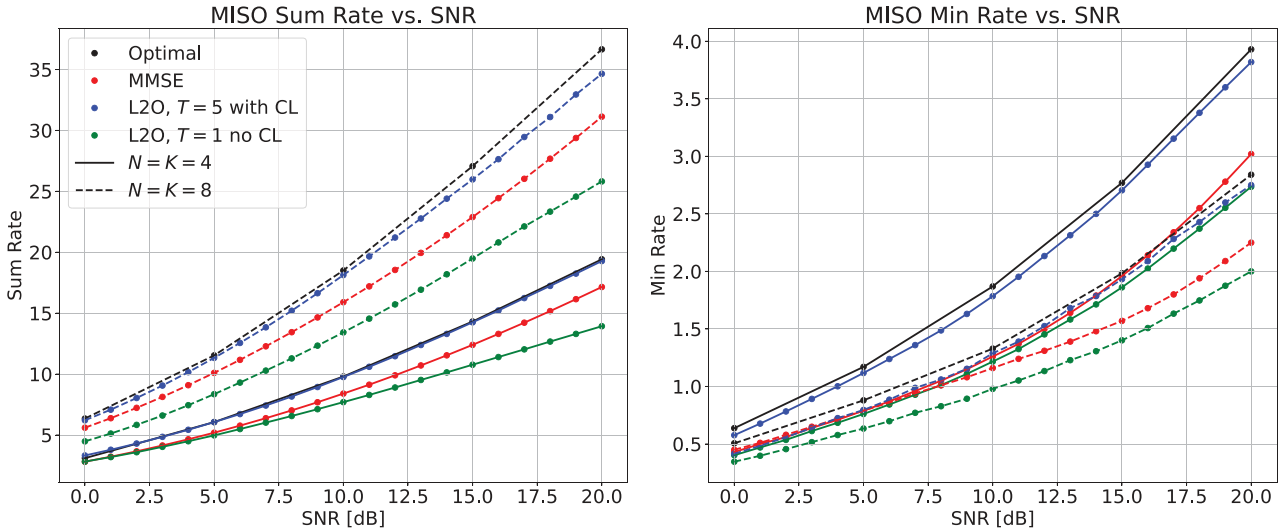


Fig. 6. MISO sum rate (left) and min rate (right) versus SNR. Three learned optimizers are trained with different training distributions: variable SNR $\in \{10, 15, 20\}$, SNR $= 10$ dB, and SNR $= 20$ dB. The variable SNR net obtains optimal beamformers within at most 1% and 7% for $N = K = 4$ (solid) and $N = K = 8$ (dashed), respectively.

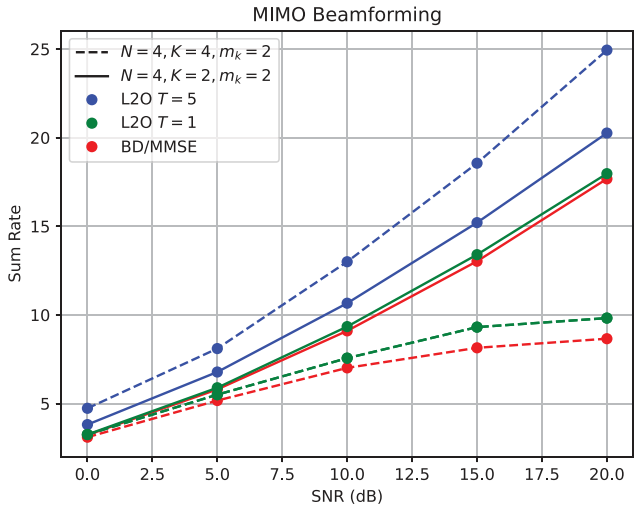


Fig. 7. MIMO sum rate versus SNR. Three learned optimizers are trained with different training distributions: variable SNR $\in \{10, 15, 20\}$, SNR $= 10$ dB, and SNR $= 20$ dB. For $N = 4$, $K = 2$, $m_k = 2$, the learned optimizers outperform the block diagonalization (BD) beamformers. For the overloaded case $N = K = 4$ and $m_k = 2$, the learned optimizers perform far better than MMSE.

is several orders of magnitude lower. Essentially, the computational burden has been amortized during the training process—when the model is deployed, for each new channel realization we only need to execute T forward passes of the trained model in order to obtain near-optimal beamformers. We emphasize that the same network architecture is used for all beamforming scenarios, therefore the per-step execution times of "Ours" also hold for the MIMO and relay scenarios.

C. MIMO Downlink Beamforming

Fig. 7 shows the sum rate versus SNR curve of the learned optimizers trained as described in Section V.A.1. We fix the number of BS antennas $N = 4$ and the number of antennas per user $m_k = 2$ for each user $k = 1, \ldots, K$, and consider $K = 2$. In the MIMO case there is no known optimal solution, so we compare with the block-diagonalization beamformers for $K = 2$ and the MMSE beamformers for $K = 4$. The block-diagonalization beamforming method is not applicable when $\sum_k m_k > N$, so for $K = 4$ we treat each antenna as a single user and compute the corresponding MMSE beamformers as in the MISO case. Algorithm 1 is used with the MIMO sum rate formula as the reward function and subspace curriculum learning (Algorithm 3). The learned optimizers achieve a substantially higher sum rate than the BD and MMSE beamformers, particularly for the overloaded case $K = 4$.

D. Relay Beamforming

For the relay problem we apply Algorithm 2 as described in Section V.B.1. The $\mathbf{W}$-optimizer, F_θ, is pretrained as if it were a MISO beamforming optimizer, so that upon initialization it outputs the optimal beamformers for any given effective channel $\tilde{\mathbf{H}} = \mathbf{HFG}$. We set $T = T_W = T_F = 6$. Assuming a Gaussian channel, we have $[\mathbf{H}]_{ij} \overset{\text{i.i.d.}}{\sim} \mathcal{CN}(0, \frac{1}{NK}\sigma_{\mathbf{H}}^2)$ and $[\mathbf{G}]_{ij} \overset{\text{i.i.d.}}{\sim} \mathcal{CN}(0, \frac{1}{MK}\sigma_{\mathbf{G}}^2)$. The relay and BS transmit powers are $P_b = P_r = 1$, we assume $\sigma^2 = \sigma_r^2$ and $\sigma_s^2 := \sigma_{\mathbf{H}}^2 = \sigma_{\mathbf{G}}^2$. The SNR is defined as $\text{SNR} := \frac{\sigma_s^2}{\sigma^2}$. Without loss of generality, we fix $\sigma^2 = 1$ so that $\text{SNR} = \sigma_s^2$. As baselines, we use the SciPy sequential least-squares quadratic programming (SLSQP) solver, as well as the alternating least squares mentioned in the discussion following (21). We consider two scenarios, $N = M = K = 4$ and $N = M = K = 8$. The resulting sum rate versus SNR performance is shown in Fig. 8. The learned optimizers perform well above Alternating MMSE and are on par with the generic solver. The generic solver run time is approximately 4 seconds per sample, alternating MMSE is approximately 500 milliseconds per sample, and the learned optimizers with $T = 5$ require approximately 30 milliseconds per sample.

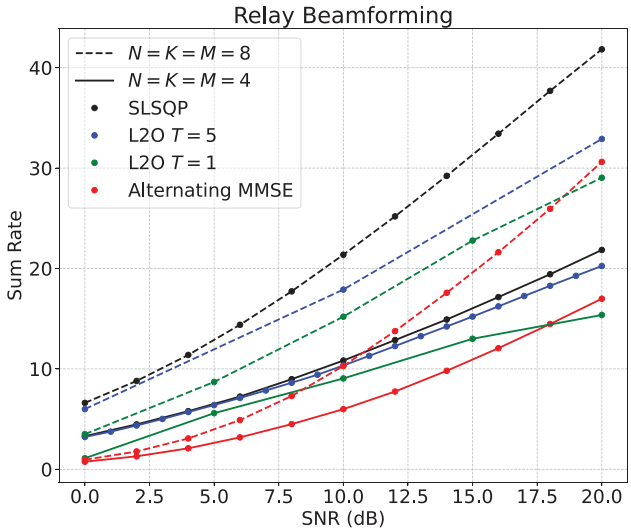


Fig. 8. Relay sum rate versus SNR. Three learned optimizers are trained with different training distributions: variable SNR $\in \{10, 15, 20\}$, SNR $= 10$ dB, and SNR $= 20$ dB. For $N = K = 4$, the learned optimizers are on par with the generic solver, but require an order of magnitude less run time to obtain the beamformers during testing.

E. Ablation Study

As seen in the previous set of experiments, the naive implementation performs worse than the baseline MMSE beamformers. The following techniques are required

- Curriculum learning (training objective or training data)
- Multi-step optimizer
- Untied parameters (neural network parameters may vary across steps)
- Variable SNR training data

To illuminate the impact of each technique, we perform an ablation study for the MISO scenario.

1) Curriculum Learning: Fig. 9 includes a set of training curves for a variety of training procedures labeled as follows:

- Reward Curriculum: Algorithm 1 with reward curriculum (Section IV) using the MSE and sum rate objective as the two tasks. First the optimizer is trained using the MSE objective for 4000 epochs, then training continues with the sum rate objective until convergence.
- Subspace Curriculum: Algorithm 1 with subspace curriculum learning with the sum rate objective.
- MSE Objective: Algorithm 1 with MSE objective (14), no curriculum learning.
- Sum Rate Objective: Algorithm 1 with sum rate objective, no curriculum learning.

We find that Reward Curriculum and Subspace Curriculum yield an optimizer that approaches optimality. To achieve a target sum rate of 95% of the optimal, the Reward Curriculum method requires about half the epochs for $N = K = 4$ and about a third of the epochs for $N = K = 8$ relative to that required by the Subspace Curriculum.

2) Tied vs. Untied Parameters: Fig. 10 illustrates the effect of the number of steps T, as well as the effect of untying the parameters across time steps (Section II). In the first stage of training, the optimizer parameters are untied (shared across steps), and after convergence, the parameters are untied (allowed to vary across steps) and training resumes until convergence. Each data point represents an optimizer that was trained with T equal to the abscissa. We find that the untied optimizer achieves higher reward than the tied optimizer.

3) Multiple Steps: Moreover, from Fig. 10 it is also seen that multi-step policies perform much better than the single step optimizer. Since action t depends on action $t-1$, the input data distribution changes at each step; therefore, for $T > 1$, at each step the optimizer observes higher quality inputs and further refines them toward the optimum. Furthermore, unrolling the optimizer for multiple steps allows it to explore the action space and escape local minima.

4) Training Data SNR: Fig. 11 investigates how performance varies as the test SNR deviates from the training SNR. The variable SNR optimizer (blue) was trained on sample channels with SNR drawn uniformly between 10 dB and 20 dB, while the green and yellow curves correspond to policies trained exclusively at 20 dB and 10 dB. The single SNR optimizers (green and yellow) achieve near-optimal sum rate when tested on sample channels with the same SNR on which they were trained, while performance slightly worsens for SNRs outside of the training SNR. The variable SNR optimizer, on the other hand, is robust across the entire SNR range.

F. Discussion

The experiments reveal the challenges of scaling neural networks to larger input dimensions. Doubling the problem size from $N = K = 4$ to $N = K = 8$ required a sixfold increase in training epochs and tenfold increase in the number of learnable parameters. Beyond $N = K = 8$, we attempted to further increase the network size but the curriculum strategies yielded diminishing returns and the training converged to a suboptimal point. Future work may investigate different network architectures and weight compression techniques to reduce the size of the parameter space.

VII. Related Work: Deep Learning for Beamforming Design

Here we give a brief survey of prior work applying deep learning techniques for beamforming design so as to situate our proposed method. [20] considers a multiuser communication setup with K SISO channels and designs learning-based approach to optimize only the power allocation; there is no beamformer design because the channels are scalar. The network is trained in a supervised manner based on a training set of optimal power allocations obtained via the WMMSE algorithm, resulting in a learned model whose output power allocations obtain near-optimal sum rate. A similar approach is applied to a system of K MIMO channels in [26] to optimize both transmit and receive beamformers, except training proceeds in two stages: first, supervised learning based on training samples obtained via WMMSE, followed by unsupervised learning using the sum rate as the training objective. In [27], the WMMSE algorithm is modified and unfolded into a neural network trained with the WMMSE objective as the unsupervised learning objective.

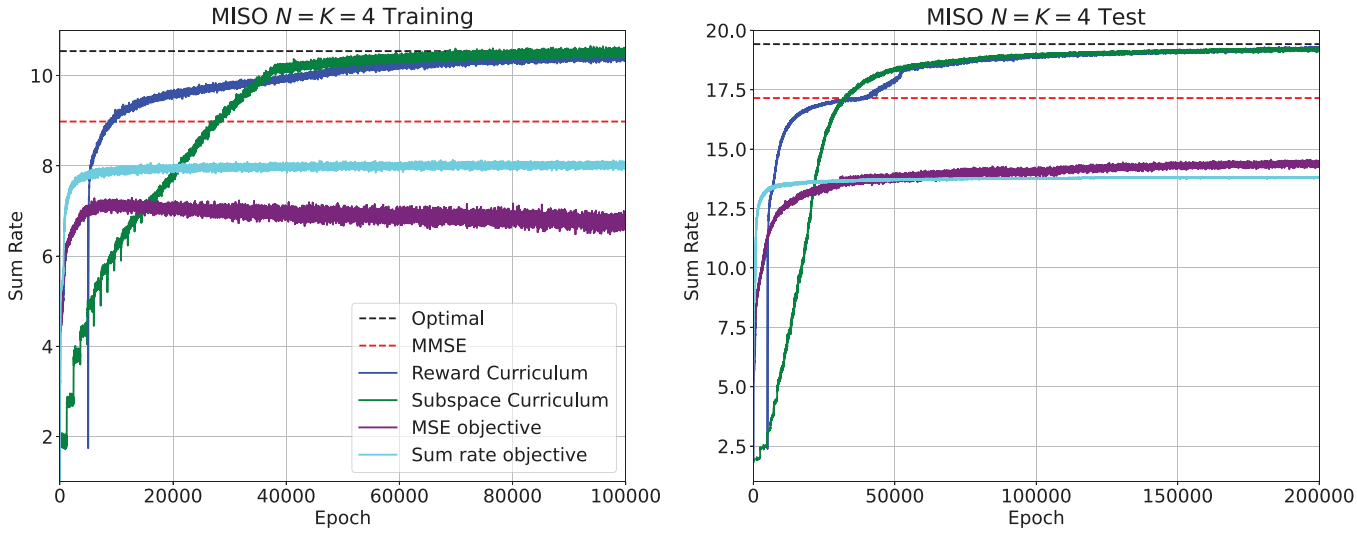


Fig. 9. Training (left) and test (right) sum rate versus training epoch for MISO Variable SNR optimizer. The training data is a uniform mixture of 10, 15 and 20 dB channels (the optimal and MMSE are computed for a subset of 100 samples, for reference). The test set consists of 100 samples with $\mathrm{SNR} = 20$ dB. Either reward curriculum (RC) or subspace curriculum (SC) is required in order to attain optimality; without curriculum learning the optimizer networks converge below the MMSE sum rate. For RC, the tasks are switched at 4000 epochs; hence the sudden drop in sum rate, followed by a steady increase. To attain 95% of the optimal sum rate, RC requires half the number of epochs than SC.

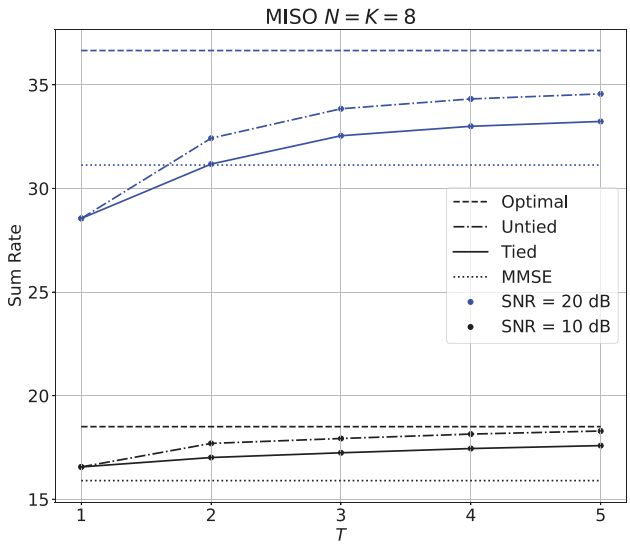


Fig. 10. Sum rate versus T for tied and untied policies in the $N = K = 8$ MISO beamforming scenario. Each point with abscissa T corresponds to a T-step optimizer. The sum rate increases with T and the untied optimizer achieves higher sum rate.

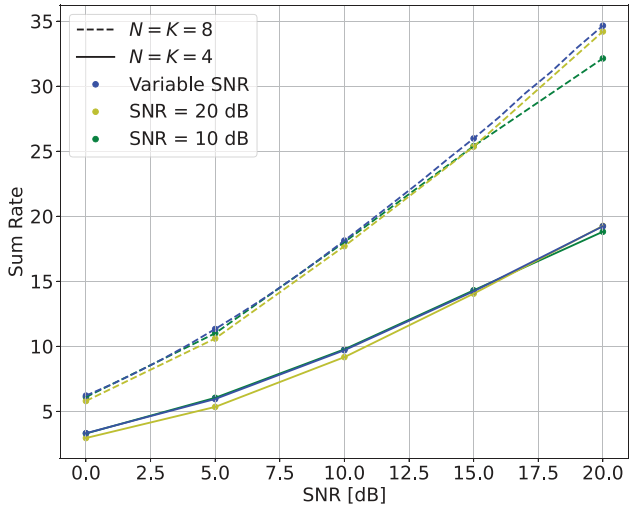


Fig. 11. Sum rate versus test set SNR in the MISO beamforming scenario, $N = K = 4$ (solid) and $N = K = 8$ (dashed). The variable SNR optimizer is trained with data drawn uniformly between 10 and 20 dB, while the other two policies are trained exclusively on data with a single SNR. The variable SNR optimizer is robust across the entire SNR range, while the performance of single-SNR policies degrades as the test SNR deviates from the training SNR.

Similar to [26], we consider a multi-stage training approach under the rubric of curriculum learning [8]. For the scenarios considered in this paper, supervised learning is not feasible because computing the optimal solution is too time consuming, therefore our approach is unsupervised.

A fully unsupervised learning approach is applied to jointly optimize beamformers and the reflective pattern of a reflective intelligent surface (RIS) in [28], where a graphical neural network architecture is chosen in order to exploit the permutation invariant property of the beamformers with respect to the user ordering and also allow the model to support a variable number of users. Our proposed architecture also seeks to exploit the permutation invariance, however we assume a fixed number of users. In [29], the NN takes as input a measurement obtained during the uplink pilot transmission phase and directly outputs the beamformer and RIS weights, hence bypassing the channel estimation task. Similarly, in [28] channel state information at the transmitter (CSIT) is not present, but the beamforming-RIS design is instead formulated as a Markov Decision Process. Without CSIT, the reward (e.g., sum rate) cannot be directly computed and therefore must be estimated based on sample

observations; the authors propose an augmented version DDPG that incorporates aspects of the system model. In our approach, we assume perfect CSIT and therefore the quality of candidate beamformers and gradients can be directly calculated for training. If CSIT is known, then the learning algorithm in [29], in the context of our presentation, is equivalent to training with the sum rate reward without curriculum learning and $T = 1$.

VIII. Conclusion

We have demonstrated that neural networks can closely approximate the optimal solution map for MISO sum rate maximization and min rate maximization problems. Curriculum learning proved essential to learning the optimizer; without a curriculum, the optimizer network converged to a suboptimal point. For the MIMO and relay case, our learned optimizers are at least on par with baseline methods and in some cases far outperform them in terms of sum rate. The learning task becomes quite difficult as the number of users and antennas grow. In future work, the network architecture may be fine tuned and implemented with more computing resources in order to accommodate larger systems. Having demonstrated that it is possible to approximate the optimal solution mapping of the MISO problem, we expect that there are other problem families out there that can be solved via our approach that would significantly improve existing real-time optimization applications and foster new ones.

References

[1] M. Andrychowicz et al., "Learning to learn by gradient descent by gradient descent," in *Proc. Adv. Neural Inf. Process. Syst.*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29., Red Hook, NY, USA: Curran Associates, 2016.

[2] K. Li and J. Malik, "Learning to optimize," in *Proc. 5th Int. Conf. Learn. Representations (ICLR)*, Toulon, France, Apr. 24–26, 2017.

[3] T. Chen et al., "Learning to optimize: A primer and a benchmark," *J. Mach. Learn. Res.*, vol. 23, no. 1, pp. 8562–8620, 2022.

[4] B. Amos, "Tutorial on amortized optimization for learning to optimize over continuous domains," 2022, *arXiv:2202.00665*.

[5] M. Jin, V. Khattar, H. Kaushik, B. Sel, and R. Jia, "On solution functions of optimization: Universal approximation and covering number bounds," *Proc. AAAI Conf. Artif. Intell.*, vol. 37, no. 7, pp. 8123–8131, 2023.

[6] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," *Comput. Oper. Res.*, vol. 134, Oct. 2021, Art. no. 105400.

[7] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," *Eur. J. Oper. Res.*, vol. 290, no. 2, pp. 405–421, Apr. 2021.

[8] Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proc. 26th Annu. Int. Conf. Mach. Learn. (ICML)*, New York, NY, USA: ACM, 2009, pp. 41–48.

[9] T. Chen et al., "Training stronger baselines for learning to optimize," in *Proc. Adv. Neural Inf. Process. Syst.*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33., Red Hook, NY, USA: Curran Associates, 2020, pp. 7332–7343.

[10] E. Björnson, M. Bengtsson, and B. Ottersten, "Optimal multiuser transmit beamforming: A difficult problem with a simple solution structure [Lecture Notes]," *IEEE Signal Process. Mag.*, vol. 31, no. 4, pp. 142–148, Jul. 2014.

[11] A. B. Gershman, N. D. Sidiropoulos, S. Shahbazpanahi, M. Bengtsson, and B. Ottersten, "Convex optimization-based beamforming," *IEEE Signal Process. Mag.*, vol. 27, no. 3, pp. 62–75, May 2010.

[12] M. Borgerding, P. Schniter, and S. Rangan, "AMP-inspired deep networks for sparse linear inverse problems," *IEEE Trans. Signal Process.*, vol. 65, no. 16, pp. 4293–4308, Aug. 2017.

[13] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Found. Trends Mach. Learn.*, vol. 3, no. 1, pp. 1–122, 2011.

[14] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[15] P. Hernandez-Leal, B. Kartal, and M. Taylor, "A survey and critique of multiagent deep reinforcement learning," *Auton. Agent Multi-Agent Syst.*, vol. 33, pp. 750–797, Nov. 2019.

[16] J. R. Hershey, J. L. Roux, and F. Weninger, "Deep unfolding: Model-based inspiration of novel deep architectures," 2014. [Online]. Available: https://arxiv.org/abs/1409.2574

[17] Y. Yang, J. Sun, H. Li, and Z. Xu, "Deep ADMM-Net for compressive sensing MRI," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 29, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds., Red Hook, NY, USA: Curran Associates, 2016, pp. 10–18.

[18] K. Gregor and Y. LeCun, "Learning fast approximations of sparse coding," in *Proc. 27th Int. Conf. Mach. Learn. (ICML)*, 2010, pp. 399–406.

[19] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 359–366, 1989.

[20] H. Sun, X. Chen, Q. Shi, M. Hong, X. Fu, and N. D. Sidiropoulos, "Learning to optimize: Training deep neural networks for interference management," *IEEE Trans. Signal Process.*, vol. 66, no. 20, pp. 5438–5453, Oct. 2018.

[21] M. Eisen, C. Zhang, L. F. O. Chamon, D. D. Lee, and A. Ribeiro, "Learning optimal resource allocations in wireless systems," *IEEE Trans. Signal Process.*, vol. 67, no. 10, pp. 2775–2790, May 2019.

[22] D. Weinshall and D. Amir, "Theory of curriculum learning, with convex loss functions," *J. Mach. Learn. Res.*, vol. 21, no. 1, pp. 9184–9202, 2020.

[23] E. Björnson and E. Jorswieck, "Optimal resource allocation in coordinated multi-cell systems," *Found. Trend® Commun. Inf. Theory*, vol. 9, nos. 2–3, pp. 113–381, 2013, doi: 10.1561/0100000069.

[24] C. Peel, Q. Spencer, A. Swindlehurst, and B. Hochwald, "Downlink transmit beamforming in multi-user MIMO systems," in *Proc. Process. Workshop, Sensor Array Multichannel Signal*, 2004, pp. 43–51.

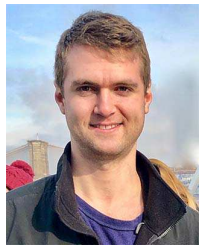
[25] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," *presented at the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2261–2269, Los Alamitos, CA, USA: IEEE Computer Society, Jul. 2017. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2017.243

[26] H. Huang, Y. Peng, J. Yang, W. Xia, and G. Gui, "Fast beamforming design via deep learning," *IEEE Trans. Veh. Technol.*, vol. 69, no. 1, pp. 1065–1069, Jan. 2020.

[27] L. Pellaco, M. Bengtsson, and J. Jaldén, "Matrix-inverse-free deep unfolding of the weighted MMSE beamforming algorithm," *IEEE Open J. Commun. Soc.*, vol. 3, pp. 65–81, 2022.

[28] X. Jia and X. Zhou, "IRS-assisted ambient backscatter communications utilizing deep reinforcement learning," *IEEE Wireless Commun. Lett.*, vol. 10, no. 11, pp. 2374–2378, Nov. 2021.

[29] T. Jiang, H. V. Cheng, and W. Yu, "Learning to reflect and to beamform for intelligent reflecting surface with implicit channel estimation," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 7, pp. 1931–1945, Jul. 2021.



Jeremy Johnston received the B.S. in electrical engineering from the University of Florida, in 2018. He is currently working toward the Ph.D. degree in electrical engineering with Columbia University.

Xiao-Yang Liu (Graduate Student Member, IEEE) received the B.Eng. degree in computer science from Huazhong University of Science and Technology, China, in 2010, and the M.S. degree in electrical engineering from Columbia University, USA, in 2018, and the Ph.D. degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, in 2017. He is now working toward the Ph.D. degree with the Department of Electrical Engineering, Columbia University. His research interests include deep learning, optimization algorithms, high-performance tensor computing, and big data analysis.

Shi-Xun Wu received the B.S. degree in computer science from Peking University, China, in 2020, and the M.S. degree in electrical engineering from Columbia University, USA, in 2022. He is now working toward the Ph.D. degree with the Department of Computer Science and Engineering, University of California, Riverside. His research interests include high-performance computing and deep learning.

Xiaodong Wang (Fellow, IEEE) received the Ph.D. degree in electrical engineering from Princeton University, Princeton, NJ, USA. He is a Professor in electrical engineering with Columbia University, New York, NY, USA. Among his publications is a book entitled *Wireless Communication Systems: Advanced Techniques for Signal Reception* (Prentice Hall, 2003). His current research interests include wireless communications, statistical signal processing, genomic signal processing, general areas of computing, and signal processing and communications, and has published extensively in these areas. He received the 1999 NSF CAREER Award, the 2001 IEEE Communications Society and Information Theory Society Joint Paper Award, and the 2011 IEEE Communication Society Award for Outstanding Paper on New Communication Topics. He was an Associate Editor for IEEE TRANSACTIONS ON COMMUNICATIONS, IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, IEEE TRANSACTIONS ON SIGNAL PROCESSING, and IEEE TRANSACTIONS ON INFORMATION THEORY. He is listed as an ISI highly cited author.