Unsupervised Learning for Parametric Optimization

Rasoul Nikbakht, *Student Member, IEEE*, Anders Jonsson, Angel Lozano, *Fellow, IEEE*

***Abstract*—This letter proposes the unsupervised training of a feedforward neural network to solve parametric optimization problems involving large numbers of parameters. Such unsupervised training, which consists in repeatedly sampling parameter values and performing stochastic gradient descent, foregoes the taxing precomputation of labeled training data that supervised learning necessitates. As an example of application, we put this technique to use on a rather general constrained quadratic program. Follow-up letters subsequently apply it to more specialized wireless communication problems, some of them nonconvex in nature. In all cases, the performance of the proposed procedure is very satisfactory and, in terms of computational cost, its scalability with the problem dimensionality is superior to that of convex solvers.**

***Index Terms*—Machine learning, neural networks, unsupervised learning, parametric optimization, convex optimization, quadratic program**

I. Introduction

In parametric optimization [1]–[4], the aim is not to solve a specific optimization problem, but rather to represent the solution to an entire family of problems. This family of problems is governed by parameters, such that each combination of parameter values corresponds to a particular optimization problem. The desired output of parametric optimization is a mapping from parameter values to the optimum solution of the specific associated problem.

When the objective function is nonlinear in the input, the setting becomes that of parametric nonlinear programming [5]. For this setting, a number of approximation algorithms exist, including outer approximations [5], [6], gradient descent [7], or piecewise quadratic fitting [8]. Such state-of-the-art algorithms have in common that they completely (or approximately) solve a sequence of optimization problems in order to estimate the desired mapping between parameters and solutions. When the parameter dimensionality increases, these algorithms need to solve a very large number of such optimizations in order to estimate an accurate solution, eventually becoming computationally intractable. As a result, existing algorithms scale poorly to parametric optimizations with many parameters.

This letter propounds to leverage the expressive power of neural networks (NNs) to solve large-dimensional parametric optimizations. Because of their nonlinear nature, NNs are ideal candidates for nonlinear optimizations. Indeed, NNs have been applied already to solve parametric optimizations in a supervised fashion, by sampling parameter values and solving the associated optimization problems [9], [10]. Again, for large parameter dimensionalities this becomes unfeasible because of the sheer size of the parameter space.

As an alternative, what is proposed in this letter is to train a feedforward NN by repeatedly sampling parameter values, but, rather than completely solving the associated optimization problem for each, taking a single step along the gradient of the objective function. Despite not providing the NN with the solution for each parameter value, descending along the gradient allows the NN to generalize information for different such values.

To test the proposed idea, we apply it to quadratic programming (QP), a simple but very common—and convex—class of optimizations. In two follow-up letters, we turn to more involved problems motivated by wireless communications and not always exhibiting convex objectives. Precisely, power control for cellular networks is considered in [11] whereas power control for centralized radio-access networks is the object of [12].

II. Parametric Optimization

Letting $\boldsymbol{\theta} \in \Theta \subseteq \mathbb{R}^Q$ be a Q-dimensional parameter vector and $\boldsymbol{x} \in \mathcal{X} \subseteq \mathbb{R}^D$ a D-dimensional optimization vector, consider the optimization problem

$$\min_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{\theta}, \boldsymbol{x}). \tag{1}$$

In parametric optimization, the goal is to estimate a *minimizer function* $\boldsymbol{x}^\star : \Theta \to \mathcal{X}$ or a *value function* $g : \Theta \to \mathbb{R}$, both of which map parameter vectors to a solution of the corresponding optimization problem. For a specific $\boldsymbol{\theta}$, the minimizer and value functions are

$$\boldsymbol{x}^\star(\boldsymbol{\theta}) = \arg\min_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{\theta}, \boldsymbol{x}) \tag{2}$$

$$g(\boldsymbol{\theta}) = \min_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{\theta}, \boldsymbol{x}). \tag{3}$$

Note that, if we have access to $\boldsymbol{x}^\star(\boldsymbol{\theta})$, the value function is trivially $g(\boldsymbol{\theta}) = f\big(\boldsymbol{\theta}, \boldsymbol{x}^\star(\boldsymbol{\theta})\big)$. However, in general it is not as easy to induce a minimizer function from a value function.

Previous work in parametric optimization typically imposes restrictions on the sets Θ and $\mathcal{X}$ (convexity) as well as on the objective function f (differentiability and, again, convexity). Under these restrictions, and in particular if both $\mathcal{X}$ and f are convex, any local minimizer is also a global minimizer and hence $\boldsymbol{x}^\star$ is well defined. One contribution of our approach is that it makes minimal assumptions about the optimization problem: the only strict requirement is that f be differentiable in $\boldsymbol{x}$. We also do take the feasible set $\mathcal{X}$ to be convex, which suffices to ensure that $\boldsymbol{x}^\star$ is well defined, but we do not restrict f to being convex. Of course, for nonconvex f, what gradient

R. Nikbakht, A. Jonsson, and A. Lozano are with Univ. Pompeu Fabra, 08018 Barcelona (e-mail: angel.lozano@upf.edu). This work was supported by the European Research Council under the H2020 Framework Programme/ERC grant agreement 694974, by the Maria de Maeztu Units of Excellence Programme (MDM-2015-0502) as well as by MINECO's Projects RTI2018-102112 and RTI2018-101040, and by the ICREA Academia program.

descent can do is to find a local minimizer, with no guarantee that it is also the global minimizer $\boldsymbol{x}^\star$.

As advanced in the introduction, several algorithms do exist already that efficiently approximate $\boldsymbol{x}^\star(\boldsymbol{\theta})$ or $g(\boldsymbol{\theta})$. However, these come at the expense of having to solve a sequence of optimization problems whose number increases exponentially with the parameter dimensionality, Q.

III. Proposed Approach

A. Neural Networks

NNs are universal function approximators that combine simple nonlinear units to form complex networks with substantial expressive power. The universal approximation theorem states that feed-forward NNs with a single hidden layer can approximate continuous functions to arbitrary precision [13]–[15]. Formally, a feed-forward NN represents a vector-valued function $\boldsymbol{h}(\boldsymbol{a}, \boldsymbol{w})$ with D output dimensions, where $\boldsymbol{a}$ is an input vector and $\boldsymbol{w}$ is a vector of weights. NNs are typically trained using gradient descent on a given loss function $L(\boldsymbol{y})$, where $\boldsymbol{y} = \boldsymbol{h}(\boldsymbol{a}, \boldsymbol{w})$ is the NN's output. Since

$$L(\boldsymbol{y}) = L(\boldsymbol{h}(\boldsymbol{a}, \boldsymbol{w})) \tag{4}$$

$$= (L \circ \boldsymbol{h})(\boldsymbol{a}, \boldsymbol{w}), \tag{5}$$

the chain rule allows us to write

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{y}) = \boldsymbol{J}(\boldsymbol{a}, \boldsymbol{w})^\top \nabla_{\boldsymbol{y}} L\big(\boldsymbol{h}(\boldsymbol{a}, \boldsymbol{w})\big), \tag{6}$$

where $\boldsymbol{J}(\boldsymbol{a}, \boldsymbol{w})$ is the Jacobian matrix

$$\boldsymbol{J}(\boldsymbol{a}, \boldsymbol{w}) = \begin{bmatrix} \nabla_{\boldsymbol{w}} h_0(\boldsymbol{a}, \boldsymbol{w})^\top \\ \vdots \\ \nabla_{\boldsymbol{w}} h_{D-1}(\boldsymbol{a}, \boldsymbol{w})^\top \end{bmatrix}. \tag{7}$$

The Jacobian is not explicitly computed; rather, backpropagation is used to disseminate the gradient through the NN in order to update its weights [16]. For example, a regression task can be formulated by means of a square loss $L(\boldsymbol{y}) = \|\boldsymbol{y} - \boldsymbol{t}\|^2$, where $\boldsymbol{t}$ is the target output associated with input $\boldsymbol{a}$. The goal is to minimize the loss such that the difference between $\boldsymbol{y}$ and $\boldsymbol{t}$ becomes as small as possible for each input $\boldsymbol{a}$.

B. Unsupervised Learning for Parametric Optimization

Let us consider an NN that represents a minimizer function $\boldsymbol{x}^\star : \Theta \to \mathcal{X}$. Upon an input $\boldsymbol{\theta} \in \Theta$, the network outputs a D-dimensional vector $\boldsymbol{x} = \boldsymbol{h}(\boldsymbol{\theta}, \boldsymbol{w})$. In supervised learning [10], NN training requires a learning stage in which parameter vectors $\boldsymbol{\theta}_0, \ldots, \boldsymbol{\theta}_{S-1}$ are sampled and each of the associated convex optimization problems is solved to obtain respective target outputs $\boldsymbol{x}^\star_0, \ldots, \boldsymbol{x}^\star_{S-1}$. The NN is then trained by means of regression as described above, with the loss being a function of the difference between the predicted and the target output.

The NN is trained by means of stochastic gradient descent applied directly on the objective function, f. To do so, we define the loss function

$$L(\boldsymbol{x}) = f(\boldsymbol{\theta}, \boldsymbol{x}), \tag{8}$$

whose gradient equals

$$\nabla_{\boldsymbol{w}} L(\boldsymbol{x}) = \boldsymbol{J}(\boldsymbol{a}, \boldsymbol{w})^\top \nabla_{\boldsymbol{x}} f\big(\boldsymbol{\theta}, \boldsymbol{h}(\boldsymbol{\theta}, \boldsymbol{w})\big), \tag{9}$$

which requires f to be differentiable in $\boldsymbol{x}$. As anticipated, this is the one premise that cannot be lifted.

If the constraints are simple enough, they can be hardwired into the structure of the NN itself, say by selecting nonlinear units for the output layer that can only produce values within a certain range [17]. With a view to broader generality, though, we prefer to transform the constrained optimization problem

$$\min_{\boldsymbol{x} \in \mathcal{X}} L(\boldsymbol{x}) \tag{10}$$

into the unconstrained optimization problem

$$\min_{\boldsymbol{x}} L(\boldsymbol{x}) + \boldsymbol{\beta}^\top \boldsymbol{C}(\boldsymbol{x}), \tag{11}$$

where $\boldsymbol{C}(\boldsymbol{x})$ is the vector of constraints that define the feasible set $\mathcal{X}$ and $\boldsymbol{\beta}$ is a vector of Lagrangian (or KKT) multipliers. Hence, the gradient also involves a problem-dependent term that corresponds to $\boldsymbol{C}(\boldsymbol{x})$. Since the loss function is a combination of the objective function and of the cost that is associated with the constraints, training the NN by gradient descent has the effect of minimizing $f(\boldsymbol{\theta}, \boldsymbol{x})$ over the feasible set $\mathcal{X}$, which is precisely the aim of parametric optimization. To circumvent having to identify the optimum $\boldsymbol{\beta}$ analytically, which would require solving the dual optimization problem, we adopt the common strategy of cross-validation: apply the same multiplier β to every constraint, and select its value after trialing $\beta \in \{0.01, 0.1, 1, 10, 100\}$.

The crux of the proposed approach is that, rather than completely solving the optimization problem associated with each specific $\boldsymbol{\theta}$, we take a single step along the gradient before sampling a new parameter vector. We do not know the correct output for a given input, but we can compute the loss given the current prediction of $\boldsymbol{x}$ and update such prediction in order to minimize the corresponding loss. Even though the NN is not provided with the solution for each parameter vector, descending along the gradient allows the NN to quickly generalize across parameter vectors.

Altogether, our proposition can be summarized as follows:

- Treat the optimization parameters as inputs to the NN.
- Create a custom loss function based on the parametric optimization problem.
- Evaluate the NN and update the weights by means of any gradient-based optimizer.

IV. Application to Constrained QP

Before proceeding, in follow-up letters [11] and [12], to more specific parametric optimizations motivated by wireless communications, let us herein entertain a simpler yet much more general one.

A. Formulation

A constrained QP is a convex instance of the parametric optimization in (1) with

$$f(\boldsymbol{\theta}, \boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{R} \boldsymbol{x} + \boldsymbol{b}^\top \boldsymbol{x}, \tag{12}$$

where $\boldsymbol{R}$ is a positive-semidefinite matrix and $\boldsymbol{b}$ is a vector, together constituting the parameter $\boldsymbol{\theta} = \{\boldsymbol{R}, \boldsymbol{b}\}$. The feasible

TABLE I
NN SETTINGS FOR QP.

	Input layer	Hidden layer	Output layer
Neurons	900	500	$D = 10 + 10$ or $30 + 30$
Activation function	RLU	RLU	Linear
Regularization	L2 norm $\lambda = .001$	L2 norm $\lambda = .001$	L2 norm $\lambda = .001$

set for every entry of $\boldsymbol{x}$ is the interval $[0, 1]$, ensuring that $\mathcal{X}$ is a D-dimensional convex set.

To demonstrate the functionality of the proposed approach as broadly as possible, we do not posit any structure for $\boldsymbol{R}$ and $\boldsymbol{b}$. Rather, $\boldsymbol{R}$ is generated as

$$\boldsymbol{R} = \sum_{i=0}^{I-1} \boldsymbol{r}_i \boldsymbol{r}_i^\top \tag{13}$$

with the entries of $\boldsymbol{r}_i$, as well as those of $\boldsymbol{b}$, drawn uniformly within $[-1, 1]$. To solve this parametric optimization using the introduced framework, the loss function is defined as

$$L_{\text{QP}} = \underbrace{\boldsymbol{x}^\top \boldsymbol{R} \boldsymbol{x} + \boldsymbol{b}^\top \boldsymbol{x}}_{\text{Objective}} + 100 \underbrace{\sum_{k=0}^{D-1} \Big([-x_k]^+ + [x_k - 1]^+ \Big)}_{\text{Constraints}}, \tag{14}$$

where $[z]^+ = \max(0, z)$ and the multiplier of 100 applied to the constraint portion is the result of cross-validation. By increasing it, the (very small) probability that the constraints are violated and that the solution thus falls outside the feasible set could be reduced further, at the expense of the objective value. This issue is tackled in [12] within the context of a wireless communications problem.

B. Learning Stage

The learning pipeline employed to train the NN is depicted in Fig. 1 and the NN's parameters are summarized in Table I. A single hidden layer is featured, as additional ones have not been found to improve the performance—yet the training does become longer. With D dimensions, there are $D \times D$ dimensional interactions and thus the number of neurons in the input layer should be at least D^2 while the number of neurons in the hidden layer should be between D and D^2.

After centering and scaling, $\boldsymbol{\theta}$ is fed to a feature-extracting input layer equipped with rectified linear unit (RLU) activation functions. A hidden layer then processes the data also via RLUs, and an output layer with linear activation functions generates values $\boldsymbol{x}$. To improve the dynamic range as well as the sensitivity to small values, these outputs are taken to be in log scale, yet this restricts them to being positive. This limitation is circumvented by allowing the NN to output two distinct vectors, $\boldsymbol{x}^+$ and $\boldsymbol{x}^-$, both in log scale, which are linearized into positive and negative values, respectively, and then summed. With that, the NN can produce any real vector $\boldsymbol{x}$ while preserving the advantages of an internal log

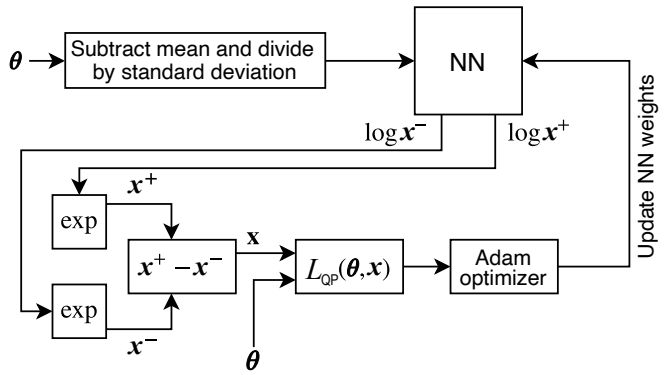

Fig. 1. Learning pipeline.

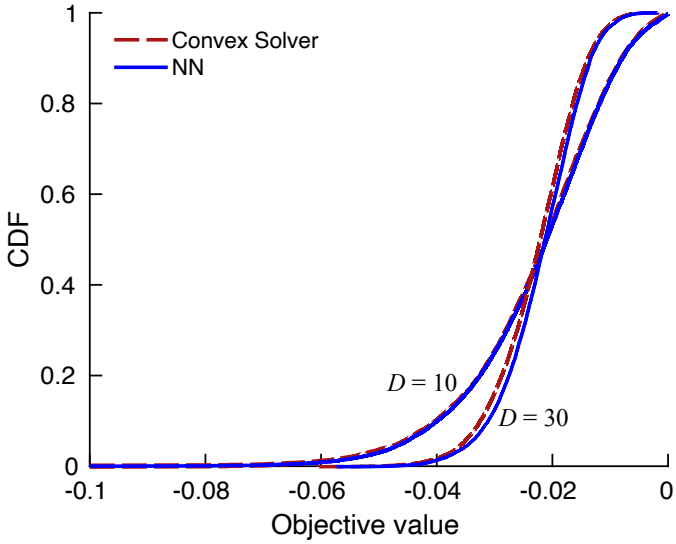

Fig. 2. CDF of $\min_{\boldsymbol{x} \in \mathcal{X}} f(\boldsymbol{\theta}, \boldsymbol{x})$ for a QP with either $D = 10$ or $D = 30$ dimensions. The distribution is induced by that of $\boldsymbol{\theta} = \{\boldsymbol{R}, \boldsymbol{b}\}$ with $I = 5$.

representation. From $\boldsymbol{\theta}$ and from the output $\boldsymbol{x}$, the loss L_{QP} is quantified and an Adam optimizer—a standard algorithm that updates NN weighs iteratively [18]—is applied to minimize it. To avoid oscillations around local optima during the weight adjustment, the learning rate, i.e., the amplitude of the gradient steps, is reduced gradually from 0.001 down to 0.0001. And, to prevent overfitting, L2-norm regularization is used in conjunction with the Adam optimizer. Precisely, a portion $\lambda = 0.001$ of the L2 norm of the weights is added to the loss.

In order to streamline the learning, rather than a single large database, we generate multiple small databases. Specifically, $M = 500$ databases of 12800 parameter realizations are generated and, over each such database, 100 updates of the NN weights take place; every update involves a randomly selected batch of 128 realizations. Altogether, $12800M$ realizations are produced for learning purposes, and the NN weights are updated $100M$ times.

C. Performance

Once the learning stage has concluded, the inference of solutions can take place. To evaluate the performance, an ensemble of realizations of the parameter $\boldsymbol{\theta}$ is produced; each

realization is fed to the NN, which outputs the corresponding inference of the QP solution. Depicted in Fig. 2 is the cumulative distribution function (CDF) of the objective value, $\min_{\boldsymbol{x}\in\mathcal{X}} f(\boldsymbol{\theta},\boldsymbol{x})$, for a constrained QP with either $D = 10$ or $D = 30$ dimensions and with $I = 5$ in (13). The match between the solutions produced by the NN and by a convex solver [19] is very satisfying for $D = 30$, when the number of neurons in the input layer is tight at its minimum value of D^2, while the match is absolute for $D = 10$, when excess neurons are available therein.

D. Computational Cost

Relative to a convex solver, the computational cost of the unsupervised-learning approach scales better with the dimensionality of the QP. Likewise, its training scales better than it would if the learning were supervised, as that would require solving the QP for each individual training parameter.

For the QP herein invoked to illustrate the performance, however, the computational savings and the improvement in scalability are modest. It is for more intricate problems such as the ones tackled in [11], [12] that the advantage becomes prominent, hence its assessment is deferred to these sequels.

V. Conclusion

An unsupervised NN-based procedure to tackle parametric optimizations over convex sets has been presented and put to use in the classic problem of constrained QP. The results match those produced by convex solvers with extreme accuracy. In the follow-up letters [11] and [12], more involved problems motivated by wireless communications are tackled and major advantages in computational scalability are revealed relative to convex solvers. For some of these problems, furthermore, the feasible set is convex but the objective function is not.

We believe that the restriction of the feasible set being convex could be lifted, at least in those cases in which the existence of a local minimizer can be established [20], and this is an interesting avenue for subsequent research.

Also suggestive of further work is that, while computationally more scalable than convex solvers (when applicable) and appearing to perform well even with nonconvex objectives, the presented approach still suffers from weaknesses:

- The NN is fully connected, hence the number of neuronal interconnects grows rapidly with the dimensionality of the parameter and optimization vectors, and so does the number of training samples.
- Retraining needs to take place whenever those dimensions change.

The first issue can be mitigated by exploiting the structure of each problem, say the existence of entries within $\boldsymbol{\theta}$ that are zero (or small enough to be negligible); this is often the case in the problems confronted in [11], [12]. Then, a non-fully-connected NN could be employed, at the expense of generality.

As of the second issue, one idea would be to dimension the problem for its largest possible size and then train with some of the dimensions randomly zeroed out [21]. Alternatively, a modular NN could be considered, and interesting ideas in this direction are propounded in [22], [23], also in the context of wireless communications. Modular NN designs that could be applied to generic parametric optimization would be a welcome proposition.

Acknowledgment

The invitation from the Editor-in-Chief, the handling of the Associated Editor, and the constructive feedback from the anonymous reviewers, as well as from Dr. Giovanni Geraci, are all gratefully acknowledged.

References

[1] A. V. Fiacco, *Introduction to Sensitivity and Stability Analysis in Nonlinear Programming*, Academic Press, New York, 1983.

[2] G. Bank, J. Guddat, D. Klatte, B. Kummer, and D. Tammer, *Non-linear Parametric Optimization*, Birkhauser Verlag, Basel, 1983.

[3] F. Bonnans and A. Shapiro, *Perturbation analysis of optimization problems*, Springer Series in Operations Research, Springer-Verlag, New York, 2000.

[4] D. Klatte and B. Kummer, *Nonsmooth equations in Optimization*, Kluwer Ac. Publishers Dortrecht, 2002.

[5] J. Acevedo and E. N. Pistikopoulos, "A parametric MINLP algorithm for process synthesis problems under uncertainty," *Ind. Eng. Chem. Res.*, vol. 35, pp. 147–158, 1996.

[6] V. Dua and E. N. Pistikopoulos, "An outer-approximation algorithm for the solution of multiparametric MINLP problems," *Computers them. Engng*, vol. 22, pp. S955–S958, 1998.

[7] A. Rantzer, "Dynamic dual decomposition for distributed control," in *Proceedings of the American Control Conference*, 2009, p. 884–888.

[8] P. Patrinos and H. Sarimveis, "Convex parametric piecewise quadratic optimization: Theory and algorithms," *Automatica*, vol. 47, pp. 1770–1777, 2011.

[9] L. Sanguinetti, A. Zappone, and M. Debbah, "Deep learning power allocation in massive MIMO," in *Asilomar Conf. Signals, Systems, and Computers*, 2018, pp. 1257–1261.

[10] T. Van Chien, E. Björnson, and E. G. Larsson, "Sum spectral efficiency maximization in massive MIMO systems: Benefits from deep learning," *CoRR*, vol. abs/1903.08163, 2019.

[11] R. Nikbakht, A. Jonsson, and A. Lozano, "Unsupervised learning for cellular power control," *IEEE Commun. Letters*, vol. 24, 2020.

[12] R. Nikbakht, A. Jonsson, and A. Lozano, "Unsupervised learning for C-RAN power control and power allocation," *IEEE Commun. Letters*, vol. 24, 2020.

[13] G. Cybenko, "Approximations by superpositions of sigmoidal functions," *Mathematics of Control, Signals, and Systems*, vol. 2(4), pp. 303–314, 1989.

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

[15] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural Networks*, vol. 4(2), pp. 251–257, 1991.

[16] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323(6088), pp. 533–536, 1986.

[17] F. Liang, C. Shen, W. Yu, and F. Wu, "Towards optimal power control via ensembling deep neural networks," *IEEE Trans. Commun.*, vol. 68, no. 3, pp. 1760–1776, 2019.

[18] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[19] M. Grant and S. Boyd, "CVX: Matlab software for disciplined convex programming, version 2.1," http://cvxr.com/cvx, Mar. 2014.

[20] G. Still, "Lectures on parametric optimization: An introduction," *Optimization Online*, 2018.

[21] M. Khani, M. Alizadeh, J. Hoydis, and P. Fleming, "Adaptive neural signal detection for massive MIMO," *IEEE Trans. Wireless Commun.*, 2020.

[22] W. Cui, K. Shen, and W. Yu, "Spatial deep learning for wireless scheduling," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 6, pp. 1248–1261, 2019.

[23] W. Lee, M. Kim, and D.-H. Cho, "Deep power control: Transmit power control scheme based on convolutional neural network," *IEEE Commun. Letters*, vol. 22, no. 6, pp. 1276–1279, 2018.