# Learning Convex Optimization Models

Akshay Agrawal, Shane Barratt, and Stephen Boyd, *Fellow, IEEE*

*Abstract*—A convex optimization model predicts an output from an input by solving a convex optimization problem. The class of convex optimization models is large, and includes as special cases many well-known models like linear and logistic regression. We propose a heuristic for learning the parameters in a convex optimization model given a dataset of input-output pairs, using recently developed methods for differentiating the solution of a convex optimization problem with respect to its parameters. We describe three general classes of convex optimization models, maximum a posteriori (MAP) models, utility maximization models, and agent models, and present a numerical experiment for each.

*Index Terms*—Convex optimization, differentiable optimization, machine learning.

## I. INTRODUCTION

### A. Convex Optimization Models

WE consider the problem of learning to predict outputs $y \in \mathcal{Y}$ from inputs $x \in \mathcal{X}$, given a set of input-output pairs $\mathcal{D} = \{(x^i, y^i)\}_{i=1}^N \subseteq (\mathcal{X} \times \mathcal{Y})^N$. We assume that $\mathcal{Y} \subseteq \mathbf{R}^m$ is a convex set, but make no assumptions on $\mathcal{X}$. In this paper, we specifically consider models $\phi : \mathcal{X} \to \mathcal{Y}$ that predict the output $y$ by solving a convex optimization problem that depends on the input $x$. We call such models *convex optimization models*. While convex optimization has historically played a large role in fitting machine learning models, we emphasize that in this paper, we solve convex optimization problems to perform *inference*.

A convex optimization model has the form

$$\phi(x; \theta) = \underset{y \in \mathcal{Y}}{\operatorname{argmin}} E(x, y; \theta) \tag{1}$$

where the objective function $E : \mathcal{X} \times \mathcal{Y} \to \mathbf{R} \cup \{+\infty\}$ is convex in its second argument, and $\theta \in \Theta$ is a parameter belonging to a set of allowable parameters $\Theta$. The objective function $E$ is the model's *energy function*, and the quantity $E(x, y; \theta)$ is the energy of $y$ given $x$ ; the energy $E(x, y; \theta)$ can depend arbitrarily on $x$ and $\theta$, as long as it is convex in $y$ . Infinite values of $E$ encode additional constraints on the prediction, since $E(x, y; \theta) = +\infty$ implies $\phi(x; \theta) \neq y$. Evaluating a convex optimization model at $x$ corresponds to finding an output $y \in \mathcal{Y}$ of minimum energy. The function $\phi$ is in general set-valued, since the convex optimization problem in (1) may

The authors are with the Department of Electrical Engineering, Stanford University, Stanford, CA 94305 USA (e-mail: sbarratt@stanford.edu; boyd@stanford.edu; akshayka@stanford.edu).

have zero, one, or many solutions. Throughout this paper, we only consider the case where the argmin exists and is unique.

Convex optimization models are particularly well-suited for problems in which the outputs $y \in \mathcal{Y}$ are known to have structure. For example, if the outputs are probability mass functions, we can take $\mathcal{Y}$ to be the probability simplex; if they are sorted vectors, we can take $\mathcal{Y}$ to be the monotone cone; or if they are covariance matrices, we can take $\mathcal{Y}$ to be the set of symmetric positive semidefinite matrices. In all cases, convex optimization models provide an efficient way of searching over a structured set to produce predictions satisfying known priors.

Because convex optimization models can depend arbitrarily on $x$ and $\theta$, they are quite general. They include familiar models for regression and classification as specific instances, such as linear and logistic regression. In the basic examples of linear and logistic regression, the corresponding convex optimization models have analytical solutions. But in most cases, convex optimization models must be evaluated by iterating a numerical algorithm.

Learning a parametric model requires tuning the parameters to make good predictions on $\mathcal{D}$ and ultimately on held-out input-output pairs. In this paper, we present a gradient method for learning the parameters in a convex optimization model; this learning problem is in general non-convex, since the solution map of a convex optimization model is a complicated function. Our method uses the fact that the solution map is often differentiable, and its derivative can be computed efficiently, without differentiating through each step of the numerical solver [1]–[4].

We limit our attention to optimization models that are convex for three reasons: convex programs can be solved globally, efficiently, and reliably [5], which makes inference tractable; the solution map is often differentiable [1], [3], which makes learning tractable; and convex optimization models are general, since they can depend arbitrarily on $x$ and $\theta$ (e.g., we can take $E(x, y; \theta) = \|y - f(x, \theta)\|_2$, where $f(x, \theta)$ can be any function, such as a neural network).

*Contributions:* The contributions of our paper are the following:

1) We describe a new type of machine learning model, the convex optimization model, which makes it both possible and easy to express priors and enforce constraints on the model output.

2) We show how to fit convex optimization models using recently developed methods for differentiating through the solution map of convex programs.

3) We give several examples of convex optimization models, grouping the examples by application area, i.e., MAP inference, utility-maximizing processes, and agent models. We include numerical experiments showing how to use open-

source software to fit these models.

*Outline:* Our learning method is presented in Section II for the general case. In the following three sections, we describe special cases of convex optimization models with particular forms or interpretations. In Section III, we interpret convex optimization models as solving a maximum a posteriori (MAP) inference task, and we give examples of these MAP models in regression, classification, and graphical models. In Section IV, we show how convex optimization models can be used to model utility-maximizing processes, and in Section V, we give examples of modeling agents using the framework of stochastic control. In Section VI, we present numerical experiments of learning convex optimization problems for a number of prediction tasks.

### B. Related Work

*Structured Prediction:* Structured prediction refers to supervised learning problems where the output has known structure [6]. A common approach to structured prediction is energy-based models, which associate a scalar energy to each output, and select a value of the output that minimizes the energy, subject to constraints on the output [7]. Most energy-based learning methods are learned by reducing the energy for input-output pairs in the training set and increasing it for other pairs [8]–[11]. More recently, the authors of [12], [13] proposed a method for end-to-end learning of energy networks by unrolled optimization. Indeed, a convex optimization model can be viewed as a form of energy-based learning where the energy function is convex in the output. For example, input-convex neural networks (ICNNs) [14] can be viewed as a convex optimization model where the energy function is an ICNN. We also note that several authors have proposed using structured prediction methods as the final layer of a deep neural network [15]–[17]; of particular note is [18], in which the authors used a second-order cone program (SOCP) as their final layer.

*Inverse Optimization:* Inverse optimization refers to the problem of recovering the structure or parameters of an optimization problem, given solutions to it [19], [20]. In general, inverse optimization is very difficult. One special case where it is tractable is when the optimization problem is a linear program and the loss function is convex in the parameters [19], and another is when the optimization problem is convex and the parameters enter in a certain way [21], [22]. This paper can be viewed as a heuristic method for inverse optimization for general convex optimization problems.

*Differentiable Optimization:* There has been significant recent interest in differentiating the solution maps of optimization problems; these differentiable solution maps are sometimes called *optimization layers*. Reference [23] showed how quadratic programs can be embedded as optimization layers in machine learning pipelines, by implicitly differentiating the KKT conditions (as in the early works [24], [25]). Recently, [2], [3] showed how to efficiently differentiate through convex cone programs by applying the implicit function theorem to a residual map introduced in [4], and [1] showed how to differentiate through convex optimization problems by an automatable reduction to convex

cone programs; our method for learning convex optimization models builds on this recent work. Optimization layers have been used in many applications, including control [26]–[29], game-playing [30], [31], computer graphics [18], combinatorial tasks [32], automatic repair of optimization problems [33], and data fitting more generally [14], [34]–[36]. Differentiable optimization for nonconvex problems is often performed numerically by differentiating each individual step of a numerical solver [37]–[40], although sometimes it is done implicitly; see, e.g., [26], [41], [42].

## II. LEARNING CONVEX OPTIMIZATION MODELS

In this section we describe a general method for learning the parameter $\theta$ in a convex optimization model, given a data set consisting of input-output pairs $(x^1, y^1), \ldots, (x^N, y^N) \in \mathcal{X} \times \mathcal{Y}$. We let $\hat{y}^i = \phi(x^i; \theta)$ denote the prediction of $y^i$ based on $x^i$, for $i = 1, \ldots, N$. These predictions depend on $\theta$, but we suppress this dependency to lighten the notation.

### A. Learning Problem

The fidelity of a convex optimization model's predictions is measured by a loss function $L : \mathcal{Y} \times \mathcal{Y} \to \mathbf{R}$. The value $L(\hat{y}^i, y^i)$ is the loss for the $i$th data point; the lower the loss, the better the prediction. Through $\hat{y}^i$, this depends on the parameter $\theta$. Our ultimate goal is to construct a model that generalizes, i.e., makes accurate predictions for input-output pairs not present in $\mathcal{D}$. To this end, we first partition the data pair indices into two sets, a training set $\mathcal{T} \subset \{1, \ldots, N\}$ and a validation set $\mathcal{V} = \{1, \ldots, N\} \setminus \mathcal{T}$. We define the average training loss as

$$\mathcal{L}(\theta) = \frac{1}{|\mathcal{T}|} \sum_{i \in \mathcal{T}} L(\hat{y}^i, y^i).$$

We fit the model by choosing $\theta$ to minimize the average training loss plus a regularizer $R : \Theta \to \mathbf{R} \cup \{\infty\}$, i.e., solving the optimization problem

$$\text{minimize } \mathcal{L}(\theta) + R(\theta) \qquad (2)$$

with variable $\theta$. The regularizer measures how compatible $\theta$ is with prior knowledge, and we assume that $R(\theta) = \infty$ for $\theta \notin \Theta$, i.e., the regularizer encodes the constraint $\theta \in \Theta$. We describe below a gradient-based method to (approximately) solve the problem (2).

We can check how well a convex optimization model generalizes by computing its average loss on the validation set

$$\mathcal{L}^{\text{val}}(\theta) = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} L(\hat{y}^i, y^i).$$

In some cases, the model or learning procedure depends on parameters other than $\theta$, called hyper-parameters. It is common to learn multiple models over a grid of hyper-parameter values and use the model with the lowest validation loss.

### B. A Gradient-Based Learning Method

In general, $\mathcal{L}$ is not convex, so we must resort to an approximate or heuristic method for learning the parameters. One could consider zeroth-order methods, e.g., evolutionary

strategies [43], Bayesian optimization [44], or random search [45]. Instead, we use a first-order method, taking advantage of the fact that the convex optimization model is often differentiable in the parameter $\theta$.

*Differentiation:* The output of a non-pathological convex optimization model is an implicit function of the input $x$ and the parameter $\theta$. When some regularity conditions are satisfied, this implicit function is differentiable, and its derivative with respect to $\theta$ can often be computed in less time than is needed to compute the solution [2]. One generic way of differentiating through convex optimization problems involves a reduction to an equivalent convex cone program, and implicit differentiation of a residual map of the cone program [2]; this is the method we use in this paper. For readers interested in more details on the derivative computation, we suggest [1], [2], [4]. In our experience, it is unnecessary to check regularity conditions, since we and others have empirically observed that the derivative computation in [2] usually provides useful first-order information in the rare cases when the solution map is not differentiable at the current iterate [1], [29]. In this sense, convex optimization models are similar to other kinds of machine learning models, such as neural networks, which can be trained using gradient descent despite only being differentiable almost everywhere.

*Learning Method:* We propose a proximal stochastic gradient method. The method is iterative, starting with an initial parameter $\theta^1$. The first step in iteration $k$ is to choose a batch of (training) data denoted $\mathcal{B}^k \subset \mathcal{T}$. There are many ways to do this, e.g., by cycling through the training set or by selecting at random. The next step is to compute the gradient of the loss averaged over the batch

$$g^k = \frac{1}{|\mathcal{B}^k|} \sum_{i \in \mathcal{B}^k} \nabla_\theta L(\hat{y}^i, y^i).$$

This step requires applying the chain rule for differentiation to $|\mathcal{B}^k|$ compositions of the convex optimization model (discussed above) and the loss function. The final step is to update $\theta$ by first taking a step in the negative gradient direction, and then applying the proximal operator of $R$

$$\theta^{k+1} = \mathbf{prox}_{t^k R}(\theta^k - t^k \nabla g^k)$$
$$= \mathrm{argmin}_{\theta \in \Theta} R(\theta) + \frac{1}{2t^k} \|\theta - \theta^k + t^k \nabla g^k\|_2^2$$

where $t^k > 0$ is a step size. We assume that the proximal operator of $R$ is single-valued and easy to evaluate. When $R(\theta)$ is the $\{0, \infty\}$ indicator function of $\Theta$, this method reduces to the standard projected stochastic gradient method

$$\theta^{k+1} = \Pi_\Theta(\theta^k - t^k \nabla g^k)$$

where $\Pi_\Theta$ is the Euclidean projection operator onto $\Theta$. There are many ways to select the step sizes $t^k$; see, e.g., [46]–[48].

## III. MAP MODELS

Let the inputs $x \in \mathcal{X}$ and outputs $y \in \mathcal{Y}$ be random vectors, and suppose that the conditional distribution of $y$ given $x$ has a log-concave density $p$, parametrized by $\theta$. The energy function

$$E(x, y; \theta) = -\log p(y \mid x; \theta)$$

yields a *maximum a posteriori* (MAP) model: $\hat{y} = \phi(x; \theta)$ is the MAP estimate of the random vector $y$, given $x$ [49, §1.2.5]. Conversely, any convex optimization model can be interpreted as a MAP model, by identifying the density of $y$ given $x$ with an exponential transformation of the negative energy

$$p(y \mid x; \theta) = \frac{1}{Z(x; \theta)} \exp(-E(x, y; \theta))$$

where $Z$ is the normalizing constant or partition function

$$Z(x; \theta) = \int_{y \in \mathcal{Y}} \exp(-E(x, y; \theta)).$$

Crucially, evaluating a MAP model does not require computing $Z(x; \theta)$ since it does not depend on $y$; i.e., MAP models can be used even when the partition function is computationally intractable, as is often the case [50, §18].

### A. Regression

Several basic regression models can be described as MAP models, with

$$p(y \mid x; \theta) \propto \exp(-f(\theta^T x - y))$$

where $x \in \mathcal{X} = \mathbf{R}^n$, $y \in \mathcal{Y} = \mathbf{R}^m$, $\theta \in \mathbf{R}^{n \times m}$ is the parameter and $f : \mathbf{R}^m \to \mathbf{R}$ is a convex penalty function (The expression $\theta^T x$ can be replaced with a more complex function, such as a neural network, since convex optimization models can depend arbitrarily on $x$ and $\theta$; we focus on the linear case for simplicity). If the penalty $f$ is minimized at 0, then the MAP model is the linear predictor $\phi(x; \theta) = \theta^T x$. In this case, fitting the MAP model with a mean-squared loss $L(\hat{y}, y) = \|\hat{y} - y\|_2^2$ is equivalent to fitting a linear regression model; fitting it with an $\ell_1$ loss is equivalent to $\ell_1$ regression; and fitting it with the Huber loss [5, §6.1] yields robust Huber regression.

These very basic examples can be made more interesting by constraining the outputs $y$ to lie in a convex subset $C$ of $\mathcal{Y}$, using a density of the form

$$p(y|x) \propto \begin{cases} \exp(-f(\theta^T x - y)) & y \in C \\ 0 & \text{otherwise.} \end{cases}$$

Because the output is constrained, different choices of the penalty function $f$ yield different MAP models. When the penalty function $f$ is the squared Euclidean norm, $f(u) = \|u\|_2^2$, the MAP estimate $\hat{y} = \phi(x; \theta)$ is the Euclidean projection of $\theta^T x$ onto $C$. Other penalty functions, like the $\ell_1$ norm $f(u) = \|u\|_1$ or the Huber function [5, §6.1] yield interesting non-trivial regression models. We present some examples of the constraint set $C$ below.

*Nonnegative Regression:* Taking $C = \mathbf{R}_+^m$ (the set of nonnegative $m$-vectors) yields a MAP model for nonnegative regression, i.e., the MAP estimates in this model are guaranteed to be nonnegative.

*Monotonic Output Regression:* When $C$ is the monotone cone, i.e., the set of ordered vectors

$$C = \{y \in \mathbf{R}^m \mid y_1 \le y_2 \le \cdots \le y_m\}$$

the MAP estimates in the regression model are guaranteed to be sorted in ascending order. When $f$ is the Euclidean norm, the MAP estimate is the projection of $\theta^T x$ onto the monotone

cone, and evaluating it requires solving a convex quadratic program (QP); in this special case, once $\theta^T x$ has been computed (which takes $O(mn)$ time), evaluating the convex optimization model is equivalent to monotonic or isotonic regression [51], which takes $O(m)$ time [52], meaning it has the same complexity as the standard linear regression model.

We note the distinction between traditional isotonic regression [51] and a convex optimization model with monotone constraint. In isotonic regression, we seek a single vector with nondecreasing components. In a convex optimization model with a monotone constraint, we seek a model that maps $x \in X$ to a prediction $\hat{y}$ that always has nondecreasing components.

### B. Classification

In (probabilistic) classification tasks, the outputs are vectors in the probability simplex, i.e.,

$$\mathcal{Y} = \Delta^{m-1} = \{y \in \mathbf{R}^m \mid \mathbf{1}^T y = 1, \ y \geq 0\}.$$

The output $y$ can be interpreted as a probability distribution over $\{1, \ldots, m\}$ associated with an input $x \in X = \mathbf{R}^n$. The MAP estimate $\hat{y} = \phi(x; \theta)$ is therefore the most likely distribution associated with $x$, under a particular density $p(y \mid x; \theta)$. This includes as a special case the familiar setting in which each output is a label, e.g., a number in $\{1, \ldots, m\}$, since the label $k$ can be represented by a vector $y$ such that $y_k = 1$ and $y_i = 0$ for $i \neq k$.

As a simple first example, consider the MAP model with density

$$p(y|x;\theta) \propto \begin{cases} \exp(x^T \theta y + H(y)) & y \in \Delta^{m-1} \\ 0 & \text{otherwise} \end{cases}$$

where $\theta \in \mathbf{R}^{n \times m}$ and $H(y) = -\sum_{i=1}^{m} y_i \log y_i$ is the entropy function. The resulting convex optimization model is just the softmax of $\theta^T x$, i.e., $\phi(x; \theta) = \exp(\theta^T x)/\mathbf{1}^T \exp(\theta^T x)$, where the exponentiation and the division are meant elementwise (This fact is readily verified via the KKT conditions of the convex optimization model [3, §2.4.4]).

Since the outputs are probability distributions, a natural loss function is the KL-divergence from the true output $y$ to the prediction $\hat{y} = \phi(x; \theta)$, i.e.,

$$L(\hat{y}, y) = \sum_{i=1}^{m} y_i \log(y_i/\hat{y}_i) = \sum_{i=1}^{m} y_i \log y_i - y_i \log \hat{y}_i.$$

Discarding the constant terms $y_i \log y_i$, which do not affect learning, recovers the commonly used cross-entropy loss [53, §2.6]. Using this loss function with the softmax model recovers multinomial logistic regression [53, §4.4]. This model can be made more interesting by simple extensions.

*Constrained Logistic Regression:* We can readily add constraints on the distribution $\hat{y}$. As a simple example, a box-constrained logistic regression model has the form

$$\phi(x; \theta) = \mathrm{argmin}_y \quad -x^T \theta y - H(y)$$
$$\text{s.t.} \quad y \in C$$

where $C$ is a convex subset of $\Delta^{m-1}$. There are many interesting constraints we can impose on the distribution $y$. As

a simple example, the constraint set

$$C = \{y \in \Delta^{m-1} \mid \alpha \leq y \leq \beta\}$$

where $\alpha, \beta \in \mathbf{R}^m$ are vectors and the the inequalities are meant elementwise can be used to require that $\hat{y}$ have heavy tails, by making the leading and trailing components of $\alpha$ large, or thin tails, by making the leading and trailing components of $\beta$ small. Another simple example is to specify the expected value of an arbitrary function on $\{1, \ldots, m\}$ under $\hat{y}$, which is a simple linear equality constraint on $\hat{y}$. More generally, any affine equality constraints and convex inequality constraints on $\hat{y}$ may be imposed; these include constraints on the quantiles of the random variable associated with $y$, lower bounds on its variance, and inequality constraints on conditional probability distributions.

*Piecewise-Constant Logistic Regression:* A piecewise-constant logistic regression model has the form

$$\phi(x; \theta) = \mathrm{argmin}_y \quad -x^T \theta y - H(y) + r(y)$$
$$\text{s.t.} \quad y \in \mathcal{Y}$$

where the parameter is $\theta$,

$$r(y) = \lambda \sum_{i=1}^{m-1} |y_{i+1} - y_i|$$

and $\lambda > 0$ is a (hyper-) parameter. To the standard energy we add a total variation term that encourages $y$ to have few "jumps", i.e., few indices $i$ such that $y_i \neq y_{i+1}$, $i = 1, \ldots, m-1$ [54, §7.4]. The larger the hyper-parameter $\lambda$ is, the fewer jumps it will have.

### C. Graphical Models

A Markov random field (MRF) is an undirected graphical model that describes the joint distribution of a set of random variables, which are represented by the nodes in the graph. An MRF associates parametrized potential functions to cliques of nodes, and the joint distribution it describes is proportional to the product of these potential functions. MRFs are commonly used for structured prediction, but learning their parameters is in general difficult [49, §8.3]. When the potential functions are log-concave, however, we can fit the parameters using the methods described in this paper.

Suppose we are given an MRF describing the joint distribution of the random vectors $x$ and $y$. Let $z = (x, y) \in \mathbf{R}^{n+m}$, and let $c_1, c_2, \ldots, c_p$ denote the indices of the graph cliques; we write $z_{c_k}$ to denote the components of $z$ in clique $c_k$. For example, if $c_k = (1, 4, 5)$, then $z_{c_k} = (z_1, z_4, z_5)$. Suppose the MRF has a Boltzmann distribution, meaning

$$p(y \mid x; \theta) \propto \exp(-(E_1(z_{c_1}) + E_2(z_{c_2}) + \cdots + E_p(z_{c_p}))).$$

Here, $\exp(-E_k(z_{c_k}))$ are the potential functions, and $E_k$ is a local energy function, parametrized by $\theta$, for the clique $k$. As long as the functions $E_1, \ldots, E_p$ are convex, the corresponding MAP model

$$\phi(x; \theta) = \mathrm{argmax}_{y \in \mathcal{Y}} \log p(y \mid x; \theta)$$

is a convex optimization model. In this case, given a dataset of input-output pairs $(x, y)$, we can fit the parameter $\theta$ without

evaluating or differentiating through the partition function.

*Quadratic MRFs:* Consider an MRF in which the variables $x$ and $y$ lie in convex sets (such as slabs, or all of $\mathbf{R}^n$ or $\mathbf{R}^m$). Suppose the MRF has $\binom{n+m}{2} + n + m$ pairwise cliques of the form $\{z_i, z_j\}$ ($1 \leq i \leq j \leq n + m$), and a Boltzmann distribution with local energy functions

$$E_{(i,j)}(z_i, z_j) = \theta_{ij} z_i z_j, \quad 1 \leq i \leq j \leq n + m$$

where $\theta \in \Theta = \mathbf{S}_+^{n+m}$ is the parameter ($\mathbf{S}_+^{n+m}$ is the set of positive semidefinite matrices). The MAP inference task for this MRF is a convex optimization model, of the form

$$\phi(x;\theta) = \underset{y \in \mathcal{Y}}{\operatorname{argmax}} -z^T \theta z = \underset{y \in \mathcal{Y}}{\operatorname{argmin}} \, z^T \theta z.$$

MRFs with a similar clique structure have been proposed for various signal and image denoising tasks. We give a numerical example of fitting a quadratic MAP model of an MRF in Section VI.

We emphasize that the dependence on $x$ can be arbitrary; e.g., if the energy function were

$$E(x, y; \theta) = (f(x), y)^T \theta (f(x), y))$$

where $f$ were a neural network, the MAP model would remain convex.

## IV. UTILITY MAXIMIZATION MODELS

We now consider the case where the output $y$ is a decision, and the input $x$ is a context or feature vector that affects the decision. We assume that the decision $y$ is chosen to maximize some given parametrized utility function

$$U : \mathcal{X} \times \mathcal{Y} \to \mathbf{R} \cup \{-\infty\}$$

where $U(x, y; \theta)$ is the utility of choosing a decision $y$ given the context $x$ and the parameters $\theta$, and is concave in $y$ (Infinite values of $U$ are used to constrain the decision $y$). The energy function in a utility maximization model is simply the negative utility

$$E(x, y; \theta) = -U(x, y; \theta).$$

The resulting convex optimization model $\phi(x; \theta)$ gives a maximum utility decision in the context $x$. The same losses used for regression (see Section III-A) and classification (see Section III-B) can be used for utility maximization. The context $x$ might include, for example, a total budget on the decision $y$, prices that affect the decision, or availabilities that affect the decision.

*Resource Allocation:* A standard example of utility maximization is resource allocation. In the simplest case, this involves allocating a single, finite resource across $m$ agents or tasks. The decision $y \in \mathbf{R}_+^m$ gives the allocation across those tasks, where $y_i$ is the resource allocated to task $i$; because the resource is finite, the allocation must satisfy $\mathbf{1}^T y \leq B$, where $B \in \mathbf{R}_+$ is a nonnegative budget. The context $x$ contains the budget $B$, and possibly other important parameters such as limits on allocations to the tasks. When the input $x$ is just the budget, the utility has the form

$$U(x, y; \theta) = \begin{cases} U(y; \theta) & y \geq 0, \quad \mathbf{1}^T y \leq B \\ -\infty & \text{otherwise} \end{cases}$$

where $U(y; \theta)$ is some parametrized concave utility function, describing the utility of an allocation. In this simple case, $\phi(x; \theta)$ gives the maximum utility allocation that satisfies the budget constraint.

The input $x$ is not limited to just the budget; it can also contain additional context that affects or constrain the decision. One important case is when the resource to be allocated is dollars, and $x$ contains the prices of the resource for each of the agents, denoted $p \in \mathbf{R}_{++}^m$. When there are prices, an allocation of $y_i$ dollars provides $y_i / p_i$ units of some good to agent $i$. The utility in this case has form

$$U(x, y; \theta) = \begin{cases} U(y/p; \theta) & y \geq 0, \quad \mathbf{1}^T y \leq B \\ -\infty & \text{otherwise} \end{cases}$$

where the division is meant elementwise, and $U(z; \theta)$ gives the utility of the agents receiving $z_i$ units of the resource, $i = 1, \ldots, m$. The resulting convex optimization model $\phi(x; \theta)$ gives the maximum utility allocation that satisfies the budget constraints, given the current prices.

We can just as well model the allocation of multiple resources, each with its own budget, across agents or tasks; e.g., we might model the allocation of computational resources, such as cpu cores, memory, and disk space, to a pool of tasks. If there are $k$ resources and $m'$ agents, then the output would be the $k$ allocation vectors for each resource, stacked together to form a vector $y \in \mathbf{R}_+^m$, where $m = km'$.

*Utility Functions:* A simple family of utility functions are the separable functions

$$U(y; \theta) = \sum_{i=1}^{m} U_i(y_i; \theta)$$

where $U_i(y_i; \theta)$ is the utility of allocating $y_i$ of the resource to the $i$th agent or task. In this case the entries of the decision $y$ are coupled by budget constraints. A simple example for separable utility is exponential utility $U_i(y_i; \theta) = -\exp(\theta_i y_i)/\theta_i$.

However, $U$ need not be separable. A common example is when $y$ represents an allocation of dollars in a portfolio of stocks; the Markowitz utility or risk-adjusted return is

$$U(y; \theta) = \mu^T y - \gamma y^T \Sigma y$$

where $\mu \in \mathbf{R}^m$ is the expected return of each investment, $\Sigma \in \mathbf{S}_{++}^m$ is the covariance of the returns and $\gamma > 0$ is a risk aversion parameter. We can take $\theta = (\mu, \Sigma, \gamma)$, in which case we are observing portfolios and attempting to infer the mean covariance, and risk aversion parameter that best model the observed portfolio allocations.

## V. STOCHASTIC CONTROL AGENT MODELS

In this section, we consider a setting in which $x \in \mathcal{X} = \mathbf{R}^n$ is the context or state of a dynamical system, and $y \in \mathcal{Y} = \mathbf{R}^m$ represents the actions taken in that state. Our goal is to model the policy, i.e., the mapping from state to action, that the agent is using. In this section, we describe generic ways to model an agent's policy with a convex optimization model. The convex optimization models we present are all instances of convex optimization policies commonly used for stochastic control [29]. When learning these models, one can use the same losses

proposed for regression (see Section III-A).

*Stochastic Control:* To motivate the models presented in this section, here, we describe a general stochastic control problem. Let $x_t$ and $y_t$ denote the state and action at time $t$. Suppose the state evolves according to the dynamics

$$x_{t+1} = f(x_t, y_t, w_t) \qquad (3)$$

where $w_t \in \mathcal{W}$ is a random variable, and the function $f : \mathbf{R}^n \times \mathbf{R}^m \times \mathcal{W} \to \mathbf{R}^n$ gives the (stochastic) dynamics of the dynamical system. Suppose also that the agent selects actions according to

$$y_t = \phi(x_t), \quad t = 0, 1, \dots \qquad (4)$$

where $\phi : \mathbf{R}^n \to \mathbf{R}^m$ is the policy, and that the agent's goal is to minimize a discounted sum of stage costs $g : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R} \cup \{+\infty\}$ over time

$$\sum_{t=0}^{\infty} \gamma^t g(x_t, y_t)$$

where $\gamma \in (0, 1]$ is a discount factor, subject to the dynamics (3) and the policy (4). It is well known (see, e.g., [55]) that an optimal policy is given by

$$\phi^\star(x) = \underset{u}{\operatorname{argmin}} \, g(x, y) + \mathbf{E}V(f(x, y))$$

where $V : \mathbf{R}^n \to \mathbf{R}$ is the cost-to-go function, which satisfies Bellman's equation

$$V(x) = \inf_u g(x, y) + \mathbf{E}V(f(x, y)), \quad x \in \mathbf{R}^n. \qquad (5)$$

In general, given a dataset describing an agent's actions, we have no reason to believe that the agent chooses actions by solving a stochastic control problem. Nonetheless, choosing a model that corresponds to a policy for stochastic control can work well in practice. As we will see, our models involve learning the parameters in three functions that can be interpreted as dynamics, stage costs, and an approximate value function.

*Approximate Dynamic Programming (ADP):* One possible model of agent behavior is the ADP model [55, §6], which has the form

$$\phi(x; \theta) = \operatorname{argmin}_y \quad g(x, y; \theta) + \hat{V}(x_+; \theta)$$

$$\text{s.t.} \quad x_+ = f(x, y; \theta)$$

where $x_+ \in \mathbf{R}^n$ and $y \in \mathbf{R}^m$ are the variables. The function $f : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}^n$, which must be affine in its second argument, can be interpreted as the dynamics; the function $g : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R} \cup \{+\infty\}$ is the stage cost (which is convex in its second argument); and the convex function $\hat{V}$ can be interpreted as an approximation of the cost-to-go or value function. All three of these functions are parametrized by the vector $\theta$. The value $\hat{y} = \phi(x; \theta)$ is the optimal value of the variable $y$, i.e., the ADP model chooses the action that minimizes the current stage cost plus an estimate of the cost-to-go of the next state.

One reasonable parametrized stage cost $g$ is the weighted sum of a number of convex functions $h_1, \dots, h_p : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R} \cup \{+\infty\}$

$$g(x, y; \theta) = \sum_{i=1}^{p} \theta_i h_i(x, y).$$

In this case we would have $\Theta = \mathbf{R}_+^p$. For example, if the dynamical system were a car, the state was the physical state of the car, and the action was the steering wheel angle and the acceleration, there would be many reasonable costs: e.g., tracking, fuel use, and comfort. Such a stage cost could be used to trade off these costs, or to derive them from data.

Similarly, the cost-to-go function might be a weighted sum of functions $\hat{V}_1, \dots, \hat{V}_p : \mathbf{R}^n \to \mathbf{R} \cup \{\infty\}$

$$\hat{V}(x; \theta) = \sum_{i=1}^{p} \theta_i \hat{V}_i(x)$$

e.g., taking

$$\hat{V}_{i+jn}(x) = x_i x_j, \quad i = 1, \dots, n, \quad j = 1, \dots, n$$

yields a quadratic cost-to-go function.

*Model Predictive Control (MPC):* An MPC policy is an instance of the ADP policy [55, §6.4.3]

$$\phi(x; \theta) = \operatorname{argmin} \sum_{t=0}^{T-1} g_t(x_t, y_t; \theta)$$

$$\text{s.t.} \quad x_{t+1} = f_t(x_t, y_t; \theta)$$

$$x_0 = x$$

with variables $x_0, \dots, x_T$ and $y_0, \dots, y_{T-1}$, where $T$ is the time horizon. Here $f_t : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R}^n$ is the (affine) dynamics function at time $t$, and $g_t : \mathbf{R}^n \times \mathbf{R}^m \to \mathbf{R} \cup \{+\infty\}$ is the stage cost function at time $t$, which is convex in $y_t$; both functions are parametrized by $\theta$ (The expression $\sum_{t=1}^{T-1} g_t(x_t, y_t; \theta)$ can be interpreted as the approximate value function of an ADP policy). The objective is the sum of the stage costs $g_t$ through time, and the constraints enforce the dynamics and the initial state. The MPC model chooses the action as the first action in a planned sequence of future actions $y_0, \dots, y_{T-1}$, i.e., $\hat{y} = \phi(x; \theta)$ is the optimal value for the variable $y_0$.

## VI. NUMERICAL EXPERIMENTS

In this section we present four numerical experiments that mirror the examples from Sections III–V. The code for all of these examples can be found online[1].

*Monotonic Output Regression:* We consider the monotonic output regression model (see Section III-A). We take $n = 20$ and $m = 10$. We generate a true parameter $\theta^{\text{true}} \in \mathbf{R}^{n \times m}$ with entries sampled independently from a standard normal distribution, and sample 100 training data pairs and 50 validation data pairs according to

$$x \sim \mathcal{N}(0, I), \quad y = \phi(x + z; \theta^{\text{true}}), \quad z \sim \mathcal{N}(0, I).$$

We compare the convex optimization model to linear regression, using the standard sum of squares loss and no regularizer. The results for both of these methods are displayed in Fig. 1. On the left, we show the validation loss versus training iteration. The final validation loss for linear regression is 3.375, for the convex optimization model is
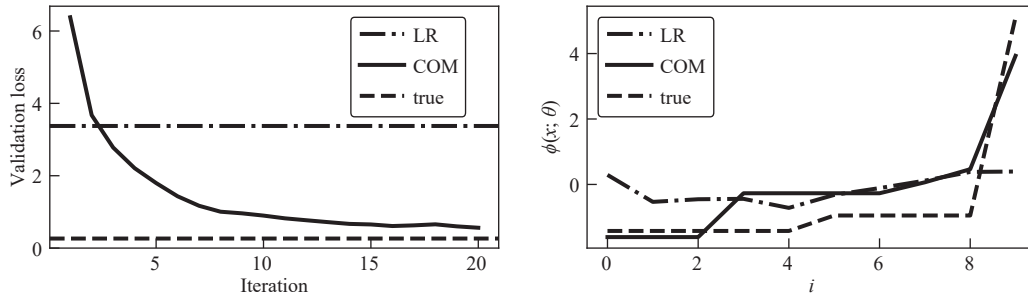
[1] redacted

Fig. 1.   Monotonic output regression: linear regression (LR), convex optimization model (COM), true model (true). Left: validation loss. Right: predictions for a held-out input.
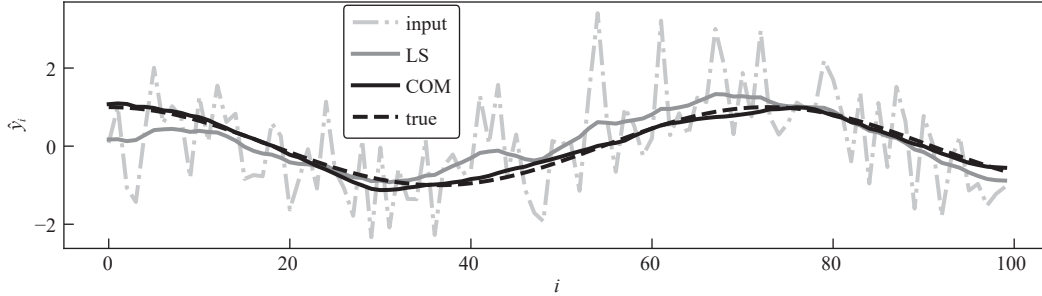


Fig. 2.   Signal denoising. Predictions for a held-out input; least squares (LS), convex optimization model (COM), and true output (true).

0.562, and for the true model is 0.264. We also calculated the validation loss of a convex optimization model with the linear regression parameters; this resulted in a validation loss of 1.511. While better than 3.375, this shows that here our learning method is superior to learning the parameters using linear regression and then projecting the outputs onto the monotone cone. On the right, we show both model's predictions for a validation input.

*Signal Denoising:* Here, we fit the parameters in a quadratic MRF (see Section III-C) for a signal denoising problem. We consider a denoising problem in which each input $x \in \mathbf{R}^n$ is a noise-corrupted observation of an output $y \in \mathbf{R}^m$, where $n = m$. The goal is to reconstruct the original signal $y$, given the noise-corrupted observation. We model the conditional density of $y$ given $x$ as

$$p(y \mid x; \theta) \propto \exp\left(-\left(\|M(x-y)\|_2^2 + \lambda\|Dy\|_2^2\right)\right)$$

where $\theta = (M \in \mathbf{R}^{n \times n}, \lambda \in \mathbf{R}_{++})$ is the parameter and $D$ is the first-order difference matrix. The MAP estimate of $y$ given $x$ is $\phi(x; \theta) = \text{argmax}_y \log p(y \mid x; \theta)$, and the corresponding convex optimization model has the energy function

$$E(x, y; \theta) = \|M(x-y)\|_2^2 + \lambda\|Dy\|_2^2.$$

The first term says that $x$ should be close to $y$, as measured by the squared quadratic $M$-norm, while the second term says that the entries of $y$ should vary smoothly. When $M = I$, this model is equivalent to least-squares denoising with Laplacian regularization. We note that this convex optimization model has the analytical solution

$$\phi(x; \theta) = (M^T M + \lambda D^T D)^{-1} M^T M x.$$

We use $n = 100$, $m = 100$, and $N = 500$ training pairs. Each output $y$ is generated by sampling a different scale factor $a$

from a uniform distribution over the interval $[1, 3]$, and then evaluating the cosine function at 100 linearly spaced points in the interval $[0, 2\pi a]$. The outputs are corrupted by Gaussian noise to produce the inputs. We generate a covariance matrix $\Sigma$ according to

$$\Sigma = P^T P, \quad P \sim \mathcal{N}(0, 0.01I)$$

and then generate the components of each input $x$

$$v \sim \mathcal{N}(0, \Sigma), \quad x = y + v.$$

We generate 100 validation points in the same way. As a baseline, we use least-squares denoising with Laplacian regularization, sweeping $\lambda$ to find the value which minimizes the error on the training set. The least-squares reconstruction achieves a validation loss of 0.090; after learning, the convex optimization model achieves a validation loss of 0.014. Fig. 2 compares a prediction of the convex optimization model with least squares and the true output, for a held-out input-output pair.

*Resource Allocation:* We consider an instance of the resource allocation problem with prices, as described in Section IV, and use the separable exponential utility function. The input $x$ consists of the budget $B \in \mathbf{R}_+$ and the prices $p \in \mathbf{R}_+^m$, and the output $y \in \mathbf{R}_+^m$ is the resource allocation. Our convex optimization model has the form

$$\phi(x; \theta) = \text{argmin}_y \sum_{i=1}^m \exp(-\theta_i y_i / p_i)/\theta_i$$

$$\text{s.t.} \quad y \geq 0, \quad \mathbf{1}^T y \leq B.$$

Here the feasible parameter set is $\Theta = \mathbf{R}_+^m$. We take $m = 10$, and sample 100 training and 50 validation inputs and the true parameter according to
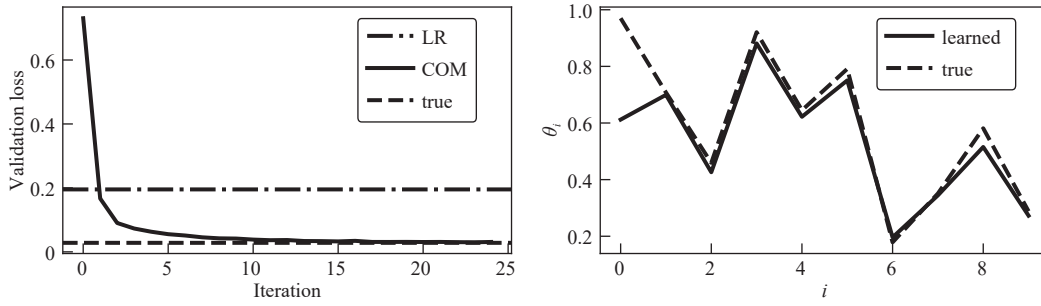
Fig. 3.   Resource allocation. Left: validation loss versus iteration for logistic regression and our convex optimization model. Right: learned and true parameters.
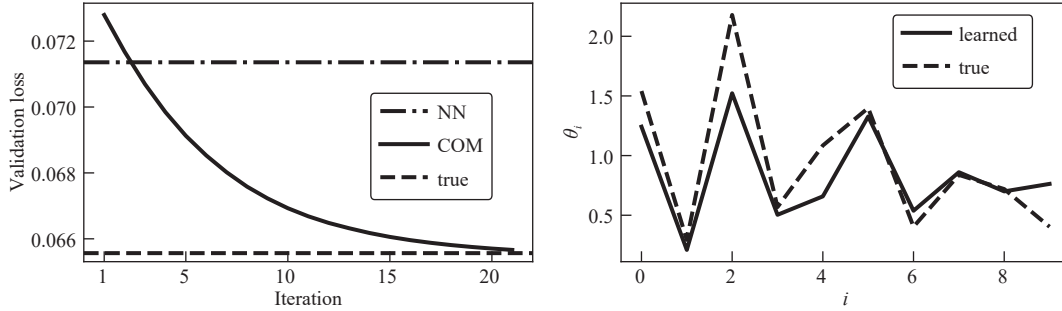


Fig. 4.   Constrained MPC. Left: validation loss for a neural network (NN), convex optimization model (COM), and true model (true). Right: learned and true parameters.

$$B \sim U[0,1], \quad p_i \sim U[0,1], \quad \theta_i^{\text{true}} \sim U[0,1]$$

where $U[a,b]$ denotes the uniform distribution over the interval $[a,b]$. The outputs were generated according to

$$y = B \frac{\phi(B,p;\theta^{\text{true}}) \circ z}{\mathbf{1}^T(\phi(B,p;\theta^{\text{true}}) \circ z)}, \quad z^i \sim U[0.5, 1.5]$$

where $\circ$ denotes elementwise multiplication. In other words, we evaluate the true convex optimization model, multiply each output by a random number between 0.5 and 1.5, and re-scale the allocation so it sums to the budget $B$. We compare the convex optimization to logistic regression using the prices as features and the (normalized) allocation as the output. In Fig. 3 we show results for these two methods. On the left, we show the validation loss versus iteration for the convex optimization model, with horizontal lines for the validation loss of the logistic regression baseline and the true model. On the right, we show the learned and true utility function parameters, and observe that the learned parameters are quite close to the true parameters.

*Constrained MPC:* We fit a convex optimization model for an instance of the MPC problem described in Section V, with $n = 10$ states, $m = 4$ controls with $\mathcal{Y} = \{y \in \mathbf{R}^m \mid \|y\|_\infty \leq 0.5\}$, and a horizon of $T = 5$. Our convex optimization model has the form

$$\phi(x;\theta) = \text{argmin} \quad \sum_{t=0}^{T-1} \theta^T x_t^2 + \|y_t\|_2^2$$
$$\text{s.t.} \quad x_{t+1} = Ax_t + By_t$$
$$\|y_t\|_\infty \leq 0.5$$
$$x_0 = x \qquad\qquad (6)$$

where the variables are the states $x_0,\ldots,x_T \in \mathbf{R}^n$ and the controls $y_0,\ldots,y_{T-1} \in \mathbf{R}^m$, the square $(x_t)^2$ is meant elementwise, and $\phi(x;\theta)$ is the optimal value of $y_0$. The parameter $\theta \in \mathbf{R}_+^n$ parametrizes the stage cost, and the dynamics matrices $A$ and $B$ are known numerical constants.

We generate a true weight $\theta^{\text{true}} \in \mathbf{R}_+^n$ with entries set to the absolute value of samples from a standard normal distribution. The dataset is generated by rolling out an MPC policy $\phi^{\text{true}}$ of the form (6), with parameter $\theta^{\text{true}}$. The policy is simulated from an initial state $x_0 \sim \mathcal{N}(0,I)$. The outputs are noise-corrupted controls, generated according to

$$u^i = \phi^{\text{true}}(x^i), \; z^i \sim \mathcal{N}(0,0.1I), \; y^i = \Pi_{\mathcal{Y}}(u^i + z^i)$$
$$v^{i-1} \sim \mathcal{N}(0,I), \; x^i = Ax^{i-1} + Bu^{i-1} + v^{i-1}$$

for $i = 1,\ldots,1000$. We generate 1000 validation points in the same way.

We use the mean-squared loss for the loss function $L$, and train for 20 iterations. As a baseline, we compare against a two-layer feedforward ReLU network with hidden layer dimension $n$, and with output clamped to have absolute value no greater than 0.5. The results are displayed in Fig. 4. The ReLU network achieves a validation loss of 0.071. The trained convex optimization model achieves a validation loss of 0.066, which is close to the validation loss of the underlying model. Additionally, the convex optimization model nearly recovers the true weights.

## REFERENCES

[1]  A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and J. Z. Kolter, "Differentiable convex optimization layers," in *Proc. 33rd Conf. Neural Information Processing Systems*, Vancouver, Canada, 2019, pp.

9558–9570.

[2] A. Agrawal, S. Barratt, S. Boyd, E. Busseti, and W. Moursi, "Differentiating through a cone program," *J. Appl. Numer. Optim.*, vol. 1, no. 2, pp. 107–115, Aug. 2019.

[3] B. Amos, " Differentiable optimization-based modeling for machine learning," Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, USA, 2019.

[4] E. Busseti, W. M. Moursi, and S. Boyd, "Solution refinement at regular points of conic problems," *Comput. Optim. Appl.*, vol. 74, no. 3, pp. 627–643, Aug. 2019.

[5] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge, USA: Cambridge University Press, 2004.

[6] G. BakIr, T. Hofmann, B. Schölkopf, A. J. Smola, B. Taskar, and S. V. N. V. Vishwanathan, *Predicting Structured Data*. Cambridge, USA: MIT Press, 2007.

[7] Y. LeCun, S. Chopra, R. Hadsell, M. A. Ranzato, and F. J. Huang, "A tutorial on energy-based learning," in *Predicting Structured Data*, G. Bakir, T. Hofman, B. Scholkopf, A. Smola, and B. Taskar, Eds. Cambridge, USA: MIT Press, 2006.

[8] B. Taskar, V. Chatalbashev, D. Koller, and C. Guestrin, "Learning structured prediction models: A large margin approach," in *Proc. 22nd Int. Conf. Machine Learning*, Bonn, Germany, 2005, pp. 896–903.

[9] B. Taskar, C. Guestrin, and D. Koller, "Max-margin Markov networks," in *Proc. 16th Int. Conf. Neural Information Processing Systems*, Vancouver and Whistler, British Columbia, Canada, 2004, pp. 25–32.

[10] I. Tsochantaridis, T. Joachims, T. Hofmann, and Y. Altun, "Large margin methods for structured and interdependent output variables," *J. Mach. Learn. Res.*, vol. 6, pp. 1453–1484, Dec. 2005.

[11] S. Chopra, R. Hadsell, and Y. LeCun, " Learning a similarity metric discriminatively, with application to face verification," in *Proc. IEEE Computer Society Conf. Computer Vision and Pattern Recognition*, San Diego, USA, 2005, pp. 539–546.

[12] D. Belanger and A. McCallum, " Structured prediction energy networks," in *Proc. 33rd Int. Conf. Machine Learning*, New York, USA, 2016, pp. 983–992.

[13] D. Belanger, B. S. Yang, and A. McCallum, "End-to-end learning for structured prediction energy networks," in *Proc. 34th Int. Conf. Machine Learning*, Sydney, Australia, 2017, pp. 429–439.

[14] B. Amos, L. Xu, and J. Z. Kolter, " Input convex neural networks," in *Proc. 34th Int. Conf. Machine Learning*, Sydney, Australia, 2017, pp. 146–155.

[15] J. Peng, L. F. Bo, and J. B. Xu, "Conditional neural fields," in *Proc. 22nd Int. Conf. Neural Information Processing Systems*, Vancouver, Canada, 2009, pp. 1419–1427.

[16] S. Zheng, S. Jayasumana, B. Romera-Paredes, V. Vineet, Z. Z. Su, D. L. Du, C. Huang, and P. H. S. Torr, " Conditional random fields as recurrent neural networks," in *Proc. IEEE Int. Conf. Computer Vision*, Santiago, Chile, 2015, pp. 1529–1537.

[17] L. C. Chen, A. G. Schwing, A. L. Yuille, and R. Urtasun, "Learning deep structured models," in *Proc. 32nd Int. Conf. Machine Learning*, Lille, France, 2015, pp. 1785–1794.

[18] Z. L. Geng, D. Johnson, and R. Fedkiw, "Coercing machine learning to output physically accurate results," *J. Comput. Phys.*, vol. 406, Article No. 109099, Apr. 2020.

[19] R. K. Ahuja and J. B. Orlin, "Inverse optimization," *Oper. Res.*, vol. 49, no. 5, pp. 771–783, Oct. 2001.

[20] C. Heuberger, " Inverse combinatorial optimization: A survey on problems, methods, and results," *J. Comb. Optim.*, vol. 8, no. 3, pp. 329–361, Sept. 2004.

[21] V. Chatalbashev, "Inverse convex optimization," M.S. thesis, Stanford Univ., Stanford, USA, 2005.

[22] A. Keshavarz, Y. Wang, and S. Boyd, "Imputing a convex objective function," in *Proc. IEEE Int. Symp. Intelligent Control*, Denver, USA, 2011, pp. 613–619.

[23] B. Amos and J. Z. Kolter, "OptNet: Differentiable optimization as a layer in neural networks," in *Proc. 34th Int. Conf. Machine Learning*, Sydney, Australia, 2017, pp. 136–145.

[24] A. V. Fiacco and G. P. McCormick, *Nonlinear Programming—Sequential Unconstrained Minimization Techniques*. New York, USA: John Wiley & Sons Ltd, 1990.

[25] A. V. Fiacco, " Sensitivity analysis for nonlinear programming using penalty methods," *Mathem. Program.*, vol. 10, no. 1, pp. 287–311, Dec. 1976.

[26] B. Amos, I. D. Jimenez, J. Sacks, B. Boots, and J. Z. Kolter, "Differentiable MPC for end-to-end planning and control," in *Proc. 32nd Int. Conf. Neural Information Processing Systems*, Montreal, Canada, 2018, pp. 8299–8310.

[27] F. de A. Belbute-Peres, K. Smith, K. R. Allen, J. B. Tenenbaum, and J. Z. Kolter, "End-to-end differentiable physics for learning and control," in *Proc. 32nd Conf. Neural Information Processing Systems*, Montreal, Canada, 2018, pp. 7178–7189.

[28] S. T. Barratt and S. P. Boyd, "Fitting a Kalman smoother to data," in *Proc. American Control Conf.*, Denver, USA, 2020.

[29] A. Agrawal, S. Barratt, S. Boyd, and B. Stellato, " Learning convex optimization control policies," in *Proc. 2nd Annu. Conf. Learning for Dynamics and Control*, 2020.

[30] C. K. Ling, F. Fang, and J. Z. Kolter, "What game are we playing? end-to-end learning in normal and extensive form games," in *Proc. 27th Int. Joint Conf. Artificial Intelligence*, Stockholm, Sweden, 2018.

[31] C. K. Ling, F. Fang, and J. Z. Kolter, "Large scale learning of agent rationality in two-player zero-sum games," in *Proc. AAAI Conf. Artificial Intelligence*, Palo Alto, USA, 2019, pp. 6104–6111.

[32] Q. Berthet, M. Blondel, O. Teboul, M. Cuturi, J. P. Vert, and F. Bach, "Learning with differentiable perturbed optimizers," arXiv preprint arXiv: 2002.08676, Jun. 2020.

[33] S. Barratt, G. Angeris, and S. Boyd, " Automatic repair of convex optimization problems," *Optim. Eng.*, vol. 22, pp. 247–259, 2021.

[34] B. Amos and D. Yarats, " The differentiable cross-entropy method," arXiv preprint arXiv: 1909.12830, Aug. 2020.

[35] S. Barratt and S. Boyd, " Least squares auto-tuning," *Eng. Optim.*, vol. 53, no. 5, pp. 789–810, May 2021.

[36] S. Barratt and R. Sharma, "Optimizing for generalization in machine learning with cross-validation gradients," arXiv preprint arXiv: 1805.07072, May 2018.

[37] S. Diamond, V. Sitzmann, F. Heide, and G. Wetzstein, "Unrolled optimization with deep priors," arXiv preprint arXiv: 1705.08041, Dec. 2018.

[38] J. Domke, " Generic methods for optimization-based modeling," in *Proc. 15th Int. Conf. Artificial Intelligence and Statistics*, La Palma, Canary Islands, 2012, pp. 318–326.

[39] C. Finn, "Learning to learn with gradients," Univ. California, Berkeley, USA, Tech. Rep. No. UCB/EECS-2018–105, 2018.

[40] D. Maclaurin, D. Duvenaud, and R. P. Adams, "Gradient-based hyperparameter optimization through reversible learning," in *Proc. 32nd Int. Conf. Machine Learning*, Lille, France, 2015, pp. 2113–2122.

[41] A. Agrawal and S. Boyd, " Differentiating through log-log convex programs," arXiv preprint arXiv: 2004.12553, May 2020.

[42] J. Lorraine, P. Vicol, and D. Duvenaud, " Optimizing millions of hyperparameters by implicit differentiation," arXiv preprint arXiv: 1911.02590, Nov. 2019.

[43] N. Hansen and A. Ostermeier, " Completely derandomized self-adaptation in evolution strategies," *Evol. Comput.* , vol. 9, no. 2, pp. 159–195, Jun. 2001.

[44] J. Močkus, "On Bayesian methods for seeking the extremum," in *Proc. IFIP Technical Conf. Optimization Techniques*, Novosibirsk, Russia, 1974, pp. 400–404.

[45] F. J. Solis and R. J. B. Wets, " Minimization by random search techniques," *Math. Oper. Res.*, vol. 6, no. 1, pp. 19–30, Feb. 1981.

[46] A. Beck and M. Teboulle, "Gradient-based algorithms with applications to signal-recovery," in *Convex Optimization in Signal Processing and Communications*, D. P. Palomar and Y. C. Eldar, Eds. Cambridge, USA: Cambridge University Press, 2009, pp. 42–88.

[47] L. Bottou, " Large-scale machine learning with stochastic gradient descent," in *Proc. COMPSTAT*, Y. Lechevallier and G. Saporta, Eds. Heidelberg: Physica-Verlag, 2010, pp. 177–186.

[48] J. Nocedal and S. J. Wright, *Numerical Optimization*. New York, USA: Springer, 2006.

[49] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, USA: Springer, 2006.

[50] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, USA: MIT Press, 2016.

[51] R. E. Barlow and H. D. Brunk, "The isotonic regression problem and its dual," *J. Am. Stat. Assoc.*, vol. 41, no. 5, pp. 140–147, Mar. 1972.

[52] M. J. Best and N. Chakravarti, " Active set algorithms for isotonic regression; A unifying framework," *Math. Program.* , vol. 47, no. 1–3, pp. 425–439, May 1990.

[53] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. 2nd ed. Tokyo, Japan: Springer Science & Business Media, 2009.

[54] S. J. Kim, K. Koh, S. Boyd, and D. Gorinevsky, "$\ell_1$ trend filtering" *SIAM Rev.*, vol. 51, no. 2, pp. 339–360, May 2009.

[55] D. Bertsekas, *Dynamic Programming and Optimal Control, Vol. I.* 4th ed. Belmont, USA: Athena Scientific, 2017.

**Akshay Agrawal** is a Ph.D. candidate at Stanford University, USA, advised by Professor Stephen Boyd. His research interests are in applications of optimization and machine learning. He received the B.S. and M.S. in computer science from Stanford University in 2016 and 2017, respectivelly. Akshay is the developer of PyMDE, a library for computing embeddings of large datasets, and a core developer of CVXPY, a domain-specific language for convex optimization used by many universities and companies.



**Shane Barratt** is a Ph.D. candidate in the Department of Electrical Engineering at Stanford University. His research focuses on convex optimization, and in particular its applications to machine learning and control. He received the M.S. degree in electrical engineering from Stanford University in 2019, and the B.S. degree in electrical engineering and computer science from the University of California, USA, in 2017.



**Stephen Boyd** (F' 99) is the Samsung Professor of engineering, and Professor of electrical engineering in the Information Systems Laboratory at Stanford University, with courtesy appointments in computer science and management science and engineering. He received the A.B. degree in mathematics from Harvard University, USA, in 1980, and the Ph.D. in electrical engineering and computer science from the University of California, USA, in 1985, and then joined the faculty at Stanford. His current research interests include convex optimization applications in control, signal processing, machine learning, finance, and circuit design.