

Deep Declarative Networks: A New Hope

Stephen Gould, Richard Hartley, Dylan Campbell

Abstract—We explore a new class of end-to-end learnable models wherein data processing nodes (or network layers) are defined in terms of desired behavior rather than an explicit forward function. Specifically, the forward function is implicitly defined as the solution to a mathematical optimization problem. Consistent with nomenclature in the programming languages community, we name these models *deep declarative networks*. Importantly, we show that the class of deep declarative networks subsumes current deep learning models. Moreover, invoking the implicit function theorem, we show how gradients can be back-propagated through many declaratively defined data processing nodes thereby enabling end-to-end learning. We show how these declarative processing nodes can be implemented in the popular PyTorch deep learning software library allowing declarative and imperative nodes to co-exist within the same network. We also provide numerous insights and illustrative examples of declarative nodes and demonstrate their application for image and point cloud classification tasks.

Index Terms—Deep Learning, Implicit Differentiation, Declarative Networks

1 INTRODUCTION

MODERN deep learning models are composed of parametrized processing nodes (or layers) organized in a directed graph. There is an entire zoo of different model architectures categorized primarily by graph structure and mechanisms for parameter sharing [33]. In all cases the function for transforming data from the input to the output of a processing node is explicitly defined. End-to-end learning is then achieved by back-propagating an error signal along edges in the graph and adjusting parameters so as to minimize the error. Almost universally, the error signal is encoded as the gradient of some global objective (or regularized loss) function and stochastic gradient descent based methods are used to iteratively update parameters. Automatic differentiation (or hand-derived gradients) is used to compute the derivative of the output of a processing node with respect to its input, which is then combined with the chain rule of differentiation proceeding backwards through the graph. The error gradient can thus be calculated efficiently for all parameters of the model as the signal is passed backwards through each processing node.

In this paper we advocate for a new class of end-to-end learnable models, which we call *deep declarative networks* (DDNs), and which collects several recent works on differentiable optimization [1, 2, 3, 4, 28] and related ideas [11, 45, 47] into a single framework. Instead of explicitly defining the forward processing function, **nodes in a DDN are defined implicitly by specifying behavior**. That is, the input-output relationship for a node is defined in terms of an objective and constraints in a mathematical optimization problem, the output being the solution of the problem conditioned on the input (and parameters). Importantly, as we will show, we can still perform back-propagation through a DDN by making use of implicit differentiation. Moreover, the gradient calculation does not require knowl-

edge of the method used to solve the optimization problem, only the form of the objective and constraints, thereby allowing any state-of-the-art solver to be used during the forward pass.

DDNs subsume conventional deep learning models in that any explicitly defined forward processing function can also be defined as a node in a DDN. Furthermore, declaratively defined nodes and explicitly defined nodes can coexist within the same end-to-end learnable model. To make the distinction clear, when both types of nodes appear in the same model we refer to the former as *declarative nodes* and the latter as *imperative nodes*. To this end, we have developed a reference DDN implementation within PyTorch [39], a popular software library for deep learning, supporting both declarative and imperative nodes.

We present some theoretical results that show the conditions under which exact gradients can be computed and the form of such gradients. We also discuss scenarios in which the exact gradient cannot be computed (such as non-smooth objectives) but a descent direction can still be found, allowing stochastic optimization of model parameters to proceed. These ideas are explored through a series of illustrative examples and tested experimentally on the problems of image and **point cloud classification** using modified ResNet [30] and PointNet [40] architectures, respectively.

The decisive advantage of the declarative view of deep neural networks is that **it enables the use of classic constrained and unconstrained optimization algorithms as a modular component within a larger, end-to-end learnable network**. This extends the concept of a neural network layer to include, for example, geometric model fitting, such as relative or absolute pose solvers or bundle adjustment, model-predictive control algorithms, expectation-maximization, matching, optimal transport, and structured prediction solvers to name a few. Moreover, the change in perspective can help us envisage variations of standard neural network operations with more desirable properties, such as robust feature pooling instead of standard average pooling. There is also the potential to reduce opacity and redundancy in networks by incorporating local model fit-

- S. Gould and D. Campbell are with the Australian Centre for Robotic Vision and the Research School of Computer Science at the Australian National University. R. Hartley is with the the Australian Centre for Robotic Vision and the Research School of Engineering at the Australian National University. E-mail: stephen.gould@anu.edu.au

ting as a component within larger models. For example, we can directly use the (often nonlinear) underlying physical and mathematical models rather than having to re-learn these within the network. Importantly, this allows us to provide guarantees and enforce hard constraints on representations within a model (for example, that a rotation within a geometric model is valid or that a permutation matrix is normalized correctly). Furthermore, the approach is still applicable when no closed form solution exists, allowing sophisticated approaches with non-differentiable steps (such as RANSAC [26]) to be used internally. Global end-to-end learning of network parameters is still possible even in this case.

Since this is a new approach, there are still several challenges to be addressed. We discuss some of these in this paper but leave many for future work by us and the community. Some preliminary ideas relating to declarative networks appear in previous works (discussed below) but, to the best of our knowledge, this paper is the first to present a general coherent framework for describing these models.

2 BACKGROUND AND RELATED WORK

The ability to differentiate through declarative nodes relies on the implicit function theorem, which has a long history whose roots can be traced back to works by Descartes, Leibniz, Bernoulli and Euler [44]. It was Cauchy who first placed the theorem on rigorous mathematical grounds and Dini who first presented the theorem in its modern multivariate form [21, 32]. Roughly speaking, the theorem states conditions under which the derivative of a variable y with respect to another variable x exists for implicitly defined functions, $f(x, y) = 0$, and provides a means for computing the derivative of y with respect to x when it does exist.

In the context of deep learning, it is the derivative of the output of a (declarative) node with respect to its input that facilitates end-to-end parameter learning by back-propagation [33, 41]. In this sense the learning problem is formulated as an optimization problem on a given error metric or regularized loss function. When declarative nodes appear in the network, computing the network output and hence the loss function itself requires solving an inner optimization problem. Formally, we can think of the learning problem as an upper optimization problem and the network output as being obtained from a lower optimization problem within a bi-level optimization framework [7, 46].

Bi-level optimization (and the need for implicit differentiation) has appeared in various settings in the machine learning literature, most notably for the problem of meta-learning. For example, Do et al. [19] consider the problem of determining regularization parameters for log-linear models formulated as a bi-level optimization problem. Domke [20] addresses the problem of learning parameters of continuous energy-based models wherein inference (equivalent to the forward pass in a neural network) requires finding the minimizer of a so-called energy function. The resulting learning problem is then necessarily bi-level. Last, Klatzer and Pock [31] propose a bi-level optimization framework for choosing hyper-parameter settings for support vector machines that avoids the need for cross-validation.

In computer vision and image processing, bi-level optimization has been used to formulate solutions to pixel-labeling problems. Samuel and Tappen [42] propose learning parameters of a continuous Markov random field with bi-level optimization and apply their technique to image denoising and in-painting. The approach is a special case of energy-based model learning [20]. Ochs et al. [37] extend bi-level optimization to handle non-smooth lower-level problems and apply their method to image segmentation tasks.

Recent works have started to consider the question of placing specific optimization problems within deep learning models [4, 11, 28, 45, 47]. These approaches can be thought of as establishing the groundwork for DDNs by developing specific declarative components. Gould et al. [28] summarize some general results for differentiating through unconstrained, linearly constrained and inequality constrained optimization problems. In the case of unconstrained and equality constrained problems the results are exact whereas for inequality constrained problems they make use of an interior-point approximation. We extend these results to the case of exact differentiation for problems with nonlinear equality and inequality constraints.

Amos and Kolter [4] also show how to differentiate through an optimization problem, for the specific case of quadratic programs (QPs). A full account of the work, including discussion of more general cone programs, appears in Amos [3]. Along the same lines Agrawal et al. [2] report results for efficiently differentiating through cone programs with millions of parameters. In both cases (quadratic programs and cone programs) the problems are convex and efficient algorithms exist for finding the minimum. We make no restriction on convexity for declarative nodes but still assume that an algorithm exists for evaluating them in the forward pass. Moreover, our paper establishes a unified framework for viewing these works in an end-to-end learnable model—the deep declarative network (DDN).

Other works consider the problem of differentiating through discrete submodular problems [18, 45]. These problems have the nice property that minimizing a convex relaxation still results in the optimal solution to the submodular minimization problem, which allows the derivative to be computed [18]. For submodular maximization there exist polynomial time algorithms to find approximate solutions. Smoothing of these solutions results in a differentiable model as demonstrated in Tschitschek et al. [45].

Close in spirit to the idea of a deep declarative network is the recently proposed SATNet [47]. Here MAXSAT problems are approximated by solving a semi-definite program (SDP), which is differentiable. A fast solver allows the problem to be evaluated efficiently in the forward pass and thanks to implicit differentiation there is no need to explicitly unroll the optimization procedure, and hence store Jacobians, making it memory and time efficient in the backward pass. The method is applied to solving Sudoku problems presented as images, which requires that the learned network encode logical constraints.

Another interesting class of models that fit into the deep declarative networks framework are deep equilibrium models recently proposed by Bai et al. [6]. Here the model executes a sequence of fixed-point iterations $y^{(t)} = f(x, y^{(t-1)})$ until convergence in the forward pass. Bai et al. [6] show

that rather than back-propagating through the unbounded sequence of fixed-point iterations, the derivative of the solution y with respect to input x can be computed directly via the implicit function theorem by observing that y satisfies implicit function $f(x, y) - y = 0$.

Chen et al. [11] show how to differentiate through ordinary differential equation (ODE) initial value problems at some time T using the adjoint sensitivity method, and view residual networks [30] as a discretization of such problems. This approach can be interpreted as solving a feasibility problem with an integral constraint function, and hence an exotic type of declarative node. The requisite gradients in the backward pass are computed elegantly by solving a second augmented ODE. While such problems can be included within our declarative framework, in this paper we focus on results for twice-differentiable objective and constraints functions that can be expressed in closed form.

There have also been several works that have proposed differentiable components based on optimization problems for addressing specific tasks. In the context of video classification, Fernando and Gould [23, 24] show how to differentiate through a rank-pooling operator [25] within a deep learning model, which involves solving a support vector regression problem. The approach was subsequently generalized to allow for a subspace representation of the video and learning on a manifold [12].

Santa Cruz et al. [43] propose a deep learning model for learning permutation matrices for visual attribute ranking and comparison. Two variants are proposed both relaxing the permutation matrix representation to a doubly-stochastic matrix representation. The first variant involves iteratively normalizing rows and columns to approximately project a positive matrix onto the set of doubly-stochastic matrices. The second variant involves formulating the projection as a quadratic problem and solving exactly.

Lee et al. [34] consider the problem of few-shot learning for visual recognition. They embed a differentiable QP [4] into a deep learning model that allows linear classifiers to be trained as a basis for generalizing to new visual categories. Promising results are achieved on standard benchmarks and they report low training overhead in that solving the QP takes about the same time as image feature extraction.

In the context of planning and control, Amos et al. [5] propose a differentiable model predictive controller in a reinforcement learning setting. They are able to demonstrate superior performance on classic pendulum and cartpole problems. De Avila Belbute-Peres et al. [14] show how to differentiate through physical models, specifically, the optimal solution of a linear complementarity problem, which allows a simulated physics environment to be placed within an end-to-end learnable system.

Last, imposing hard constraints on the output of a deep neural network using a Krylov subspace method was investigated in Márquez-Neila et al. [35]. The approach is applied to the task of human pose estimation and demonstrates the feasibility of training a very high-dimensional model that enforces hard constraints, albeit without improving over a softly constrained baseline.

3 NOTATION

Our results require differentiating vector-valued functions with respect to vector arguments. To assist presentation we clarify our notation here. Consider the function $f(x, y, z)$ where both y and z are themselves functions of x . We have

$$\frac{d}{dx}f = \frac{\partial f}{\partial x} \frac{dx}{dx} + \frac{\partial f}{\partial y} \frac{dy}{dx} + \frac{\partial f}{\partial z} \frac{dz}{dx} \quad (1)$$

by the chain rule of differentiation. For functions taking vector arguments, $f : \mathbb{R}^n \rightarrow \mathbb{R}$, we write the derivative vector as

$$Df = \left[\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \dots, \frac{\partial f}{\partial x_n} \right] \in \mathbb{R}^{1 \times n}. \quad (2)$$

For vector-valued functions $f : \mathbb{R} \rightarrow \mathbb{R}^m$ we write

$$Df = \left[\frac{df_1}{dx}, \dots, \frac{df_m}{dx} \right]^T \in \mathbb{R}^{m \times 1}. \quad (3)$$

More generally, we define the derivative Df of $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ as an $m \times n$ matrix with entries

$$(Df(x))_{ij} = \frac{\partial f_i}{\partial x_j}(x). \quad (4)$$

Then the chain rule for $h(x) = g(f(x))$ is simply

$$Dh(x) = Dg(f(x))Df(x) \quad (5)$$

where the matrices automatically have the right dimensions and standard matrix-vector multiplication applies. When taking partial derivatives we use a subscript to denote the formal variable over which the derivative is computed (the remaining variables fixed), for example $D_X f(x, y)$. For brevity we use the shorthand $D_{XY}^2 f$ to mean $D_X(D_Y f)^T$.

When no subscript is given for multi-variate functions we take D to mean the total derivative with respect to the independent variables. So, with x as the independent variable, the vector version of Equation 1 becomes

$$Df = D_X f + D_Y f D_Y + D_Z f D_Z. \quad (6)$$

4 DEEP DECLARATIVE NETWORKS

We are concerned with data processing nodes in deep neural networks whose output $y \in \mathbb{R}^m$ is defined as the solution to a constrained optimization problem parametrized by the input $x \in \mathbb{R}^n$. In the most general setting we have

$$y \in \arg \min_{u \in C} f(x, u) \quad (7)$$

where $f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$ is an arbitrary parametrized objective function and $C \subseteq \mathbb{R}^m$ is an arbitrary constraint set (that may also be parametrized by x).¹

To distinguish such processing nodes from processing nodes (or layers) found in conventional deep learning models, which specify an explicit mapping from input to output, we refer to the former as declarative nodes and the latter as imperative nodes, which we formally define as follows.

Definition 4.1: (Imperative Node). An *imperative* node is one in which the implementation of the forward processing

1. In practical deep learning systems inputs and outputs are generally one or more multi-dimensional tensors. For clarity of exposition we consider inputs and outputs as single vectors without loss of generality.

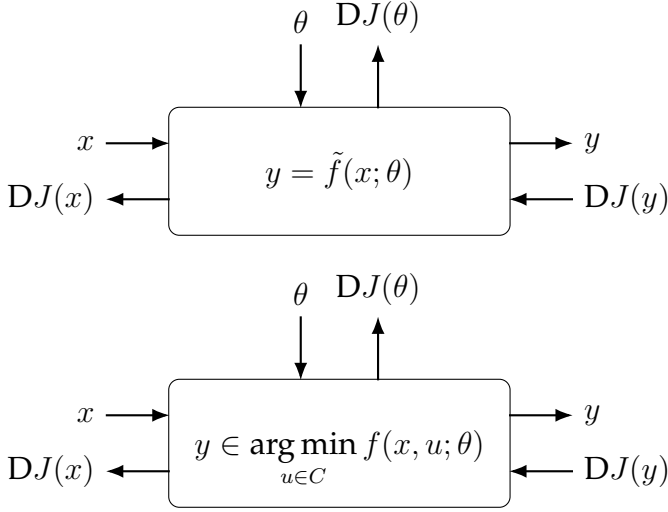


Fig. 1: Parametrized data processing nodes in an end-to-end learnable model with global objective function J . During the forward evaluation pass of an imperative node (top) the input x is transformed into output y based on some explicit parametrized function $\tilde{f}(\cdot; \theta)$. During the forward evaluation pass of a declarative node (bottom) the output y is computed as the minimizer of some parametrized objective function $f(x, \cdot; \theta)$. During the backward parameter update pass for either node type, the gradient of the global objective function with respect to the output $DJ(y)$ is propagated backwards via the chain rule to produce gradients with respect to the input $DJ(x)$ and parameters $DJ(\theta)$.

function \tilde{f} is explicitly defined. The output is then defined mathematically as

$$y = \tilde{f}(x; \theta)$$

where θ are the parameters of the node (possibly empty).

Definition 4.2: (Declarative Node). A *declarative* node is one in which the exact implementation of the forward processing function is not defined; rather the input–output relationship ($x \mapsto y$) is defined in terms of behavior specified as the solution to an optimization problem

$$y \in \arg \min_{u \in C} f(x, u; \theta)$$

where f is the (parametrized) objective function, θ are the node parameters (possibly empty), and C is the set of feasible solutions, that is, constraints.

Figure 1 is a schematic illustration of the different types of nodes. Since nodes form part of an end-to-end learnable model we are interested in propagating gradients backwards through them. In the sequel we make no distinction between the inputs and the parameters of a node as these are treated the same for the purpose of computing gradients.

The definition of declarative nodes (Definition 4.2) is very general and encompasses many subproblems that we would like embedded within a network—robust fitting, projection onto a constraint set, matching, optimal transport, physical simulations, etc. Some declarative nodes may not be efficient to evaluate (that is, solve the optimization problem specifying its behavior), and in some cases the gradient may not exist (e.g., when the feasible set or output space

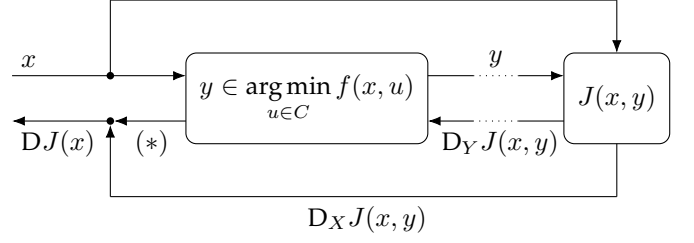


Fig. 2: Bi-level optimization problem showing back-propagation of gradients through a deep declarative node. The quantity $(*)$ is $D_Y J(x, y) D y(x)$ which when added to $D_X J(x, y)$ gives $DJ(x, y(x))$. The bypass connections (topmost and bottommost paths) do not exist when the upper-level objective J only depends on x through y . Moreover, if f appears in J as the only term involving y then $D_Y J(x, y)$ is zero and the backward edge $(*)$ is not required. That is, $DJ(x) = D_X J(x, y)$.

is discrete). Nevertheless, under certain conditions on the objective and constraints, covering a wide range of problems or problem approximations (such as shown by Wang et al. [47] for MAXSAT), the declarative node is differentiable and can be placed within an end-to-end learnable model as we discuss below.

4.1 Learning

We make no assumption about *how* the optimal solution y is obtained in a declarative node, only that there exists some algorithm for computing it. The consequence of this assumption is that, when performing back-propagation, we do not need to propagate gradients through a complicated algorithmic procedure from output to input. Rather we rely on implicit differentiation to directly compute the gradient of the node’s output with respect to its input (or parameters). Within the context of end-to-end learning with respect to some global objective (i.e., loss function) this becomes a bi-level (multi-level) optimization problem [7, 15], which was first studied in the context of two-player leader–follower games by Stackelberg in the 1930s [46]. Formally, we can write

$$\begin{aligned} & \text{minimize} && J(x, y) \\ & \text{subject to} && y \in \arg \min_{u \in C} f(x, u) \end{aligned} \quad (8)$$

where the minimization is over all tunable parameters in the network. Here $J(x, y)$ may depend on y through additional layers of processing. Note also that y is itself a function of x . As such minimizing the objective via gradient descent requires the following computation

$$DJ(x, y) = D_X J(x, y) + D_Y J(x, y) D y(x), \quad (9)$$

The key challenge for declarative nodes then is the calculation of $D y(x)$, which we will discuss in Section 4.3. A schematic illustration of a bi-level optimization problem and associated gradient calculations is shown in Figure 2.

The higher-level objective J is often defined as the sum of various terms (including loss terms and regularization terms) and decomposes over individual examples from a training dataset. In such cases the gradient of J with respect to model parameters also decomposes as the sum of gradients for losses on each training example (and regularizers).

A simpler form of the gradient arises when the lower-level objective function f appears in the upper-level objective J as the only term involving y . Formally, if $J(x, y) = g(x, f(x, y))$ and the lower-level problem is unconstrained (that is, $C = \mathbb{R}^m$), then

$$\begin{aligned} DJ(x, y) &= D_X g(x, f) + D_F g(x, f) (Df + D_Y f Dy) \\ &= D_X g(x, f) + D_F g(x, f) Df \end{aligned} \quad (10)$$

since $D_Y f(x, y) = 0$ by virtue of y being an unconstrained minimum of $f(x, \cdot)$. For example, if $J(x, y) = f(x, y)$, then $DJ(x, y) = D_X f(x, y)$. This amounts to a variant of block coordinate descent on x and y , where y is optimized fully during each step, and in such cases we do not need to propagate gradients backwards through the declarative node. The remainder of this paper is focused on the more general case where y appears in other terms in the loss function (possibly through composition with other nodes) and hence calculation of Dy is required.

4.2 Common sub-classes of declarative nodes

It will be useful to consider three very common cases of the general setting presented in Equation 7, in increasing levels of sophistication. The first is the unconstrained case,

$$y \in \arg \min_{u \in \mathbb{R}^m} f(x, u) \quad (11)$$

where the objective function f is minimized over the entire m -dimensional output space. The second is when the feasible set is defined by p possibly nonlinear equality constraints,

$$\begin{aligned} y \in \arg \min_{u \in \mathbb{R}^m} & f(x, u) \\ \text{subject to} & h_i(x, u) = 0, \quad i = 1, \dots, p. \end{aligned} \quad (12)$$

The third is when the feasible set is defined by p equality and q inequality constraints,

$$\begin{aligned} y \in \arg \min_{u \in \mathbb{R}^m} & f(x, u) \\ \text{subject to} & h_i(x, u) = 0, \quad i = 1, \dots, p \\ & g_i(x, u) \leq 0, \quad i = 1, \dots, q. \end{aligned} \quad (13)$$

Note that in all cases we can think of y as an implicit function of x , and will often write $y(x)$ to make the input-output relationship clear. We now show that, under mild conditions, the gradient of y with respect to x can be computed for these and other special cases without having to know anything about the algorithmic procedure used to solve for y in the first place.

In the sequel it will be useful to characterize solutions y using the concept of regularity due to Bertsekas [9, §3.3.1].

Definition 4.3: (Regular Point). A feasible point u is said to be *regular* if the equality constraint gradients $D_U h_i$ and the active inequality constraint gradients $D_U g_i$ are linearly independent, or there are no equality constraints and the inequality constraints are all inactive at u .²

2. An inequality constraint g_i is active for a feasible point u if $g_i(x, u) = 0$ and inactive if $g_i(x, u) < 0$.

4.3 Back-propagation through declarative nodes

Our results for computing gradients are based on implicit differentiation. We begin with the unconstrained case, which has appeared in different guises in several previous works [28]. The result is a straightforward application of Dini's implicit function theorem [21, p19] applied to the first-order optimality condition $D_Y f(x, y) = 0$.

Proposition 4.4 (Unconstrained): Consider a function $f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$. Let

$$y(x) \in \arg \min_{u \in \mathbb{R}^m} f(x, u).$$

Assume $y(x)$ exists and that f is second-order differentiable in the neighborhood of the point $(x, y(x))$. Set $H = D_{YY}^2 f(x, y(x)) \in \mathbb{R}^{m \times m}$ and $B = D_{XY}^2 f(x, y(x)) \in \mathbb{R}^{m \times n}$. Then for H non-singular the derivative of y with respect to x is

$$Dy(x) = -H^{-1}B.$$

Proof. For any optimal y , the first-order optimality condition requires $D_Y f(x, y) = 0_{1 \times m}$. The result then follows from the implicit function theorem: transposing and differentiating both sides with respect to x we have

$$\begin{aligned} 0_{m \times n} &= D(D_Y f(x, y))^T \\ &= D_{XY}^2 f(x, y) + D_{YY}^2 f(x, y) Dy(x) \end{aligned} \quad (14)$$

which can be rearranged to give

$$Dy(x) = -(D_{YY}^2 f(x, y))^{-1} D_{XY}^2 f(x, y) \quad (15)$$

when $D_{YY}^2 f(x, y)$ is non-singular. \square

In fact, the result above holds for any stationary point of $f(x, \cdot)$ not just minima. However, for declarative nodes it is the minima that are of interest. Our next result gives the gradient for equality constrained declarative nodes, making use of the fact that the optimal solution of a constrained optimization problem is a stationary point of the Lagrangian associated with the problem.

Proposition 4.5 (Equality Constrained): Consider functions $f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$ and $h : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^p$. Let

$$\begin{aligned} y(x) \in \arg \min_{u \in \mathbb{R}^m} & f(x, u) \\ \text{subject to} & h_i(x, u) = 0, \quad i = 1, \dots, p. \end{aligned}$$

Assume that $y(x)$ exists, that f and $h = [h_1, \dots, h_p]^T$ are second-order differentiable in the neighborhood of $(x, y(x))$, and that $\text{rank}(D_Y h(x, y)) = p$. Then for H non-singular

$$Dy(x) = H^{-1}A^T(AH^{-1}A^T)^{-1}(AH^{-1}B - C) - H^{-1}B$$

where

$$A = D_Y h(x, y) \in \mathbb{R}^{p \times m}$$

$$B = D_{XY}^2 f(x, y) - \sum_{i=1}^p \lambda_i D_{XY}^2 h_i(x, y) \in \mathbb{R}^{m \times n}$$

$$C = D_X h(x, y) \in \mathbb{R}^{p \times n}$$

$$H = D_{YY}^2 f(x, y) - \sum_{i=1}^p \lambda_i D_{YY}^2 h_i(x, y) \in \mathbb{R}^{m \times m}$$

and $\lambda \in \mathbb{R}^p$ satisfies $\lambda^T A = D_Y f(x, y)$.

Proof. By the method of Lagrange multipliers [8] we can form the Lagrangian

$$\mathcal{L}(x, y, \lambda) = f(x, y) - \sum_{i=1}^p \lambda_i h_i(x, y). \quad (16)$$

Assume y is optimal for a fixed input x . Since $D_Y h(x, y)$ is full rank we have that y is a regular point. Then there exists a λ such that the Lagrangian is stationary at the point (y, λ) . Here both y and λ are understood to be functions of x . Thus

$$\left[(D_Y f(x, y) - \sum_{i=1}^p \lambda_i D_Y h_i(x, y))^T \right]_{h(x, y)} = 0_{m+p} \quad (17)$$

where the first m rows are from differentiating \mathcal{L} with respect to y (that is, $D_Y \mathcal{L}^T$) and the last p rows are from differentiating \mathcal{L} with respect to λ (that is, $D_\lambda \mathcal{L}^T$).

Now observe that at the optimal point y we have that either $D_Y f(x, y) = 0_{1 \times m}$, that is, the optimal point of the unconstrained problem automatically satisfies the constraints, or $D_Y f(x, y)$ is non-zero and orthogonal to the constraint surface defined by $h(x, y) = 0$. In the first case we can simply set $\lambda = 0_p$. In the second case we have (from the first row in Equation 17)

$$D_Y f(x, y) = \sum_{i=1}^p \lambda_i D_Y h_i(x, y) = \lambda^T A \quad (18)$$

for A defined above.

Now, differentiating the gradient of the Lagrangian with respect to x we have

$$D \left[(D_Y f(x, y))^T - \sum_{i=1}^p \lambda_i (D_Y h_i(x, y))^T \right]_{h(x, y)} = 0 \quad (19)$$

For the first row this gives

$$D_{XY}^2 f + D_{YY}^2 f D_Y y - D_Y h^T D \lambda - \sum_{i=1}^p \lambda_i (D_{XY}^2 h_i + D_{YY}^2 h_i D_Y y) = 0_{m \times n} \quad (20)$$

and for the second row we have

$$D_X h + D_Y h D_Y y = 0_{p \times n}. \quad (21)$$

Therefore

$$\begin{bmatrix} D_{XY}^2 f - \sum_{i=1}^p \lambda_i D_{YY}^2 h_i & -D_Y h^T \\ D_Y h & 0_{p \times p} \end{bmatrix} \begin{bmatrix} D_Y y \\ D \lambda \end{bmatrix} + \begin{bmatrix} D_{XY}^2 f - \sum_{i=1}^p \lambda_i D_{XY}^2 h_i \\ D_X h \end{bmatrix} = 0_{(m+p) \times n} \quad (22)$$

where all functions are evaluated at (x, y) . We can now solve by variable elimination [10] to get

$$D \lambda(x) = (A H^{-1} A^T)^{-1} (A H^{-1} B - C) \quad (23)$$

$$D y(x) = H^{-1} A^T (A H^{-1} A^T)^{-1} (A H^{-1} B - C) - H^{-1} B \quad (24)$$

with A, B, C , and H as defined above. \square

The Lagrange multipliers λ in Proposition 4.5 are often made available by the method used to solve for $y(x)$, e.g., in the case of primal-dual methods for convex optimization problems. Where it is not provided by the solver it can be

computed explicitly solving $\lambda^T A = D_Y f(x, y)$, which has unique analytic solution

$$\lambda = (A A^T)^{-1} A (D_Y f)^T. \quad (25)$$

Thus given optimal y we can find λ .

Although the result looks expensive to compute, much of the computation is shared since $H^{-1} A^T$ is independent of the coordinate of x for which the gradient is being computed. Moreover, H need only be factored once and then reused in computing $H^{-1} B$ for each input dimension, that is, column of B . And for many problems H has structure that can be further exploited to speed calculation of $D y$.

Inequality constrained problems encompass a richer class of deep declarative nodes. Gould et al. [28] addressed the question of computing gradients for these problems by approximating the inequality constraints with a logarithmic barrier [10] and leveraging the result for unconstrained problems. However, it is also possible to calculate the gradient without resorting to approximation by leveraging the KKT optimality conditions [9] as has been shown in previous work for the special case of convex optimization problems [2, 4]. Here we present a result for the more general non-convex case.

We first observe that for a smooth objective function f the optimal solution $y(x)$ to the inequality constrained problem in Equation 13 is also a local (possibly global) minimum of the problem with any or all inactive inequality constraints removed. Moreover, the derivative $D y(x)$ is the same for both problems. As such, without loss of generality, we can assume that all inequality constraints are active.

Proposition 4.6 (Inequality Constrained): Consider functions $f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$, $h : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^p$ and $g : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^q$. Let

$$\begin{aligned} y(x) \in \arg \min_{u \in \mathbb{R}^m} & f(x, u) \\ \text{subject to} & h_i(x, u) = 0, i = 1, \dots, p \\ & g_i(x, u) \leq 0, i = 1, \dots, q. \end{aligned}$$

Assume that $y(x)$ exists, that f, h and g are second-order differentiable in the neighborhood of $(x, y(x))$, and that all inequality constraints are active at $y(x)$. Let $\tilde{h} = [h_1, \dots, h_p, g_1, \dots, g_q]$ and assume $\text{rank}(D_Y \tilde{h}(x, y)) = p + q$. Then for H non-singular

$$D y(x) = H^{-1} A^T (A H^{-1} A^T)^{-1} (A H^{-1} B - C) - H^{-1} B$$

where

$$A = D_Y \tilde{h}(x, y) \in \mathbb{R}^{(p+q) \times m}$$

$$B = D_{XY}^2 f(x, y) - \sum_{i=1}^{p+q} \lambda_i D_{XY}^2 \tilde{h}_i(x, y) \in \mathbb{R}^{m \times n}$$

$$C = D_X \tilde{h}(x, y) \in \mathbb{R}^{(p+q) \times n}$$

$$H = D_{YY}^2 f(x, y) - \sum_{i=1}^{p+q} \lambda_i D_{YY}^2 \tilde{h}_i(x, y) \in \mathbb{R}^{m \times m}$$

and $\lambda \in \mathbb{R}^{p+q}$ satisfies $\lambda^T A = D_Y f(x, y)$ with $\lambda_i \leq 0$ for $i = p + 1, \dots, p + q$. The gradient $D y(x)$ is one-sided whenever there exists an $i > p$ such that $\lambda_i = 0$.

Proof. The existence of λ and non-negativity of λ_i for $i = p + 1, \dots, p + q$ comes from the fact that $y(x)$ is a regular

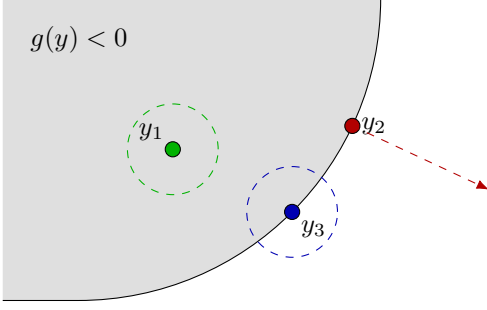


Fig. 3: Illustration of different scenarios for the solution to inequality constrained deep declarative nodes. In the first scenario (y_1) the solution is a local minimum strictly satisfying the constraints. In the second scenario (y_2) the solution is on the boundary of the constraint set with the negative gradient of the objective pointing outside of the set. In the third scenario (y_3) the solution is on the boundary of the constraint set and is also a local minimum.

point (because $D_Y \tilde{h}(x, y)$ is full rank). The remainder of the proof follows Proposition 4.5. One-sidedness of $Dy(x)$ results from the fact that $\lambda_i(x)$, for $i = p + 1, \dots, p + q$, is not differentiable at zero. \square

The main difference between the result for inequality constrained problems (Proposition 4.6) and the simpler equality constrained case (Proposition 4.5) is that the gradient is discontinuous at points where any of the Lagrange multipliers for (active) inequality constraints are zero. We illustrate this by analysing a deep declarative node with a single inequality constraint,

$$y \in \arg \min_{u \in \mathbb{R}^m} f(x, u) \quad \text{subject to} \quad g(x, u) \leq 0 \quad (26)$$

where both f and g are smooth.

Consider three scenarios (see Figure 3). First, if the constraint is inactive at solution y , that is, $g(x, y) < 0$, then we must have $D_Y f(x, y) = 0$ and can take $Dy(x)$ to be the same as for the unconstrained case. Second, if the constraint is active at the solution, that is, $g(x, y) = 0$ but $D_Y f(x, y) \neq 0$ then it must be that $\lambda \neq 0$, and from Proposition 4.6, the gradient $Dy(x)$ is the same as if we had replaced the inequality constraint with an equality. Last, if the constraint is active at y and $D_Y f(x, y) = 0$ then $Dy(x)$ is undefined. In this last scenario we can choose either the unconstrained or constrained gradient in order to obtain an update direction for back-propagation similar to how functions such as rectified linear units are treated in standard deep learning models.

4.4 Simpler equality constraints

Often there is only a single fixed equality constraint that does not depend on the inputs—we consider such problems in Section 5.2—or the equality constraints are all linear. The above result can be specialized for these cases.

Corollary 4.7: Consider functions $f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$ and $h : \mathbb{R}^m \rightarrow \mathbb{R}$. Let

$$y(x) \in \arg \min_{u \in \mathbb{R}^m} f(x, u) \quad \text{subject to} \quad h(u) = 0.$$

Assume that $y(x)$ exists, that $f(x, u)$ and $h(u)$ are second-order differentiable in the neighborhood of $(x, y(x))$ and $y(x)$, respectively, and that $D_Y h(y(x)) \neq 0$. Then

$$Dy(x) = \left(\frac{H^{-1} a a^T H^{-1}}{a^T H^{-1} a} - H^{-1} \right) B$$

where

$$a = (D_Y h(y))^T \in \mathbb{R}^m,$$

$$B = D_{XY}^2 f(x, y) \in \mathbb{R}^{m \times n},$$

$$H = D_{YY}^2 f(x, y) - \lambda D_{YY}^2 h(y) \in \mathbb{R}^{m \times m}, \text{ and}$$

$$\lambda = \left(\frac{\partial h}{\partial y_i}(y) \right)^{-1} \frac{\partial f}{\partial y_i}(x, y) \in \mathbb{R} \text{ for any } \frac{\partial h}{\partial y_i}(y) \neq 0.$$

Proof. Follows from Proposition 4.5 with $p = 1$, $D_X h \equiv 0_{1 \times n}$ and $D_{XY}^2 h \equiv 0_{m \times n}$. \square

Observation 4.8: When the constraint function is independent of x we have $Dy(x) \perp D_Y h(y)$, from Equation 21 with $D_X h = 0$. In other words, y can only change (as a function of x) in directions that maintain the constraint $h(y) = 0$.

Given this simpler form of the gradient for declarative nodes with a single equality constraint, one might be tempted to naively combine multiple equality constraints into a single constraint, for example $\tilde{h}(x, u) \triangleq \sum_{i=1}^p h_i^2(x, u) = 0$ (or any other function of the h_i 's that is identically zero if and only if the h_i 's are all zero). However, this will not work as it violates the assumptions of the method of Lagrange multipliers, namely that $D_Y \tilde{h}(x, y) \neq 0$ at the optimal point.

The following result is for the common case of multiple fixed linear equality constraints that do not depend on the inputs (which has also been reported previously [28]).

Corollary 4.9: Consider a function $f : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}$ and let $A \in \mathbb{R}^{p \times m}$ and $d \in \mathbb{R}^p$ with $\text{rank}(A) = p$ define a set of p under-constrained linear equations $Au = d$. Let

$$y(x) \in \arg \min_{u \in \mathbb{R}^m} f(x, u) \quad \text{subject to} \quad Au = d.$$

Assume that $y(x)$ exists and that $f(x, u)$ is second-order differentiable in the neighborhood of $(x, y(x))$. Then

$$Dy(x) = \left(H^{-1} A^T (A H^{-1} A^T)^{-1} A H^{-1} - H^{-1} \right) B$$

where $H = D_{YY}^2 f(x, y)$ and $B = D_{XY}^2 f(x, y)$.

Proof. Follows from Proposition 4.5 with $h(x, u) \triangleq Au - d$ so that $D_X h \equiv 0_{p \times n}$, $D_{XY}^2 h \equiv 0_{m \times n}$ and $D_{YY}^2 h \equiv 0_{m \times m}$. \square

4.5 Geometric interpretation

Consider, for simplicity, a declarative node with scalar input x ($n = 1$) and a single equality constraint ($p = 1$). An alternative form for the gradient from Proposition 4.5, which lends itself to geometric interpretation as shown in Figure 4, is given by

$$Dy(x) = H^{-\frac{1}{2}} \left(\tilde{b} - (\hat{a}^T \tilde{b}) \hat{a} + \frac{c}{\|\hat{a}\|} \hat{a} \right) \quad (27)$$

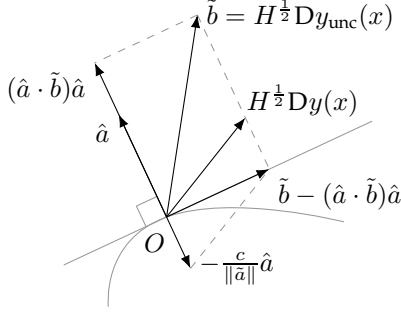


Fig. 4: Geometry of the gradient for an equality constrained optimization problem. The unconstrained gradient $Dy_{\text{unc}}(x)$ is corrected to ensure that the solution remains on the constraint surface after gradient descent with an infinitesimal step size.

where $\tilde{a} = -H^{-\frac{1}{2}}a$, $\tilde{b} = -H^{-\frac{1}{2}}b$ and $\hat{a} = \|\tilde{a}\|^{-1}\tilde{a}$. First, consider the case when the constraints do not depend on the input ($c = 0$). The unconstrained derivative $Dy_{\text{unc}}(x) = H^{-\frac{1}{2}}\tilde{b}$ (equality when $\lambda = 0$) is corrected by removing the component perpendicular to the constraint surface. That is, the unconstrained gradient is projected to the tangent plane of the constraint surface, as shown in Figure 4. The gradient is therefore parallel to the constraint surface, ensuring that gradient descent with an infinitesimal step size does not push the solution away from the constraint surface. Thus, the equation encodes the following procedure: (i) pre-multiply the unconstrained gradient vector $Dy_{\text{unc}}(x)$ by $H^{\frac{1}{2}}$; (ii) project onto the linearly transformed (by $H^{-\frac{1}{2}}$) tangent plane of the constraint surface; and (iv) pre-multiply by $H^{-\frac{1}{2}}$. Observe that the equation compensates for the curvature H before the projection. The same intuition applies to the case where the constraints depend on the inputs ($c \neq 0$), with an additional bias term accounting for how the constraint surface changes with the input x .

4.6 Relationship to conventional neural networks

Deep declarative networks subsume traditional feed-forward and recurrent neural networks in the sense that any layer in the network that explicitly defines its outputs in terms of a differentiable function of its inputs can be reduced to a declarative processing node. A similar observation was made by Amos and Kolter [4], and the notion is formalized in the following proposition.

Proposition 4.10: Let $\tilde{f} : \mathbb{R}^n \rightarrow \mathbb{R}^m$ be an explicitly defined differentiable forward processing function for a neural network layer. Then we can alternatively define the behavior of the layer declaratively as

$$y = \arg \min_{u \in \mathbb{R}^m} \frac{1}{2} \|u - \tilde{f}(x)\|^2.$$

Proof. The objective function $f(x, u) = \frac{1}{2} \|u - \tilde{f}(x)\|^2$ is strongly convex. Differentiating it with respect to u and setting to zero we get $y = \tilde{f}(x)$. By Proposition 4.4 we have $H = I$ and $B = -D_X \tilde{f}(x)$, giving $Dy(x) = D\tilde{f}(x)$. \square

Of course, despite the appeal of having a single mathematical framework for describing processing nodes, there is no reason to do this in practice since imperative and declarative nodes can co-exist in a network.

Composability. One of the key design principles in deep learning is that models should define their output as the composition of many simple functions (such as $y = \tilde{f}(\tilde{g}(x))$). Just like conventional neural networks deep declarative networks can also be composed of many optimization problems arranged in levels as the following example shows:

$$y \in \arg \min_u f(z, u) \quad (28) \\ \text{subject to } z \in \arg \min_v g(x, v).$$

Here the gradients combine by the chain rule of differentiation as one might expect:

$$Dz(x) = -H_g^{-1} B_g(x, z) \\ Dy(z) = -H_f^{-1} B_f(z, y) \quad (29) \\ Dy(x) = Dy(z) Dz(x)$$

where $H_g = D_{ZZ}^2 g(x, z)$, etc. Here, during back propagation in computing $DJ(x)$ from $DJ(y)$ for training objective J , it may be more efficient to first compute $DJ(z)$ as

$$Dz(x) = \underbrace{(Dy(z) Dy(x))}_{DJ(z)} \quad (30)$$

which we will discuss further in Section 6.1.

4.7 Extensions and discussion

4.7.1 Feasibility problems

Feasibility problems, where we simply seek a solution $y(x)$ that satisfies a set of constraints (themselves functions of x), can be formulated as optimization problems [10], and hence declarative nodes. Here the objective is constant (or infinite for an infeasible point) but the problem is more naturally written as

$$\text{find } u \\ \text{subject to } h_i(x, u) = 0, i = 1, \dots, p \quad (31) \\ g_i(x, u) \leq 0, i = 1, \dots, q.$$

The gradient can be derived by removing inactive inequality constraints and maintaining the invariant $\tilde{h}(x, y(x)) = 0$ where $\tilde{h} = [h_1, \dots, h_p, g_1, \dots, g_q]$, i.e., following the constraint surface. Implicit differentiation gives

$$ADy(x) + C = 0 \quad (32)$$

where $A = D_Y \tilde{h}(x, y)$ and $C = D_X \tilde{h}(x, y)$ as defined in Proposition 4.6. This is the same set of linear equations as the second row of the Lagrangian in our proof of Proposition 4.5, and has a unique solution whenever $\text{rank}(A) = m$.³

4.7.2 Non-smooth objective and constraints

Propositions 4.5 and 4.6 showed how to compute derivatives of the solution to parametrized equality constrained optimization problems with second-order differentiable objective and constraint functions (in the neighborhood of the solution). However, we can also define declarative nodes where the objective and constraints are non-smooth (that is, non-differentiable at some points). To enable back-propagation learning with such declarative nodes all we

3. In Proposition 4.5 we assumed that $\text{rank}(A) = p \leq m$. Typically we have $p < m$ and so $ADy(x) = -C$ is under-determined, and the curvature of the objective resolves $Dy(x)$.

require is a local descent direction (or Clarke generalized gradient [13]). This is akin to the approach taken in standard deep learning models with non-smooth activation functions such as the rectified linear unit, $y = \max\{x, 0\}$, which is non-differentiable at $x = 0$.⁴ We explore this in practice when we consider robust pooling and projecting onto the boundary of L_p -balls in our illustrative examples and experiments below.

4.7.3 Non-regular solutions

The existence of the Lagrange multipliers λ (and their uniqueness) in our results above was guaranteed by our assumption that $y(x)$ is a regular point. Moreover, the fact that $A = D_Y h$ is full rank (and H non-singular) ensures that $Dy(x)$ can be computed. Under other constraint qualifications, such as Slater's condition [10], the λ are also guaranteed to exist but may not be unique. Moreover, we may have $\text{rank}(A) \neq p + q$. If $p + q > m$ but A is full rank, which can happen if multiple constraints are active at $y(x)$, then we can directly solve the overdetermined system of equations $ADy(x) = -C$. On the other hand, if A is rank deficient then we have redundant constraints and must reduce A to a full rank matrix by removing linearly dependent rows (and fixing the corresponding λ_i to zero) or resort to using a pseudo-inverse to approximate $Dy(x)$. This can happen if two or more (nonlinear inequality) constraints are active at $y(x)$ and tangent to each other at that point.

4.7.4 Non-unique solutions

In the development of deep declarative nodes we have made no assumption about the uniqueness of the solution to the optimization problem (i.e., output of the node). Indeed, many solutions may exist and our gradients are still valid. Needless to say, it is essential to use the same value of y in the backward pass as was computed during the forward pass (which is automatically handled in modern deep learning frameworks by caching the forward computations). But there are other considerations too.

Consider for now the unconstrained case (Equation 11). If $D_{YY}^2 f(x, y) \succ 0$ then f is strongly convex in the neighborhood of y . Hence y is an isolated minimizer (that is, a unique minimizer within its neighborhood), and the gradient derived in Proposition 4.4 holds for all such points. Nevertheless, when performing parameter learning in a deep declarative network it is important to be consistent in solving optimization problems with multiple minima to avoid oscillating between solutions as we demonstrate with the following pathological example. Consider

$$y \in \arg \min_{u \in \mathbb{R}} 0 \quad \text{subject to} \quad (y - 1)^2 - x^2 = 0 \quad (33)$$

which has solution $y = 1 \pm x$ and, by inspection,

$$Dy = \begin{cases} -1, & \text{for } y = 1 - x \\ +1, & \text{for } y = 1 + x \end{cases} \quad (34)$$

Depending on the choice of optimal solution the gradient of the loss being propagated backwards through the node

4. Incidentally, the (elementwise) rectified linear unit can be defined declaratively as $y = \arg \min_{u \in \mathbb{R}^n} \frac{1}{2} \|u - x\|_2^2$.

will point in opposite directions, which is problematic for end-to-end learning.

Singular Hessians. Now, if $D_{YY}^2 f(x, y)$ is singular then y may not be an isolated minimizer. This can occur, for example, when y is an over-parametrized descriptor of some physical property such as an unnormalized quaternion representation of a 3D rotation. In such cases we cannot use Proposition 4.4 to find the gradient. In fact the gradient is undefined. There are three strategies we can consider for such points. First, reformulate the problem to use a minimal parametrization or introduce constraints to remove degrees of freedom thereby making the solution unique. Second, make the objective function strongly convex around the solution, for example by adding the proximal term $\frac{\delta}{2} \|u - y\|^2$ for some small $\delta > 0$. Third, use a pseudo-inverse to solve the linear equation $D_{YY}^2 f(x, y) \Delta y = -D_{XY}^2 f(x, y)$ for Δy and take Δy as a descent direction. Supposing H^\dagger is a pseudo-inverse of $D_{YY}^2 f(x, y)$, we can compute an entire family of solutions as

$$\Delta y \in \{-H^\dagger B + (I - H^\dagger H)Z \mid Z \in \mathbb{R}^{m \times n}\} \quad (35)$$

where $H = D_{XY}^2 f(x, y)$ and $B = D_{XX}^2 f(x, y)$.

As a toy example, motivated by the quaternion representation, consider the following problem that aligns the output vector with an input vector $x \neq 0$ in \mathbb{R}^4 ,

$$y \in \arg \min_{u \in \mathbb{R}^4} f(x, u) \quad \text{with } f(x, u) \triangleq \frac{-x^\top u}{\|u\|_2} \quad (36)$$

which has solution $y = \alpha x$ for arbitrary $\alpha > 0$. Here we have $D_{YY}^2 f(x, y) = \alpha^{-2} \|x\|_2^{-1} (I - \|x\|_2^{-2} x x^\top)$, which is singular (by the matrix inversion lemma [27]). Fixing one degree of freedom resolves the problem, which in this case is easily done by forcing the output vector to be normalized:

$$y \in \arg \min_{u \in \mathbb{R}^4} -x^\top u \quad \text{subject to} \quad \|u\|_2 = 1. \quad (37)$$

Alternatively we can compute the Moore–Penrose pseudo-inverse [36] of $D_{YY}^2 f(x, y)$ to obtain

$$Dy = \alpha \left(I - \frac{1}{\|x\|_2^2} x x^\top \right) \quad (38)$$

which is the same gradient as the constrained case where the solution is fixed to have $\alpha = 1/\|x\|_2$ for this problem.

5 ILLUSTRATIVE EXAMPLES

In this section we provide examples of unconstrained and constrained declarative nodes that illustrate the theory developed above. For each example we begin with a standard operation in deep learning models, which is usually implemented as an imperative node. We show how the operations can be equivalently implemented in a declarative framework, and then generalize the operations to situations where an explicit implementation is not possible, but where the operation results in more desirable model behavior.

5.1 Robust pooling

Average (or mean) pooling is a standard operation in deep learning models where multi-dimensional feature maps are averaged over one or more dimensions to produce a summary statistic of the input data. For the one-dimensional case, $x \in \mathbb{R}^n \mapsto y \in \mathbb{R}$, we have

$$y = \frac{1}{n} \sum_{i=1}^n x_i \quad (39)$$

and

$$Dy = \left[\frac{\partial y}{\partial x_1}, \dots, \frac{\partial y}{\partial x_n} \right] = \frac{1}{n} \mathbf{1}^\top. \quad (40)$$

As a declarative node the mean pooling operator is

$$y \in \arg \min_{u \in \mathbb{R}} \sum_{i=1}^n \frac{1}{2} (u - x_i)^2 \quad (41)$$

While the solution, and hence gradients, can be expressed in closed form, it is well-known that as a summary statistic the mean is very sensitive to outliers. We can make the statistic more robust by replacing the quadratic penalty function $\phi^{\text{quad}}(z) = \frac{1}{2}z^2$ with one that is less sensitive to outliers. The pooling operation can then be generalized as

$$y \in \arg \min_{u \in \mathbb{R}} \sum_{i=1}^n \phi(u - x_i; \alpha) \quad (42)$$

where ϕ is a penalty function taking scalar argument and controlled by parameter α .

For example, the Huber penalty function defined as

$$\phi^{\text{huber}}(z; \alpha) = \begin{cases} \frac{1}{2}z^2 & \text{for } |z| \leq \alpha \\ \alpha(|z| - \frac{1}{2}\alpha) & \text{otherwise.} \end{cases} \quad (43)$$

is more robust to outliers. The Huber penalty is convex and hence the pooled value can be computed efficiently via Newton's method or gradient descent [10]. However, no closed-form solution exists. Moreover, the solution set may be an interval of points. The pseudo-Huber penalty [29],

$$\phi^{\text{pseudo}}(z; \alpha) = \alpha^2 \left(\sqrt{1 + \left(\frac{z}{\alpha}\right)^2} - 1 \right) \quad (44)$$

has similar behaviour to the Huber but is strongly convex so the optimal solution (pooled value) is unique.

The Welsch penalty function [17] is even more robust to outliers flattening out to a fixed cost as $|z| \rightarrow \infty$,

$$\phi^{\text{welsch}}(z; \alpha) = 1 - \exp\left(-\frac{z^2}{2\alpha^2}\right) \quad (45)$$

However it is non-convex and so obtaining the solution $y(x)$ is non-trivial. Nevertheless, *given a solution* we can use Proposition 4.4 to compute a gradient for back-propagation parameter learning.

Going one step further we can defined the truncated quadratic penalty function,

$$\phi^{\text{trunc}}(z; \alpha) = \begin{cases} \frac{1}{2}z^2 & \text{for } |z| \leq \alpha \\ \frac{1}{2}\alpha^2 & \text{otherwise.} \end{cases} \quad (46)$$

In addition to being non-convex, the function is also non-smooth. The solution amounts to finding the maximal

set of values all within some fixed distance α from the mean. The objective $f(x, u) = \sum_{i=1}^n \phi^{\text{trunc}}(u - x_i; \alpha)$ is not differentiable at points where there exists an i such that $|y(x) - x_i| = \alpha$, in the same way that the rectified linear unit is non-differentiable at zero. Nevertheless, we can still compute a gradient almost everywhere (and take a one-sided gradient at non-differentiable points).

The various penalty functions are depicted in Table 2. When used to define a robust pooling operation there is no closed-form solution for all but the quadratic penalty. Yet the gradient of the solution with respect to the input data for all robust penalties can be calculated using Proposition 4.4 as summarized in the table.

5.2 L_p -sphere or L_p -ball projection

Euclidean projection onto an L_2 -sphere, equivalent to L_2 normalization, is another standard operation in deep learning models. For $x \in \mathbb{R}^n \mapsto y \in \mathbb{R}^n$, we have

$$y = \frac{1}{\|x\|_2} x \quad (47)$$

and

$$Dy(x) = \frac{1}{\|x\|_2} \left(I - \frac{1}{\|x\|_2^2} x x^\top \right). \quad (48)$$

As a declarative node the L_2 -sphere projection operator is

$$y \in \arg \min_{u \in \mathbb{R}^n} \frac{1}{2} \|u - x\|_2^2 \quad \text{subject to } \|u\|_2 = 1. \quad (49)$$

While the solution, and hence gradients, can be expressed in closed-form, it may be desirable to use other L_p -spheres or balls, for regularization, sparsification, or to improve generalization performance [38], as well as for adversarial robustness. The projection operation onto an L_p -sphere can then be generalized as

$$y_p \in \arg \min_{u \in \mathbb{R}^n} \frac{1}{2} \|u - x\|_2^2 \quad \text{subject to } \|u\|_p = 1 \quad (50)$$

where $\|\cdot\|_p$ is the L_p -norm.

For example, projecting onto the L_1 -sphere has no closed-form solution, however Duchi et al. [22] provide an efficient $O(n)$ solver. Similarly, projecting onto the L_∞ -sphere has an efficient (trivial) solver. Given a solution from one of these solvers, we can use Proposition 4.5 to compute a gradient. For both cases, the constraint functions h are non-smooth and so are not differentiable whenever the optimal projection y lies on an $(n - k)$ -face for $2 \leq k \leq n$. For example, when y lies on a vertex, changes in x may not have any effect on y . While the gradient obtained from Proposition 4.5 provides a valid local descent direction (Clarke generalized gradient [13]), it can be modified to prefer a zero gradient at these plateau dimensions by masking the gradient, and thereby becoming identical to the gradient obtained by numerical differentiation methods.

The various constraint functions and gradients are given in Table 1. When used to define a projection operation, only the L_2 case has a closed-form solution. Nonetheless, the gradient of the solution with respect to the input data for all constraint functions can be calculated using Proposition 4.5 as summarized in the table.

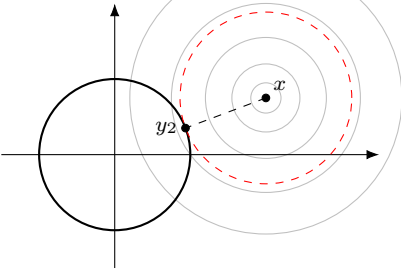
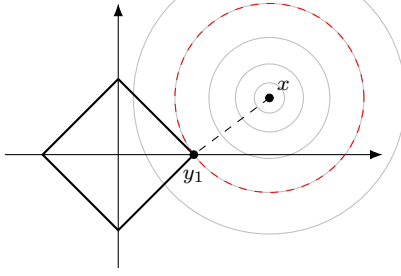
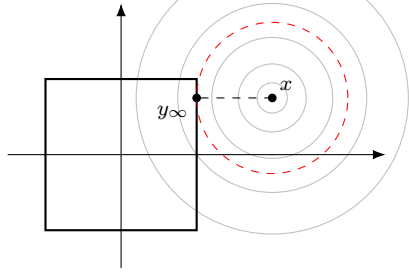
	L_2	L_1	L_∞
$h(u)$	$\ u\ _2 - 1 = \sqrt{\sum_{i=1}^n u_i^2} - 1$	$\ u\ _1 - 1 = \sum_{i=1}^n u_i - 1$	$\ u\ _\infty - 1 = \max_i \{ u_i \} - 1$
			
	closed-form, smooth, unique [†] solution	non-smooth (C^0), isolated solutions	non-smooth (C^0), isolated solutions
$D_Y h(y)$	y	$\text{vec} \{\text{sign}(y_i)\}^T$	$\text{vec} \{\llbracket i \in I^* \rrbracket \text{sign}(y_i)\}^T$ $I^* = \{i \mid y_i \geq y_j \forall j\}$
$D_{YY}^2 h(y)$	$I - yy^T$	$0_{n \times n}$	$0_{n \times n}$
λ	$1 - \ x\ _2$	$\text{sign}(y_i)(y_i - x_i) \forall i$	$\llbracket i \in I^* \rrbracket \text{sign}(y_i)(y_i - x_i) \forall i \in I^*$
$Dy(x)$	$\frac{1}{\ x\ _2} (I - yy^T)$	$I - \frac{D_Y h(y)^T D_Y h(y)}{D_Y h(y) D_Y h(y)^T}$	$I - \frac{D_Y h(y)^T D_Y h(y)}{D_Y h(y) D_Y h(y)^T}$
$D^z y(x)$	$\frac{1}{\ x\ _2} (I - yy^T)$	$\text{diag}(D_Y h(y)) - \frac{D_Y h(y)^T D_Y h(y)}{D_Y h(y) D_Y h(y)^T}$	$I - \text{diag}(D_Y h(y))$

TABLE 1: The gradient of the Euclidean projection onto various L_p -spheres with constraint functions h , when it exists. The projection is found as $y(x) = \arg \min_{u \in \mathbb{R}^n} f(x, u)$ subject to $h(u) = 0$, with $f(x, u) = \frac{1}{2} \|u - x\|_2^2$. In all cases, $B = D_{XY}^2 f(x, y) = -I$, and $D_{YY}^2 f(x, y) = I$. The plots show the Euclidean projection of an example point $x \in \mathbb{R}^2$ onto L_p -spheres for $p = 1, 2$ and ∞ . Despite not being able to solve the problem in closed form for L_1 and L_∞ constraints, given a solution the gradient $Dy(x)$ can still be calculated. While $Dy(x)$ provides a valid local descent direction in all cases, the gradient can be altered to prefer a zero gradient along plateau dimensions by zeroing the rows and columns corresponding to the zero (L_1) or non-zero (L_∞) elements of $D_Y h(y)$. This gradient $D^z y(x)$ is obtained by masking $Dy(x)$ with pp^T for L_1 or $\bar{p}\bar{p}^T$ for L_∞ , where $p = |D_Y h(y)|$ is treated as a Boolean vector. [†]Except for $x = 0$ where the solution is non-isolated (the entire sphere). For L_1 and L_∞ the solution is unique when x is outside the ball but can be non-unique for x inside the ball.

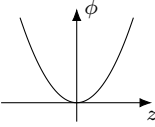
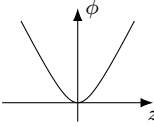
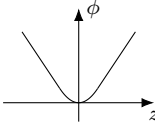
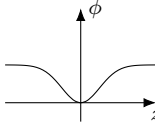
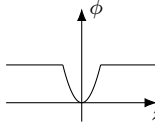
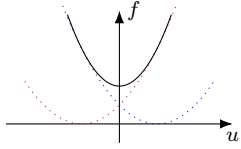
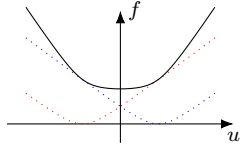
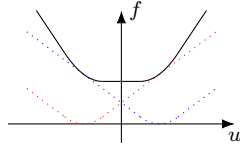
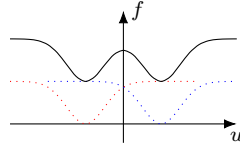
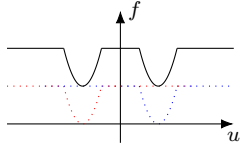
	QUADRATIC	PSEUDO-HUBER	HUBER	WELSCH	TRUNC. QUAD.
$\phi(z; \alpha)$	$\frac{1}{2}z^2$ 	$\alpha^2 \left(\sqrt{1 + \left(\frac{z}{\alpha}\right)^2} - 1 \right)$ 	$\begin{cases} \frac{1}{2}z^2 & \text{for } z \leq \alpha \\ \alpha(z - \frac{1}{2}\alpha) & \text{otherwise.} \end{cases}$ 	$1 - \exp\left(-\frac{z^2}{2\alpha^2}\right)$ 	$\begin{cases} \frac{1}{2}z^2 & \text{for } z \leq \alpha \\ \frac{1}{2}\alpha^2 & \text{otherwise.} \end{cases}$ 
					
	closed-form, convex, smooth, unique solution	convex, smooth, unique solution	convex, non-smooth (C^1), non-isolated solutions	non-convex, smooth, isolated solutions	non-convex, non-smooth (C^0), isolated solutions
$D_{YY}^2 f(x, y)$	n	$\sum_{i=1}^n \left(1 + \left(\frac{y-x_i}{\alpha} \right)^2 \right)^{-3/2}$	$\sum_{i=1}^n \mathbb{I}[y-x_i \leq \alpha]$	$\sum_{i=1}^n \frac{\alpha^2 - (y-x_i)^2}{\alpha^4} \exp\left(-\frac{(y-x_i)^2}{2\alpha^2}\right)$	$\sum_{i=1}^n \mathbb{I}[y-x_i \leq \alpha]$
$D_{XY}^2 f(x, y)$	-1_n^T	$\text{vec} \left\{ - \left(1 + \left(\frac{y-x_i}{\alpha} \right)^2 \right)^{-3/2} \right\}^T$	$\text{vec} \{ -\mathbb{I}[y-x_i \leq \alpha] \}$	$\text{vec} \left\{ \frac{(y-x_i)^2 - \alpha^2}{\alpha^4} \exp\left(-\frac{(y-x_i)^2}{2\alpha^2}\right) \right\}^T$	$\text{vec} \{ -\mathbb{I}[y-x_i \leq \alpha] \}^T$
$Dy(x)$	$\frac{1}{n} 1_n^T$	$\text{vec} \left\{ \frac{w_i}{\sum_{j=1}^n w_j} \right\}^T$ where $w_i = \left(1 + \left(\frac{y-x_i}{\alpha} \right)^2 \right)^{-3/2}$	$\text{vec} \left\{ \frac{\mathbb{I}[y-x_i \leq \alpha]}{\sum_{j=1}^n \mathbb{I}[y-x_j \leq \alpha]} \right\}^T$	$\text{vec} \left\{ \frac{w_i}{\sum_{j=1}^n w_j} \right\}^T$ where $w_i = \frac{\alpha^2 - (y-x_i)^2}{\alpha^4} \exp\left(-\frac{(y-x_i)^2}{2\alpha^2}\right)$	$\text{vec} \left\{ \frac{\mathbb{I}[y-x_i \leq \alpha]}{\sum_{j=1}^n \mathbb{I}[y-x_j \leq \alpha]} \right\}^T$

TABLE 2: The gradient of the estimate for a robust mean over a vector of values for various penalty functions $\phi(z; \alpha)$ when it exists. The robust estimate is found as $y(x) = \arg \min_{u \in \mathbb{R}} f(x, u)$ with $f(x, u) = \sum_{i=1}^n \phi(u - x_i; \alpha)$. The first plot (row 2) shows the penalty function, the second plot (row 3) shows an example $f(x, u)$ for $x = (-2\alpha, 2\alpha) \in \mathbb{R}^2$. Plots have been drawn at different scales to enhance visualization of shape differences between the penalty functions. Despite not being able to solve the problem in closed form for most penalty functions, given a solution the gradient $Dy(x)$ can still be calculated. Notice the similarity in gradient forms due to the objective f being composed of a sum of independent penalties, each penalty symmetric in x and y . Moreover, the Huber and truncated quadratic have exactly the same gradient form (when it exists) even though their solutions may be different. Under some conditions Huber and Welsch may result in a zero $D_{YY}^2 f$ and, hence, an undefined gradient. However, we did not see this in practice. More importantly, computation of the Welsch gradient is sensitive to numerical underflow and care should be taken to appropriately scale the w_i 's before dividing by their sum.

We can also consider a declarative node that projects onto the unit L_p -ball with output defined as

$$y_p^\circ \in \arg \min_{u \in \mathbb{R}^n} \frac{1}{2} \|u - x\|_2^2 \quad (51)$$

subject to $\|u\|_p \leq 1$

where we now have an inequality constrained convex optimization problem (for $p \geq 1$). Here we take the gradient Dy_p° to be zero if $\|x\|_p < 1$ and Dy_p otherwise. In words, we set the gradient to zero if the input already lies inside the unit ball. Otherwise we use the gradient obtained from projecting onto the L_p -sphere.

6 IMPLEMENTATION CONSIDERATIONS

In this section we provide some practical considerations relating to the implementation of deep declarative nodes.

6.1 Vector–Jacobian product

As discussed above, the key challenge for declarative nodes is in computing $Dy(x)$ for which we have provided linear algebraic expressions in terms of first- and second-order derivatives of the objective and constraint functions. In computing the gradient of the loss function, J , for the end-to-end model we compute

$$DJ(x) = DJ(y)Dy(x) \quad (52)$$

and the order in which we evaluate this expression can have a dramatic effect on computational efficiency.

For simplicity consider the unconstrained case (Proposition 4.4) and let $v^\top = DJ(y) \in \mathbb{R}^{1 \times m}$. We have

$$DJ(x) = -v^\top H^{-1}B \quad (53)$$

where $H \in \mathbb{R}^{m \times m}$ and $B \in \mathbb{R}^{m \times n}$. The expression can be evaluated in two distinct ways—either as $(v^\top H^{-1})B$ or as $v^\top (H^{-1}B)$ where parentheses have been used to highlight calculation order. Assuming that H^{-1} has been factored (and contains no special structure), the cost of evaluating $DJ(x) \in \mathbb{R}^{1 \times n}$ is $O(m^2 + mn)$ for the former and $O(m^2 n)$ for the latter, and thus there is a computational advantage to evaluating Equation 53 from left to right.

Moreover, it is often the case that $n \gg m$ in deep learning models, such as for robust pooling. Here, rather than pre-computing B , which would require $O(mn)$ bytes of precious GPU memory to store, it is much more space efficient to compute the elements of $DJ(x)$ iteratively as

$$DJ(x)_i = \tilde{v}^\top b_i \quad (54)$$

where $\tilde{v} = -H^{-1}v$ is cached and b_i is the i -th column of B computed on the fly and therefore only requiring $O(m)$ bytes.

6.2 Automatic differentiation

The forward processing function for some deep declarative nodes (such as those defined by convex optimization problems) can be implemented using generic solvers. Indeed, Agrawal et al. [1] provide a general framework for specifying (convex) deep declarative nodes through disciplined convex optimization and providing methods for both the

forward and backward pass. However, many other interesting nodes will require specialized solvers for their forward functions (e.g., the coordinate-based SDP solver proposed by Wang et al. [47]). Even so, the gradient calculation in the backward pass can be implemented using generic automatic differentiation techniques whenever the objective and constraint functions are twice continuously differentiable (in the neighborhood of the solution). For example, the following Python code makes use of the autograd package to compute the gradient of an arbitrary unconstrained deep declarative node with twice differentiable, first-order objective f at minimum y given input x .

Python Code

```
1 import autograd.numpy as np
2 from autograd import grad, jacobian
3
4 def gradient(f, x, y):
5     fY = grad(f, 1)
6     fYY = jacobian(fY, 1)
7     fXY = jacobian(fY, 0)
8
9     return -1.0 * np.linalg.solve(fYY(x,y), fXY(x,y))
```

If called repeatedly (such as during learning) the partial derivative functions fY , fYY and fXY can be pre-processed and cached. And of course the gradient can instead be manually coded for special cases and when the programmer chooses to introduce memory and speed efficiencies. The implementation for equality and inequality constrained problems follows in a straightforward way and is omitted here for brevity. We provide full Python and PyTorch reference implementations and examples at <http://deepdeclarativenetworks.com>.

7 EXPERIMENTS

We conduct experiments on standard image and point cloud classification tasks to assess the viability of applying declarative networks to challenging computer vision problems. Our goal is not to obtain state-of-the-art results. Rather we aim to validate the theory presented above and demonstrate how declarative nodes can easily be integrated into non-trivial deep learning pipelines. To this end, we implement the pooling and projection operations from Section 5. For efficiency, we use the symbolic derivatives obtained in Tables 2 and 1, and never compute the Jacobian directly, instead computing the vector–Jacobian product to reduce the memory overhead. All code is available at <http://deepdeclarativenetworks.com>.

For the point cloud classification experiments with robust pooling, we use the ModelNet40 CAD dataset [48], with 2048 points sampled per object, normalized into a unit ball [40]. Each model is trained using stochastic gradient descent for 60 epochs with an initial learning rate of 0.01, decaying by a factor of 2 every 20 epochs, momentum (0.9), and a batch size of 24. All models, variations of PointNet [40] implemented in PyTorch, have 0.8M parameters. Importantly, pooling layers do not add additional parameters.

The results are shown in Tables 3 and 4, for a varying fraction of outliers seen during training and testing (Table 3) or only during testing, that is trained without outliers (Table 4). We report top-1 accuracy and mean Average Precision

(mAP) on the test set. Outlier points are sampled uniformly from the unit ball, randomly replacing existing inlier points, following the same protocol as Qi et al. [40]. The robust pooling layer replaces the max pooling layer in the model, with the quadratic penalty function being denoted by Q, pseudo-Huber by PH, Huber by H, Welsch by W, and truncated quadratic by TQ. The optimizer for the last two (non-convex) functions is initialized to both the mean and the median, and then the lowest local minimum selected as the solution. It is very likely that RANSAC [26] search would produce better results, but this was not tested. For all experiments, the robustness parameter α (loosely, the inlier threshold) is set to one.

We observe that the robust pooling layers perform significantly better than max pooling and quadratic (average) pooling when outliers are present, especially when not trained with the same outlier rate. There is a clear trend that, as the outlier rate increases, the most accurate model tends to be more robust model (further towards the right).

For the image classification experiments with L_p -sphere and L_p -ball projection, we use the ImageNet 2012 dataset [16], with the standard single central 224×224 crop protocol. Each model is trained using stochastic gradient descent for 90 epochs with an initial learning rate of 0.1, decaying by a factor of 10 every 30 epochs, weight decay ($1e-4$), momentum (0.9), and a batch size of 256. All models, variations of ResNet-18 [30] implemented in PyTorch, have 11.7M parameters. As with the robust pooling layers, projection layers do not add additional parameters.

The results are shown in Table 5, with top-1 accuracy, top-5 accuracy, and mean Average Precision (mAP) reported on the validation set. The projection layer, prepended by a batch normalization layer with no learnable parameters, is inserted before the final fully-connected output layer of the network. The features are pre-scaled according to the formula $\frac{2}{3} \text{median}_i \|f_i\|_p$ where f_i are the batch-normalized training set features of the penultimate layer of the pre-trained ResNet model. This corresponds to scaling factors of 250, 15, and 3 for $p = 1, 2$, and ∞ respectively, and can be thought of as varying the radius of the L_p -ball. The chosen scale ensures that the projection affects most features, but is not too aggressive.

The results indicate that feature projection improves the mAP significantly, with a more modest increase in top-1 and top-5 accuracy. This suggests that projection encourages a more appropriate level of confidence about the predictions, improving the calibration of the model.

8 CONCLUSION

In the preceding sections we have presented some theory and practice for deep declarative networks. On the one hand we developed nothing new—argmin can simply be viewed as yet another function and all that is needed is to work out the technical details of how to compute its derivative. On the other hand deep declarative networks offers a new way of thinking about network design with expressive processing functions and constraints but without having to worry about implementation details (or having to back-propagate gradients through complicated algorithmic procedures).

TABLE 3: The effect of robust pooling layers on point cloud classification results for the ModelNet40 dataset [48], with varying percentages of outliers (O) and the same rate of outliers seen during training and testing. Outliers points are uniformly sampled from the unit ball. PointNet [40] is compared to our variants that replace max pooling with robust pooling: quadratic (Q), pseudo-Huber (PH), Huber (H), Welsch (W), and truncated quadratic (TQ), all trained from scratch. Top-1 accuracy and mean average precision are reported.

O %	Top-1 Accuracy %						Mean Average Precision $\times 100$					
	[40]	Q	PH	H	W	TQ	[40]	Q	PH	H	W	TQ
0	88.4	84.7	84.7	86.3	86.1	85.4	95.6	93.8	95.0	95.4	95.0	93.8
10	79.4	84.3	85.6	85.5	86.6	85.5	89.4	94.3	94.6	95.1	94.6	94.7
20	76.2	84.8	84.8	85.2	86.3	85.5	87.8	94.8	95.0	95.0	94.8	95.0
50	72.0	84.0	83.1	83.9	84.3	83.9	83.3	93.8	93.5	94.3	94.8	94.8
90	29.7	61.7	63.4	63.1	65.3	61.8	38.9	76.8	78.7	78.5	79.1	76.6

TABLE 4: The effect of robust pooling layers on point cloud classification results for the ModelNet40 dataset [48], with varying percentages of outliers (O) and no outliers seen during training. During testing, outlier points are uniformly sampled from the unit ball. Models are identical to those in Table 3. Top-1 accuracy and mean average precision are reported.

O %	Top-1 Accuracy %						Mean Average Precision $\times 100$					
	[40]	Q	PH	H	W	TQ	[40]	Q	PH	H	W	TQ
0	88.4	84.7	84.7	86.3	86.1	85.4	95.6	93.8	95.0	95.4	95.0	93.8
1	32.6	84.9	84.7	86.4	86.2	85.3	48.6	93.8	95.1	95.3	95.1	93.0
10	6.47	83.9	84.6	85.3	86.0	85.9	8.20	93.4	94.8	94.4	94.9	93.9
20	5.95	79.6	82.8	81.1	84.7	84.9	7.73	91.9	93.4	92.7	94.2	94.6
30	5.55	70.9	74.2	72.2	77.6	83.2	6.00	87.8	89.5	85.1	90.9	92.8
40	5.35	55.3	59.1	55.4	63.1	75.6	6.41	77.6	80.2	72.7	83.2	90.6
50	4.86	32.9	36.0	34.6	44.1	57.9	5.68	62.3	60.2	60.1	66.4	85.3
60	4.42	14.5	16.2	18.1	27.1	30.6	5.08	39.1	36.3	38.5	42.7	68.5
70	4.25	5.03	6.33	7.95	14.1	11.9	4.66	22.5	19.3	18.4	25.7	47.9
80	3.11	4.10	4.51	5.64	8.88	5.11	4.21	10.8	8.91	8.98	14.9	26.7
90	3.72	4.06	4.06	4.30	5.68	4.22	4.49	8.20	5.98	5.80	8.37	9.78

TABLE 5: The effect of projection layers on image classification results for the ImageNet 2012 dataset [16]. Top-1 and top-5 accuracy, and mean Average Precision (mAP) are reported. All models are trained from scratch, with the exception of ResNet-18-pt, which uses the PyTorch pre-trained weights.

Model	Top-1 Acc. %	Top-5 Acc. %	mAP $\times 100$
ResNet-18	69.80	89.26	58.97
ResNet-18-pt	69.76	89.08	53.76
ResNet-18- L_1 Sphere	69.92	89.38	62.72
ResNet-18- L_2 Sphere	70.66	89.60	71.97
ResNet-18- L_∞ Sphere	70.03	89.22	63.98
ResNet-18- L_1 Ball	70.17	89.47	61.26
ResNet-18- L_2 Ball	70.59	89.70	72.43
ResNet-18- L_∞ Ball	70.06	89.29	63.33

By facilitating the inclusion of declarative nodes into end-to-end learnable models, network capacity can be directed towards identifying features and patterns in the data rather than having to re-learn an approximation for theory that is already well-established (for example, physical system models) or enforce constraints that we know must hold (for example, that the output must lie on a manifold). As such, with deep declarative networks, we have the potential

to create more robust models that can generalize better from less training data (and with fewer parameters) or, indeed, approach problems that could not be tackled by deep learning previously as demonstrated in recent works such as Amos and Kolter [4] and Wang et al. [47].

As with any new approach there are some shortcomings and much still to do. For example, it is possible for problems with parametrized constraints to become infeasible during learning, so care should be taken to avoid parameterizations that may result in an empty feasible set. More generally, optimization problems can be expensive to solve and those with multiple solutions can create difficulties for end-to-end learning. Also more theory needs to be developed around non-smooth objective functions. We have presented some techniques for dealing with problems with inequality constraints (when we may only have a single-sided gradient) or when the KKT optimality conditions cannot be guaranteed but there are many other avenues to explore. Along these lines it would also be interesting to consider the trade-off in finding general descent directions rather than exact gradients and efficient approximations to the forward processing function, especially in the early stages of learning.

The work of Chen et al. [11] that shows how to differentiate through an ordinary differential equation also presents interesting directions for future work. Viewing such models as declarative nodes with differential constraints suggests many extensions including coupling ordinary differential equations with constrained optimization problems, which may be useful for physical simulations or modeling dynamical systems.

Finally, the fact that deep declarative nodes provide some guarantees on their output suggests that a principled approach to analyzing the end-to-end behavior of a deep declarative network might be possible. This would be especially useful in deploying deep learned models in safety critical applications such as autonomous driving or complex control systems. And while there are many challenges yet to overcome, the hope is that deep declarative networks will deliver solutions to problems faced by existing deep neural networks leading to better robustness and interpretability.

ACKNOWLEDGMENTS

We give warm thanks to Bob Williamson and John Lloyd for helpful discussions, and anonymous reviewers for insightful suggestions. We thank Itzik Ben-Shabat for recommending experiments on point cloud classification.

REFERENCES

- [1] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.
- [2] A. Agrawal, S. Barratt, S. Boyd, E. Busseti, and W. M. Moursi. Differentiating through a cone program. Technical report, Stanford University (arXiv:1904.09043), April 2019.
- [3] B. Amos. *Differentiable Optimization-Based Modeling for Machine Learning*. PhD thesis, Carnegie Mellon University, 2019.
- [4] B. Amos and J. Z. Kolter. OptNet: Differentiable optimization as a layer in neural networks. In *Proc. of the International Conference on Machine Learning (ICML)*, 2017.
- [5] B. Amos, I. D. J. Rodriguez, J. Sacks, B. Boots, and J. Z. Kolter. Differentiable MPC for end-to-end planning and control. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.
- [6] S. Bai, Z. Kolter, and V. Koltun. Deep equilibrium models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.
- [7] J. F. Bard. *Practical Bilevel Optimization: Algorithms and Applications*. Kluwer Academic Press, 1998.
- [8] D. P. Bertsekas. *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press, 1982.
- [9] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, 2004.
- [10] S. P. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge, 2004.
- [11] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.
- [12] A. Cherian, B. Fernando, M. Harandi, and S. Gould. Generalized rank pooling for action recognition. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.
- [13] F. H. Clarke. Generalized gradients and applications. *Transactions of the American Mathematical Society*, 205: 247–262, 1975.
- [14] F. de Avila Belbute-Peres, K. Smith, K. Allen, J. Tenenbaum, and J. Z. Kolter. End-to-end differentiable physics for learning and control. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.
- [15] S. Dempe and S. Franke. On the solution of convex bilevel optimization problems. *Computational Optimization and Applications*, pages 1–19, 2015.
- [16] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A large-scale hierarchical image database. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.
- [17] J. E. Dennis Jr and R. E. Welsch. Techniques for nonlinear least squares and robust regression. *Communications in Statistics - Simulation and Computation*, 7(4):345–359, 1978.
- [18] J. Djolonga and A. Krause. Differentiable learning of submodular models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
- [19] C. B. Do, C.-S. Foo, and A. Y. Ng. Efficient multiple hyperparameter learning for log-linear models. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2007.
- [20] J. Domke. Generic methods for optimization-based modeling. In *Proc. of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012.
- [21] A. L. Dontchev and R. T. Rockafellar. *Implicit Functions and Solution Mappings: A View from Variational Analysis*. Springer-Verlag, 2nd edition, 2014.
- [22] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra. Efficient projections onto the l_1 -ball for learning in high dimensions. In *Proc. of the International Conference on*

- Machine Learning (ICML)*, pages 272–279, 2008.
- [23] B. Fernando and S. Gould. Learning end-to-end video classification with rank-pooling. In *Proc. of the International Conference on Machine Learning (ICML)*, 2016.
 - [24] B. Fernando and S. Gould. Discriminatively learned hierarchical rank pooling networks. In *International Journal of Computer Vision (IJCV)*, volume 124, pages 335–355, Sep 2017.
 - [25] B. Fernando, E. Gavves, J. M. Oramas, A. Ghodrati, and T. Tuytelaars. Modeling video evolution for action recognition. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015.
 - [26] M. A. Fischler and R. C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.
 - [27] G. H. Golub and C. F. van Loan. *Matrix Computations*. Johns Hopkins University Press, 3rd edition, 1996.
 - [28] S. Gould, B. Fernando, A. Cherian, P. Anderson, R. Santa Cruz, and E. Guo. On differentiating parameterized argmin and argmax problems with application to bi-level optimization. Technical report, Australian National University (arXiv:1607.05447), July 2016.
 - [29] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004.
 - [30] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
 - [31] T. Klatzer and T. Pock. Continuous hyper-parameter learning for support vector machines. In *Computer Vision Winter Workshop (CVWW)*, 2015.
 - [32] S. G. Krantz and H. R. Parks. *The Implicit Function Theorem: History, Theory, and Applications*. Springer, 2013.
 - [33] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
 - [34] K. Lee, S. Maji, A. Ravichandran, and S. Soatto. Meta-learning with differentiable convex optimization. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.
 - [35] P. Márquez-Neila, M. Salzmann, and P. Fua. Imposing hard constraints on deep networks: Promises and limitations. In *CVPR Workshop on Negative Results in Computer Vision*, 2017.
 - [36] C. D. Meyer. Generalized inversion of modified matrices. *SIAM Journal on Applied Math.*, 24(3):315–323, 1973.
 - [37] P. Ochs, R. Ranftl, T. Brox, and T. Pock. Bilevel optimization with nonsmooth lower level problems. In *International Conference on Scale Space and Variational Methods in Computer Vision (SSVM)*, 2015.
 - [38] S. Oymak. Learning compact neural networks with regularization. In *Proc. of the International Conference on Machine Learning (ICML)*, pages 3963–3972, 2018.
 - [39] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. In *NeurIPS Autodiff Workshop*, 2017.
 - [40] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. PointNet: Deep learning on point sets for 3d classification and segmentation. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.
 - [41] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.
 - [42] K. G. G. Samuel and M. F. Tappen. Learning optimized MAP estimates in continuously-valued MRF models. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.
 - [43] R. Santa Cruz, B. Fernando, A. Cherian, and S. Gould. Visual permutation learning. *IEEE Trans. on Pattern Analysis and Machine Intelligence (PAMI)*, 2018.
 - [44] G. M. Scarpello and D. Ritelli. Historical outline of the theorem of implicit functions. *Divulgaciones Matematica*, 10:171–180, 2002.
 - [45] S. Tschiatschek, A. Sahin, and A. Krause. Differentiable submodular maximization. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2018.
 - [46] H. von Stackelberg, D. Bazin, L. Urch, and R. R. Hill. *Market structure and equilibrium*. Springer, 2011.
 - [47] P.-W. Wang, P. L. Donti, B. Wilder, and Z. Kolter. SATNet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver. In *Proc. of the International Conference on Machine Learning (ICML)*, 2019.
 - [48] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1912–1920, 2015.