

An Introduction to an Application of the Implicit Function Theorem

Justin Chiu
Cornell Tech
jtc257@cornell.edu

August 4, 2021

Abstract

Gradient-based learning forms the foundation of modern machine learning, and automatic differentiation allows ML practitioners to easily compute gradients. While automatic differentiation only costs a constant multiple of the time and space required to evaluate a function, it has its limitations. In particular, when evaluating a function itself is expensive, the direct application of automatic differentiation is infeasible. In this report, we review the implicit function theorem (IFT) and its use in reducing the cost of computing gradients in scenarios where function evaluation is expensive, focusing on the application of the IFT to differentiating through the solutions of optimization problems.

1 Introduction

Gradient-based learning underpins many of the recent successes in machine learning, particularly advances involving neural networks. The key to the success of gradient-based methods is automatic differentiation (AD), which greatly increases the development speed of machine learning research by allowing practitioners to circumvent the error-prone and time-consuming process of computing gradients manually. AD operates by reducing functions into compositions of atomic operations, for which we have a library of derivatives for, and composing those derivatives via the chain rule. The underlying concept behind AD is that a program's execution trace is a valid and useful representation of a function [Griewank and Walther, 2008]. **Maybe a diagram of an execution trace would be good here**

Storing the execution trace of a program allows AD systems to easily compute derivatives. However, longer execution traces can quickly consume a large amount of memory. Consider an iterative method, such as gradient descent, whose execution trace takes the form of an unrolled loop: Given an initial point $\theta = x_0$, iterates x_1, x_2, \dots, x_K are produced by running the gradient descent update for K iterations. In order to differentiate through this procedure with AD (i.e. compute $\frac{dx_K}{d\theta}$), we have to store all the x_k iterates as well the computation used to produce them. Thus, the memory complexity of storing this execution trace scales linearly in the number of iterations K as well as the dimensionality of the iterates x_k . For large K , this can be infeasible. One method for

32 overcoming the space complexity’s dependence on the number of iterations K in the above example
33 is to use the implicit function theorem (IFT), letting you throw away x_1 through x_{K-1} while still
34 being able to compute the derivative $\frac{dx_K}{d\theta}$.

35 Large execution traces are not uncommon; optimization problems are often solved with iterative
36 methods, resulting in traces very similar to the example given above. In order to speed up solvers
37 for these problems, one could learn an initialization by differentiating through the execution trace
38 of the iterative method [Finn et al., 2017, Kim et al., 2018, Venkataraman and Amos, 2021]. In this
39 report, we will cover the use of the IFT as a method for dealing with the space complexity of AD in
40 exactly these cases. In particular, we will focus on applying the IFT to differentiating the solutions
41 to optimization problems [Amos and Kolter, 2017, Agrawal et al., 2019].

42 2 Related Work

43 There are a variety of methods for reducing the space limitations of AD, of which we only mention
44 three: checkpointing, reversible computation, and implicit differentiation.

45 The first method, checkpointing, improves space complexity at the cost of time [Griewank and
46 Walther, 2008]. Rather than storing the full execution trace of a program, checkpointing instead
47 recomputes values when needed. This can result in a slowdown due to recomputation, and also
48 requires careful choosing of which part of the trace to checkpoint and recompute.

49 A second method is reversible computation [Maclaurin et al., 2015, Gomez et al., 2017], which
50 improves space complexity at the cost of expressivity, but not speed. Reversible computation en-
51 sures that a function’s derivative depends only on the output, allowing the input to be discarded
52 during function evaluation. This is typically accomplished by ensuring that the input is easily re-
53 constructed from the output, restricting the expressivity of layers.

54 A third method uses the IFT, which we focus on. Application of the IFT potentially improves
55 space complexity at the cost of stronger assumptions. The IFT gives conditions under which deriva-
56 tives can be computed independent of intermediate computation, with the primary condition being
57 the characterization of the output as the solution to a system of equations.

58 One of the main applications where the space complexity of AD limits its scalability is bilevel
59 optimization. Bilevel optimization problems are, as implied by the name, optimization problems
60 with another nested inner optimization problem embedded within. Methods for solving bilevel
61 optimization typically proceed iteratively. For every iteration when solving the outer optimization
62 problem, we must additionally solve an inner optimization problem.

63 The application we focus on in this report is expressing individual layers of a neural network
64 declaratively as the solution of an optimization problem [Amos and Kolter, 2017, Agrawal et al.,
65 2019, Gould et al., 2019]. This allows models to learn, without heavy manual specification, the
66 constraints of the problem in addition to the parameters of the objective. An example of this is
67 learning to play Sudoku from only input-output pairs [Amos and Kolter, 2017].

68 Other applications that can be formulated as bilevel optimization problems are hyperparame-
69 ter optimization, metalearning, and variational inference. Hyperparameter optimization formulates
70 hyperparameter tuning as a bilevel optimization problem, as for each hyperparameter configuration
71 a new model must be trained as the inner loop [Maclaurin et al., 2015, Lorraine et al., 2019b,a,

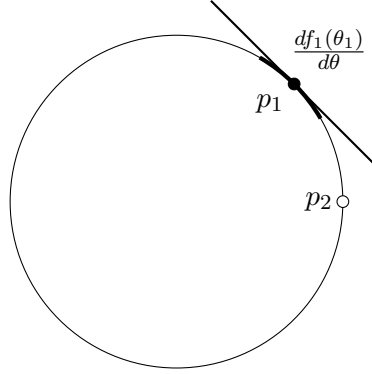


Figure 1: A circle, defined by the relation $F(\theta, x) = \theta^2 + x^2 - 1 = 0$. We view θ as a parameter, x as a solution to $F(\theta, x) = 0$ given θ and the pair (θ, x) is a solution point. Our goal is to compute the derivative $\frac{dx}{d\theta}$. While we cannot differentiate the relation F directly as it is not a function, we can compute the derivative at the solution point $p_1 = (\theta_1, x_1)$ using the local parameterization (or solution mapping) $f_1(\theta) = \sqrt{1 - x^2}$, yielding $\frac{dx}{d\theta} = \frac{df_1(\theta)}{d\theta}$. This parameterization holds in a neighbourhood around p_1 , visualized as an arc. We cannot use the same parameterization at $p_2 = (\theta_2, x_2)$ as the derivative is undefined. In general, the IFT is most useful in cases more complicated than the unit circle, where local parameterizations are too complex to easily write down.

72 Bertrand et al., 2020]. Derivatives must then be propagated through the inner training loop to the
 73 outer hyperparameter loop. Similarly, metalearning learns the parameters of a model such that the
 74 model is able to quickly be adapted to a new task via gradient descent [Finn et al., 2017, Rajeswaran
 75 et al., 2019]. This is accomplished by differentiating through the learning procedure of each new
 76 task. Finally, a variant of variational inference follows a very similar format: semi-amortized vari-
 77 ational inference (SAVI) aims to learn a model that is able to provide a good initialization for vari-
 78 ational parameters that are subsequently updated iteratively to maximize a lower bound objective
 79 [Kim et al., 2018]. This is also accomplished by differentiating through the iterative optimization
 80 procedure applied to the variational parameters during inference.

81 In all the above applications, the inner-loop optimization problem is solved with an iterative
 82 method, except in rare, simple cases. The IFT reduces the memory footprint of automatic differen-
 83 tiation, which would otherwise be difficult to scale.

84 3 The Implicit Function Theorem

85 The implicit function theorem (IFT) has a long history, as well as many applications in a wide
 86 variety of fields such as economics and differential geometry. For an overview of the history of the
 87 IFT and some of its classical applications in mathematics and economics, see the book by Krantz
 88 and Parks [2003].

89 Consider the unit circle, governed by the relation $F(\theta, x) = \theta^2 + x^2 - 1 = 0$, which can be

interpreted as a system of equations. As F fails the vertical line test, we cannot write x as a function of θ globally. This prevents us from taking derivatives, for example $\frac{dx}{d\theta}$. However, we can use local parameterizations: $f_1(\theta) = \sqrt{1-x^2}$ if $x > 0$ or $f_2(\theta) = -\sqrt{1-x^2}$ if $x < 0$. Note that the local parameterizations are functions that hold only within a neighbourhood of a particular solution point (θ, x) ; there may be solution points where we simply cannot compute a particular derivative, e.g. $\frac{dx}{d\theta}$ when $x = 0$. These local parameterizations then allow us compute the derivative $\frac{dx}{d\theta}$ at particular solution points (θ, x) using the corresponding parameterization. See Fig. 1 for an illustration. The IFT generalizes this example, and formalizes the conditions under which there exist continuous local parameterizations for a given relation or system of equations.

While the unit circle in this example has very simple local parameterizations, in general local parameterizations can be more complicated.¹ Additionally, the IFT does not give the form of the local parameterizations; it only guarantees the existence of one around a point and a way to compute its derivative. The local parameterization is left implicit, hence the ‘implicit’ in IFT.

Formally, given a system of equations $F(\theta, x) = \mathbf{0}_m$,² and a solution point $(\theta, x) \in \mathbb{R}^n \times \mathbb{R}^m$, the IFT gives sufficient conditions under which x can locally be written as a function of just the parameters θ within a neighbourhood of the solution point (θ, x) . We saw this in the unit circle example, where the local parameterizations were valid around a particular solution point. We refer to this function $x^*(\theta) = x$ as a solution mapping, x a solution, and θ as parameters. These conditions are as follows:

1. We have a solution point (θ, x) that satisfies the system of equations $F(\theta, x) = 0$.
2. F has at least continuous first derivatives: $F \in \mathcal{C}^1$.
3. The Jacobian matrix of F wrt x evaluated at the solution point (θ, x) is nonsingular: $\det \frac{dF(\theta, x)}{dx} \neq 0$.

Assuming these conditions hold for F at (θ, x) , the IFT asserts the existence of the implicit solution mapping $x^*(\theta) : \mathbb{R}^n \rightarrow \mathbb{R}^m$ (at the solution point (θ, x)), and that the derivative of the solution mapping is given by $\frac{dx^*(\theta)}{d\theta} = -[\frac{dF(\theta, x)}{dx}]^{-1} \frac{dF(\theta, x)}{d\theta} \in \mathbb{R}^{m \times n}$.

Notation: Jacobian Matrix Given a function $F : \mathbb{R}^n \times \mathbb{R}^m \rightarrow \mathbb{R}^m$, we denote the Jacobian matrix evaluated at the point $(\theta, x) \in \mathbb{R}^n \times \mathbb{R}^m$ as

$$\frac{dF(\theta, x)}{d(\theta, x)} = \begin{bmatrix} \frac{dF(\theta, x)}{d\theta} & \frac{dF(\theta, x)}{dx} \end{bmatrix},$$

where we have the matrix of partial derivatives

$$\frac{dF(\theta, x)}{d\theta} = \begin{bmatrix} \frac{dF_1(\theta, x)}{d\theta_1} & \dots & \frac{dF_1(\theta, x)}{d\theta_n} \\ \vdots & \ddots & \vdots \\ \frac{dF_m(\theta, x)}{d\theta_1} & \dots & \frac{dF_m(\theta, x)}{d\theta_n} \end{bmatrix} \in \mathbb{R}^{m \times n},$$

¹ Recall that a program trace is a valid representation of a function. One could have a local parameterization that is the trace of a long program, such as one that consists of a series of iterative updates.

² We denote vectors and matrices of all 1s and 0s by $\mathbf{1}_S$ and $\mathbf{0}_S$, where S denotes the shape, i.e. $\mathbf{0}_m \in \mathbb{R}^m$.



Figure 2: An example relationship between the parameters θ , solution x_K after K iterations of an iterative method, and implicit function $x^*(\theta)$ of the IFT. The rectangle depicts a space which, in this example, contains both θ and x_K . This is not necessary for the IFT, but simplifies illustration. The parameters θ provide an initial point, which is then iteratively refined into solution x_K , shown by the squiggly line. If x_K satisfies the conditions of the IFT, then the IFT both guarantees the existence of the implicit solution mapping $x_K = x^*(\theta)$ (dashed line) and tells us how to compute $\frac{dx^*(\theta)}{d\theta}$. This is useful if the execution trace of the iterative procedure (squiggly line) is too expensive to store in memory for use in automatic differentiation.

116 and similarly for $\frac{dF(\theta, x)}{dx} \in \mathbb{R}^{m \times m}$.

117 We can now proceed to use the IFT to compute derivatives of the solution of an optimization
 118 problem wrt parameters of the problem without storing intermediate computations, as illustrated
 119 in Fig. 2. We will use the optimality criteria of the optimization problem to define a system of
 120 equations, then apply the IFT to compute the Jacobian of the solution wrt the parameters. This
 121 methodology allows us to use the solution to an optimization problem as the output of a layer
 122 within a neural network, as done in OptNet [Amos and Kolter, 2017].

123 4 Embedding Optimization inside a Neural Network

124 As an introductory example, we will replace the softmax layer of a neural network with an equiva-
 125 lent function defined as the output of an optimization problem, then derive derivatives using the IFT.
 126 We will start by reviewing softmax and its expression as an optimization problem. After checking
 127 the conditions of the IFT hold, we can then compute derivatives. Since the Jacobian of softmax is
 128 known, we can directly verify that the IFT gives the correct answer.

129 4.1 Softmax

130 Softmax is often used to parameterize categorical distributions within neural networks, such as in
 131 attention layers. Given n items with independent utilities, where $\theta \in \mathbb{R}^n$ indicate preferences,
 132 softmax gives the following distribution over items: $z_i = \frac{\exp(\theta_i)}{\sum_j \exp(\theta_j)}$, with $z \in \mathbb{R}^n$. While there
 133 is a closed-form equation for both softmax and its Jacobian, we use it as an introduction to the

134 mechanism behind OptNet (and other differentiable optimization layers) [Amos and Kolter, 2017,
135 Agrawal et al., 2019].

The output of softmax is the solution of the following constrained optimization problem [Gao and Pavel, 2018]:

$$\begin{aligned} & \text{maximize} && z^\top \theta + H(z) \\ & \text{subject to} && z^\top \mathbf{1} = 1 \\ & && z_i \geq 0, \forall i, \end{aligned} \tag{1}$$

136 where $H(z) = -\sum_i z_i \log z_i$ is the entropy. The first term in the objective, $z^\top \theta$, is highest when z
137 points in the same direction as θ . Given the constraint that z must sum to one and have nonnegative
138 entries, maximizing just the first term subject to those constraints results in a solution that picks out
139 the highest component of θ , i.e. $\text{argmax}(\theta)$. The addition of the entropy term penalizes solutions
140 that put too much mass on just a few items, resulting in a smoothed version of argmax . We will
141 refer to this entropy-regularized version as the softmax problem.³

142 Our goal is to compute the Jacobian of softmax $\frac{dz}{d\theta} = \frac{d\text{softmax}(\theta)}{d\theta}$ using the IFT and the opti-
143 mization problem above. While this may seem trivial because softmax has a closed form expression
144 for both the output and Jacobian, it is a worthwhile exercise in applying the IFT. Applying the IFT
145 to optimization problems consists of four steps:

- 146 1. Find a solution to the optimization problem.
- 147 2. Write down the system of equations.
- 148 3. Check that the conditions of the IFT hold.
- 149 4. Compute the derivative of the implicit solution mapping wrt the parameters.

150 We assume the first step has been done for us, and we have a solution z to the softmax problem.⁴
151 We will then use the IFT to compute gradients of z wrt the parameters θ by following the rest of the
152 steps.

153 Step 2: The KKT conditions determine the system of equations

154 Given an optimization problem, the Karush-Kuhn-Tucker (KKT) conditions determine a system of
155 equations that the solution must satisfy, i.e the optimality criteria [Karush, 1939, Kuhn and Tucker,
156 1951]. They are, roughly, stationarity (the gradient should be 0 at a local optima) and feasibility
157 (the constraints of the problem should not be violated). For a thorough introduction to the KKT
158 conditions, see chapter 5 of Boyd and Vandenberghe [2004] or the Wikipedia article

We will use the KKT conditions of the softmax problem in Eqn. 1 to determine the function F in the IFT. First, we introduce dual variables $u \in \mathbb{R}$, $v \in \mathbb{R}^n$ and write out the Lagrangian:

$$\mathcal{L}(\theta, z, u, v) = z^\top \theta + H(z) + u(z^\top \mathbf{1}_n - 1) + v^\top z.$$

³ Removing the entropy regularization term results in the argmax optimization problem: maximize $z^\top \theta$, subject to $z^\top \mathbf{1}_n = 1$ and $z \succeq 0$.

⁴ This is trivial for softmax since we can compute it using the closed form expression. However, in more general optimization problems, we would obtain z from a solver.

We therefore have the solution point (θ, z, u, v) , with parameters θ and solution $x = (z, u, v)$. We then have the following necessary conditions for a solution (z, u, v) , i.e. the KKT conditions:

$$\begin{aligned}\frac{d}{dx}\mathcal{L}(\theta, z, u, v) &= \mathbf{0}_n \quad (\text{stationarity}) \\ u(z^\top \mathbf{1} - 1) &= 0 \quad (\text{primal feasibility, equality}) \\ \text{diag}(v)z &= \mathbf{0}_n \quad (\text{complementary slackness}) \\ z &\succeq \mathbf{0}_n \quad (\text{primal feasibility, inequality}) \\ v &\succeq \mathbf{0}_n \quad (\text{dual feasibility})\end{aligned}\tag{2}$$

As we only need a system of equations with $2n + 1$ equations to determine the $2n + 1$ solution variables $x = (z, u, v)$, we use the first three conditions: stationary, primal feasibility (equality), and complementary slackness.

In full, the system of equations $F(\theta, z, u, v) = 0$ is

$$\begin{aligned}\theta + -\log(z) - 1 + u\mathbf{1}_n + v &= \mathbf{0}_n \\ u(z^\top \mathbf{1}_n - 1) &= 0 \\ \text{diag}(v)z &= \mathbf{0}_n.\end{aligned}\tag{3}$$

Note that the first and third equations are vector-valued.

This completes the second step, where we chose a subset of the KKT conditions in order to produce a nonlinear system of equations. We can now proceed check the conditions of the IFT, which will determine whether F is locally well-behaved at a particular solution point (θ, z, u, v) .

Step 3: Check the conditions of the IFT

The IFT requires the following three conditions, which must be checked on a case-by-case basis for particular solution points:

- $F(\theta, z, u, v) = 0$,
- F has at least continuous first derivatives,
- $\det \frac{dF(\theta, z, u, v)}{d(z, u, v)} \neq 0$, or equivalently $\frac{dF(\theta, z, u, v)}{d(z, u, v)}$ is full rank.

The first condition holds as we have a solution to the optimization problem and F was chosen using the KKT conditions.⁵ The second condition also holds, as F has continuous first derivatives. All that remains is to check the third condition, that the Jacobian matrix $\frac{dF(\theta, z, u, v)}{d(z, u, v)}$ (evaluated at the solution point) is non-singular.

The Jacobian matrix $\frac{dF(\theta, z, u, v)}{d(z, u, v)} \in \mathbb{R}^{2n+1 \times 2n+1}$ is given by

$$\frac{dF}{d(z, u, v)} = \begin{bmatrix} \text{diag}(z)^{-1} & -\mathbf{1}_n & -I_{n \times n} \\ u\mathbf{1}_n^\top & z^\top \mathbf{1}_n - 1 & 0 \\ \text{diag}(v) & 0 & \text{diag}(z) \end{bmatrix}.\tag{4}$$

⁵ Recall that softmax also has a closed form expression.

176 Since a solution must be feasible, we know that $z^\top \mathbf{1} - 1 = 0$ and $u > 0$. However, the upper
 177 left block, $\text{diag}(z)^{-1}$, contains a divide-by-zero term if any component $z_i = 0$.⁶ To avoid this, we
 178 consider only strictly positive $z \succ 0$ for the IFT.⁷ Constraining to strict positivity for z , we can
 179 deduce that the Jacobian of F is full rank and therefore has nonzero determinant. This shows that
 180 the conditions of the IFT hold for the solution points that are feasible, optimal, and have strictly
 181 positive z .

182 **Step 4: Compute $\frac{dx}{d\theta}$**

183 Now that we have a set of solution points where conditions of the IFT hold, we can use the IFT
 184 to compute $\frac{dz}{d\theta}$. Recall that we have the solution $x = (z, u, v)$; we will switch to x for brevity.
 185 The second part of the IFT tells us that we can compute the Jacobian of the solution mapping
 186 $\frac{dx}{d\theta} = \frac{dx^*(\theta)}{d\theta} = \left[\frac{dF(\theta, x)}{dx} \right]^{-1} \frac{dF(\theta, x)}{d\theta}$, then pick out the relevant components.

The second term, $\frac{dF(\theta, x)}{d\theta}$, is simple. Since θ only appears in the first vector-valued function of F (see Eqn. 3), we have

$$\frac{dF(\theta, x)}{d\theta} = \begin{bmatrix} I_{n \times n} \\ \mathbf{0}_{(n+1) \times (n+1)} \end{bmatrix}. \quad (5)$$

187 The large amount of sparsity allows us to skip some computation further down.⁸

Next, we have to invert the Jacobian from Eqn. 4:

$$\left[\frac{dF(\theta, x)}{dx} \right]^{-1} = \begin{bmatrix} \text{diag}(z)^{-1} & -\mathbf{1}_n & -I_{n \times n} \\ u\mathbf{1}_n^\top & x^\top \mathbf{1}_n - 1 & 0 \\ \text{diag}(v) & 0 & \text{diag}(z) \end{bmatrix}^{-1}. \quad (6)$$

The remainder of this section is compute-intensive; feel free to skip ahead to Sec. 4.2 for a discussion on the limitations of applying the IFT. We use the block-wise inversion formula

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} (A - BD^{-1}C)^{-1} & 0 \\ 0 & (D - CA^{-1}B)^{-1} \end{bmatrix} \begin{bmatrix} I & -BD^{-1} \\ -CA^{-1} & I \end{bmatrix},$$

where

$$\begin{aligned} A &= \begin{bmatrix} \text{diag}(z)^{-1} & -\mathbf{1}_n \\ u\mathbf{1}_n^\top & 0 \end{bmatrix} & B &= \begin{bmatrix} -I_{n \times n} \\ 0 \end{bmatrix} \\ C &= [\text{diag}(v) \quad 0] & D &= \text{diag}(z). \end{aligned}$$

⁶ This term was obtained by differentiating the entropy term of the Lagrangian, $H(z) = \sum_i z_i \log z_i$. While we could use the convention $0 \log 0 = 0$, this does not fix the divide-by-zero issue with the second derivative, which we see here.

⁷ We saw a similar issue in the unit circle example, where the derivative $\frac{dx}{d\theta}$ was undefined when $x = 0$ (see Fig. 1).

⁸ This sparsity is due to the simple constraints in the softmax problem, which is no longer available in more general optimization problems.

However, by complementary slackness, we have $v = 0$,⁹ reducing the above to

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} & 0 \\ 0 & D^{-1} \end{bmatrix} \begin{bmatrix} I_{(n+1) \times (n+1)} & -BD^{-1} \\ 0 & I_{n \times n} \end{bmatrix}.$$

As we are interested in computing $\frac{dz}{d\theta}$, rather than the full derivative $\frac{dx}{d\theta}$ (recall $x = (z, u, v)$), in addition to the sparsity of $\frac{dF}{d\theta}$, we only have to solve for the upper-left $n \times n$ block of $A^{-1} \in \mathbb{R}^{(n+1) \times (n+1)}$. To do so, we will repeat the same block-wise inverse computation. Let us denote

$$A = \begin{bmatrix} \text{diag}(z)^{-1} & -\mathbf{1}_n \\ u\mathbf{1}_n^\top & 0 \end{bmatrix} = \begin{bmatrix} E & F \\ G & H \end{bmatrix}.$$

First, we compute the Schur complement of A ,

$$A/E = H - GE^{-1}F = 0 + u\mathbf{1}_n^\top \text{diag}(z)\mathbf{1}_n = uz^\top \mathbf{1}_n. \quad (7)$$

Since z is feasible, we have $A/E = u$ due to the equality constraints (z must sum to 1 as a probability mass function). Then, we have

$$A^{-1} = \begin{bmatrix} \text{diag}(z)^{-1} & -\mathbf{1}_n \\ u\mathbf{1}_n^\top & 0 \end{bmatrix}^{-1} = \begin{bmatrix} E & F \\ G & H \end{bmatrix}^{-1} = \begin{bmatrix} E^{-1} + E^{-1}F(A/E)^{-1}GE^{-1} & -E^{-1}F(A/E)^{-1} \\ -(A/E)^{-1}GE^{-1} & (A/E)^{-1} \end{bmatrix}. \quad (8)$$

Plugging in,

$$\begin{aligned} A^{-1} &= \begin{bmatrix} \text{diag}(z) - \text{diag}(z)\mathbf{1}_n u^{-1} u\mathbf{1}_n^\top \text{diag}(z) & \text{diag}(z)\mathbf{1}_n u^{-1} \\ -u^{-1} u\mathbf{1}_n^\top \text{diag}(z) & u^{-1} \end{bmatrix} \\ &= \begin{bmatrix} \text{diag}(z) - zz^\top & u^{-1}z \\ -z^\top & u^{-1} \end{bmatrix}. \end{aligned} \quad (9)$$

188 Pulling out the top-left $n \times n$ block yields the Jacobian $\frac{dz}{d\theta} = \text{diag}(z) - zz^\top$, which agrees with
 189 directly differentiating softmax [Martins and Astudillo, 2016].

190 Given the formulation of softmax as an optimization problem, we have successfully shown how
 191 to differentiate the output of softmax wrt the parameters in a solver-agnostic manner using the IFT.
 192 This concludes the exercise of using the IFT to differentiate through the softmax problem.

193 4.2 Limitations

194 In order to compute the derivative $\frac{dx}{d\theta}$, we had to invert the Jacobian of F , i.e. compute $\left[\frac{dF(\theta, x)}{dx}\right]^{-1}$.
 195 However, The first part of F (recall from Eqn. 3), the stationarity condition $\frac{d}{dx}\mathcal{L} = 0$, already
 196 involved the Jacobian of the Lagrangian \mathcal{L} . In general, this means that in order to apply the IFT
 197 to solutions of optimization problems, we must compute the inverse Hessian of the Lagrangian
 198 (or at least a Hessian-vector-product). The Hessian is a matrix of size $O(n^2)$, and inverting this

⁹ We apply the IFT to solutions where $z \succ 0$ due to a divide-by-zero issue in $\frac{d^2 H(z)}{dz_i^2} = \frac{1}{z_i}$.

would take $O(n^3)$ computation.¹⁰ Thankfully, there are relatively cheap ways of approximating this computation, such as with approximate (inverse) Hessian-vector-product techniques [Rajeswaran et al., 2019, Lorraine et al., 2019a].

4.3 Extensions

The methods we covered can be extended to variations of argmax problems other than the softmax problem.¹¹ The softmax problem altered the argmax problem by introducing entropy regularization. Rather than regularizing with entropy, one could instead alter the objective to find the Euclidean projection of the parameters onto the probability simplex, resulting in SparseMax [Martins and Astudillo, 2016]. While the output of softmax variants often have a closed form expression, the IFT provides another way of deriving their Jacobians and could potentially pave the way for differentiating through argmax problems that do not have closed-form expressions.

More generally, the IFT can be applied to cases where, unlike softmax, we do not have an explicit functional form (i.e., the unit circle), and outputs are governed only by a system of equations. This includes more general optimization problems, such as quadratic programs [Amos and Kolter, 2017] or other convex optimization problems [Agrawal et al., 2019].

5 OptNet

OptNet generalizes the methodology applied above to the softmax problem by extending the optimization problems considered, in particular including parameterized constraints. This allows us to learn not only the objective, but also the constraints. Explicitly incorporating families of constraints in models with optimization layers to allows them to perform well on tasks with rigid constraints, such as learning to play Sudoku from only inputs and outputs [Amos and Kolter, 2017].

OptNet applies the IFT to quadratic programs (QPs) in particular. As the simplest nonlinear optimization problem, QPs strike a balance between expressivity and computational tractability [Frank and Wolfe, 1956]. The methodology remains the same as the softmax problem: Given a QP and a solution, use the KKT conditions to produce a system of equations then apply the IFT / implicit differentiation to compute the derivative of the solution wrt the parameters of the objective and constraints.

Quadratic programs take the following form:

$$\begin{aligned} & \text{minimize} && \frac{1}{2}z^\top Qz + q^\top z \\ & \text{subject to} && Gz \leq h \\ & && Az = b, \end{aligned} \tag{10}$$

where we optimize over $z \in \mathbb{R}^n$ and the parameters are $\theta = \{Q, q, A, b, G, h\}$, with $Q \in \mathbb{R}^{n \times n} \succeq 0, q \in \mathbb{R}^n, A \in \mathbb{R}^{m \times n}, b \in \mathbb{R}^m, G \in \mathbb{R}^{p \times n}, h \in \mathbb{R}^p$. Compared to the softmax problem in Eqn. 1,

¹⁰ The number of solution variables scales with the number of primal variables, but also the number of constraints. This potentially makes applying the IFT to optimization problems with exponentially many constraints difficult.

¹¹ The argmax problem is given by maximize $z^\top \theta$, subject to $z^\top \mathbf{1}_n = 1$ and $z \succeq 0$.

the main difference is the learnable parameters in the constraints. As the application of the IFT is very similar, we will not cover it in as much detail as the softmax problem.

Similar to the softmax problem, we will first assume we already have a solution, provided by a solver.¹²

Step 2: The KKT conditions determine the system of equations

The Lagrangian of the QP is given by

$$\mathcal{L}(\theta, z, u, v) = \frac{1}{2}z^\top Qz + q^\top z + u^\top (Gz - h) + v^\top (Az - b),$$

introducing dual variables $u \in \mathbb{R}^p, v \in \mathbb{R}^m$, where we have the solution $x = (z, u, v)$.

We use the following subset of the KKT conditions to determine our system of equations: stationarity, primal feasibility (equality), and complementary slackness. This yields, written in full,

$$\begin{aligned} Qz + q + G^\top u + A^\top v &= \mathbf{0}_n \\ \text{diag}(u)(Gz - h) &= \mathbf{0}_p \\ \text{diag}(v)(Az - b) &= \mathbf{0}_m. \end{aligned} \tag{11}$$

Step 3: Check the conditions of the IFT

The only nontrivial part is checking that the Jacobian matrix of $\frac{dF(\theta, x)}{dx}$ is not singular at the solution point (θ, x) . See the proof of Theorem 1 in the OptNet paper for more details [Amos and Kolter, 2017].

Step 4: Compute $\frac{dx}{d\theta}$

We could then compute $\frac{dx}{d\theta} = \left[\frac{dF(\theta, x)}{dx} \right]^{-1} \frac{dF(\theta, x)}{d\theta}$ by hand or numerically. Amos and Kolter [2017] also show how to obtain the vector-Jacobian product efficiently using quantities readily available from a QP solver.

6 Semi-Amortized Variational Inference (POSTPONED / dont read)

We now apply the IFT to variational inference.

Variational inference has found success in recent applications to generative models, in particular by allowing practitioners to depart from conjugate models and extend emission models with expressive neural network components. The main insight that led to this development is that inference can be amortized through the use of an inference network. One approach to variational inference,

¹² One contribution of OptNet was the extension of a state-of-the-art interior point solver [Amos and Kolter, 2017], and its adaptation to parallel machines (GPUs) and batch processing. While outside the scope of this report, see the paper by Amos and Kolter [2017] for the details.

stochastic variational inference (SVI), introduces local, independent variational parameters for every instance of hidden variable. While flexible, the storage of all variational parameters is expensive, and the optimization of each parameter independently slow []. Amortized variational inference (AVI) solves that by instead using a hierarchical process. Variational parameters are produced hierarchically via an inference network, which in turn generates the local variational parameters []. The resulting local parameters may or may not be subsequently optimized.

Failure to further optimize local variational parameters may result in an amortization gap []. Prior work has shown that this gap can be ameliorated by performing a few steps of optimization on the generated local parameters obtained from the inference network, and even by propagating gradients through the optimization process. Optimizing through the inner optimization problem results in semi-amortized variational inference (SAVI) [].

As our main motivating example, we will examine whether we can apply the IFT to SAVI. We will start by formalizing the problem of variational inference for a simple model.

We will start with a model defined by the following generative process, used by Dai et al. [2019] to analyze posterior collapse:

1. Choose a latent code from the prior distribution $z \sim p(z) = N(0, I)$.
2. Given the code, choose an observation from the emission distribution $x | z \sim p_\theta(x | z) = N(\mu_x(z, \theta), \gamma I)$,

where $\mu_x(z, \theta) \equiv \text{MLP}(z, \theta)$ and $\gamma > 0$ is a hyperparameter. This yields the joint distribution $p(x, z) = p(x | z)p(z)$.

Since the latent code z is unobserved, training this model would require optimizing the evidence $p(x) = \int p(x, z)$. However, due to the MLP parameterized μ_x , the integral is intractable. Variational inference performs approximate inference by introducing variational distribution $q_\phi(z | x)$ and maximizing the following lower bound on $\log p(x)$:

$$\log p(x) - D_{\text{KL}}[q(z | x) || p(z | x)] = \mathbb{E}_{q_\phi(z|x)} \left[\log \frac{p_\theta(x, z)}{q_\phi(z | x)} \right] = L(\theta, \phi). \quad (12)$$

(Write out objective in full.)

While SVI introduces local parameters for each instance of z , and AVI uses a single $q(z | x)$ for all instances, we will follow the approach of SAVI. We will perform inference as follows: For each instance x , produce local variational parameter $z^{(0)} = g(x; \phi)$. Obtain z^* by solving $\mathcal{L}(\theta, z^{(0)}) = 2$, with (local) optima ℓ^* . Take gradients through the whole procedure, i.e. compute $\frac{\partial \ell^*}{\partial \phi} = \frac{\partial \ell^*}{\partial z^*} \frac{\partial z^*}{\partial z^{(0)}} \frac{\partial z^{(0)}}{\partial \lambda}$. The main difficulty lies in computing $\frac{\partial z^*}{\partial z^{(0)}}$. (Highlight challenge)

In order to avoid the memory costs of storing all intermediate computation performed in a solver, we will instead apply the IFT. In order to apply the IFT, we must satisfy the three conditions. First, we must have a solution point to a system of equations, $F(x_0, z_0) = 0$. In this setting, we will use the KKT conditions of the optimization problem to define F .

7 Limitations

References

- A. Agrawal, B. Amos, S. T. Barratt, S. P. Boyd, S. Diamond, and J. Z. Kolter. Differentiable convex optimization layers. *CoRR*, abs/1910.12430, 2019. URL <http://arxiv.org/abs/1910.12430>.
- B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks. *CoRR*, abs/1703.00443, 2017. URL <http://arxiv.org/abs/1703.00443>.
- Q. Bertrand, Q. Klopfenstein, M. Blondel, S. Vaiter, A. Gramfort, and J. Salmon. Implicit differentiation of lasso-type models for hyperparameter optimization, 2020.
- S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, March 2004. ISBN 0521833787. URL <http://www.amazon.com/exec/obidos/redirect?tag=citeulike-20&path=ASIN/0521833787>.
- B. Dai, Z. Wang, and D. P. Wipf. The usual suspects? reassessing blame for VAE posterior collapse. *CoRR*, abs/1912.10702, 2019. URL <http://arxiv.org/abs/1912.10702>.
- C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017. URL <http://arxiv.org/abs/1703.03400>.
- M. Frank and P. Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956. URL <https://EconPapers.repec.org/RePEc:wly:navlog:v:3:y:1956:i:1-2:p:95-110>.
- B. Gao and L. Pavel. On the properties of the softmax function with application in game theory and reinforcement learning, 2018.
- A. N. Gomez, M. Ren, R. Urtasun, and R. B. Grosse. The reversible residual network: Backpropagation without storing activations. *CoRR*, abs/1707.04585, 2017. URL <http://arxiv.org/abs/1707.04585>.
- S. Gould, R. Hartley, and D. Campbell. Deep declarative networks: A new hope. *CoRR*, abs/1909.04866, 2019. URL <http://arxiv.org/abs/1909.04866>.
- A. Griewank and A. Walther. *Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation*. SIAM, Philadelphia, 2nd edition, 2008.
- W. Karush. *Minima of functions of several variables with inequalities as side conditions*. PhD thesis, Thesis (S.M.)—University of Chicago, Department of Mathematics, December 1939., 1939.
- Y. Kim, S. Wiseman, A. C. Miller, D. Sontag, and A. M. Rush. Semi-amortized variational autoencoders, 2018.

- 310 S. Krantz and H. Parks. The implicit function theorem : History, theory, and applications / s.g.
311 krantz, h.r. parks. 01 2003. doi: 10.1007/978-1-4612-0059-8.
- 312 H. W. Kuhn and A. W. Tucker. Nonlinear programming. In *Proceedings of the Second Berkeley*
313 *Symposium on Mathematical Statistics and Probability, 1950*, pages 481–492, Berkeley and Los
314 Angeles, 1951. University of California Press.
- 315 J. Lorraine, P. Vicol, and D. Duvenaud. Optimizing millions of hyperparameters by implicit differ-
316 entiation. *CoRR*, abs/1911.02590, 2019a. URL <http://arxiv.org/abs/1911.02590>.
- 317 J. Lorraine, P. Vicol, and D. Duvenaud. Optimizing millions of hyperparameters by implicit differ-
318 entiation. *CoRR*, abs/1911.02590, 2019b. URL <http://arxiv.org/abs/1911.02590>.
- 319 D. Maclaurin, D. Duvenaud, and R. P. Adams. Gradient-based hyperparameter optimization through
320 reversible learning, 2015.
- 321 A. F. T. Martins and R. F. Astudillo. From softmax to sparsemax: A sparse model of attention and
322 multi-label classification. *CoRR*, abs/1602.02068, 2016. URL <http://arxiv.org/abs/1602.02068>.
- 324 A. Rajeswaran, C. Finn, S. M. Kakade, and S. Levine. Meta-learning with implicit gradients. *CoRR*,
325 abs/1909.04630, 2019. URL <http://arxiv.org/abs/1909.04630>.
- 326 S. Venkataraman and B. Amos. Neural fixed-point acceleration for convex optimization. *CoRR*,
327 abs/2107.10254, 2021. URL <https://arxiv.org/abs/2107.10254>.

328 **A Example Appendix**

329 Neural ODEs use reversibility.