On Differentiating Parameterized Argmin and Argmax Problems with Application to Bi-level Optimization

Stephen Gould[1,2], Basura Fernando[1,3], Anoop Cherian[1,3], Peter Anderson[1,2], Rodrigo Santa Cruz[1,2], and Edison Guo[1,2]

[1]Australian Centre for Robotic Vision
[2] Research School of Computer Science, ANU
[3] Research School of Engineering, ANU
<firstname.lastname>@anu.edu.au

July 21, 2016

Abstract

Some recent works in machine learning and computer vision involve the solution of a bi-level optimization problem. Here the solution of a parameterized lower-level problem binds variables that appear in the objective of an upper-level problem. The lower-level problem typically appears as an argmin or argmax optimization problem. Many techniques have been proposed to solve bi-level optimization problems, including gradient descent, which is popular with current end-to-end learning approaches. In this technical report we collect some results on differentiating argmin and argmax optimization problems with and without constraints and provide some insightful motivating examples.

1 Introduction

Bi-level optimization has a long history of study dating back to the early 1950s and investigation of the so-called Stackelberg model [Bard, 1998]. In this model, two players—a market leader and a market follower—compete for profit determined by the price that the market is willing to pay as a function of total goods produced and each player's production cost. Recently, bi-level optimization problems have found application in machine learning and computer vision where they have been applied to parameter and hyper-parameter learning [Do et al., 2007, Domke, 2012, Klatzer and Pock, 2015], image denoising [Samuel and Tappen, 2009, Ochs et al., 2015], and most recently, video activity recognition [Fernando and Gould, 2016, Fernando et al., 2016].

A bi-level optimization problem consists of an upper problem and a lower problem. The former defines an objective over two sets of variables, say $\boldsymbol{x}$ and $\boldsymbol{y}$. The latter binds $\boldsymbol{y}$ as a function of $\boldsymbol{x}$, typically by solving a minimization problem. Formally, we can write the problem as

$$\begin{array}{ll} \text{minimize}_{\boldsymbol{x}} & f^U(\boldsymbol{x}, \boldsymbol{y}) \\ \text{subject to} & \boldsymbol{y} \in \operatorname{argmin}_{\boldsymbol{y}'} f^L(\boldsymbol{x}, \boldsymbol{y}') \end{array} \tag{1}$$

where f^U and f^L are the upper- and lower-level objectives, respectively. As can be seen from the structure of the problem, the lower-level (follower) optimizes its objective subject to the

value of the upper-level variable $\boldsymbol{x}$. The goal of the upper-level (leader) is to choose $\boldsymbol{x}$ (according to its own objective) knowing that the lower-level will follow optimally.

In one sense the argmin appearing in the lower-level problem is just a mathematical function and so bi-level optimization can be simply viewed as a special case of constrained optimization. In another sense it is useful to consider the structure of the problem and study bi-level optimization in its own right, especially when the argmin cannot be computed in closed-form. As such, many techniques have been proposed for solving different kinds of bi-level optimization problems [Bard, 1998, Dempe and Franke, 2015]. In this technical report we focus on first-order gradient based techniques, which have become very important in machine learning and computer vision with the wide spread adoption of deep neural network models [LeCun et al., 2015, Schmidhuber, 2015, Krizhevsky et al., 2012].

Our main aim is to collect results on differentiating parameterized argmin and argmax problems. Some of these results have appeared in one form or another in earlier works, e.g., Faugeras [1993, §5.6.2–5.6.3] considers the case of unconstrained and equality constrained argmin problems when analysing uncertainty in recovering 3D geometry. However, with the growth in popularity of deep learning we feel it important to revisit the results (and present examples) in the context of first-order gradient procedures for solving bi-level optimization problems.

We begin with a brief overview of methods for solving bi-level optimization problems to motivate our results in Section 2. We then consider unconstrained variants of the lower-level problem, either argmin or argmax (in Section 3), and then extend the results to problems with equality and inequality constraints (in Section 4). We include motivating examples with gradient calculations and discussion throughout the paper leading to a small bi-level optimization example for learning a novel softmax classifier in Section 5. Examples are accompanied by supplementary Python code.[1]

2 Background

The canonical form of a bi-level optimization problem is shown in Equation 1. There are three general approaches to solving such problems that have been proposed over the years. In the first approach an analytic solution is found for the lower-level problem, that is, an explicit function $\boldsymbol{y}^\star(\boldsymbol{x})$ that when evaluated returns an element of $\operatorname{argmin}_{\boldsymbol{y}} f^L(\boldsymbol{x}, \boldsymbol{y})$. If such a function can be found then we are in luck because we now simply solve the single-level problem

$$\operatorname{minimize}_{\boldsymbol{x}} \quad f^U(\boldsymbol{x}, \boldsymbol{y}^\star(\boldsymbol{x})) \tag{2}$$

Of course this problem, may itself be difficult to solve. Moreover, it is not always the case that an analytic solution for the lower-level problem will exist.

The second general approach to solving bi-level optimization problems is to replace the lower-level problem with a set of sufficient conditions for optimiality (e.g., the KKT conditions for a convex lower-level problem). If we think of these conditions being encapsulated by the function $h^L(\boldsymbol{x}, \boldsymbol{y}) = 0$ then we can solve the following constrained problem instead of the original,

$$\begin{array}{ll} \operatorname{minimize}_{\boldsymbol{x},\boldsymbol{y}} & f^U(\boldsymbol{x}, \boldsymbol{y}) \\ \text{subject to} & h^L(\boldsymbol{x}, \boldsymbol{y}) = 0 \end{array} \tag{3}$$

The main difficulty here is that the sufficient conditions may be hard to express and the resulting problem hard to solve. Indeed, even if the lower-level problem is convex, the resulting constrained problem may not be.

[1] Available for download from `http://users.cecs.anu.edu.au/sgould/`.

The third general approach to solving bi-level optimization problems is via gradient descent on the upper-level objective. The key idea is to compute the gradient of the solution to the lower-level problem with respect to the variables in the upper-level problem and perform updates of the form

$$x \leftarrow x - \eta \left(\frac{\partial f^U}{\partial x} + \frac{\partial f^U}{\partial y} \frac{\partial y}{\partial x} \right) \Bigg|_{(x, y^\star)} \tag{4}$$

In this respect the approach appears similar to the first approach. However, now the function $\boldsymbol{y}^\star(\boldsymbol{x})$ does not need to be found explicitly. All that we require is that the lower-level problem be efficiently solveable and that a method exists for finding the gradient at the current solution.[2]

This last approach is important in the context large-scale and end-to-end machine learning applications where first-order (stochastic) gradient methods are often the preferred method. This then motivates the results included in this technical report, i.e., computing the gradients of parameterized argmin and argmax optimization problems where the parameters are to be optimized for some external objective or are themselves the output of some other parameterized function to be learned.

3 Unconstrained Optimization Problems

We begin by considering the gradient of the scalar function $g(x) = \operatorname{argmin}_{y \in \mathbb{R}} f(x, y)$ and follow an approach similar to the implicit differentiation derivation that appears in earlier works (e.g., [Samuel and Tappen, 2009, Faugeras, 1993]). In all of our results we assume that the minimum (or maximum) over y of the function $f(x, y)$ exists over the domain of x. When the minimum (maximum) is not unique then $g(x)$ can be taken as any one of the minimum (maximum) points. Moreover, we do not need for $g(x)$ to have a closed-form representation.

Lemma 3.1.: Let $f : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$ be a continuous function with first and second derivatives. Let $g(x) = \operatorname{argmin}_y f(x, y)$. Then the derivative of g with respect to x is

$$\frac{dg(x)}{dx} = -\frac{f_{XY}(x, g(x))}{f_{YY}(x, g(x))}$$

where $f_{XY} \doteq \frac{\partial^2 f}{\partial x \partial y}$ and $f_{YY} \doteq \frac{\partial^2 f}{\partial y^2}$.

Proof.

$$\frac{\partial f(x, y)}{\partial y} \Bigg|_{y = g(x)} = 0 \qquad \text{(since } g(x) = \operatorname*{argmin}_y f(x, y)\text{)} \tag{5}$$

$$\therefore \frac{d}{dx} \frac{\partial f(x, g(x))}{\partial y} = 0 \qquad \text{(differentiating lhs and rhs)} \tag{6}$$

But, by the chain rule,

$$\frac{d}{dx} \left(\frac{\partial f(x, g(x))}{\partial y} \right) = \frac{\partial^2 f(x, g(x))}{\partial x \partial y} + \frac{\partial^2 f(x, g(x))}{\partial y^2} \frac{dg(x)}{dx} \tag{7}$$

[2] Note that we have made no assumption about the uniqueness of $y^\star$ nor the convexity of f^L or f^U. When multiple minima of f^L exist care needs to be taken during iterative gradient updates to select consistent solutions or when jumping between modes. However, these considerations are application dependent and do not invalidate any of the results included in this report.

Equating to zero and rearranging gives the desired result

$$\frac{dg(x)}{dx} = -\left(\frac{\partial^2 f(x, g(x))}{\partial y^2}\right)^{-1} \frac{\partial^2 f(x, g(x))}{\partial x \partial y} \tag{8}$$

$$= -\frac{f_{XY}(x, g(x))}{f_{YY}(x, g(x))} \tag{9}$$

□

We now extend the above result to the case of optimizing over vector-valued arguments.

Lemma 3.2.: Let $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$ be a continuous function with first and second derivatives. Let $\boldsymbol{g}(x) = \text{argmin}_{\boldsymbol{y} \in \mathbb{R}^n} f(x, \boldsymbol{y})$. Then the vector derivative of $\boldsymbol{g}$ with respect to x is

$$\boldsymbol{g}'(x) = -f_{YY}(x, \boldsymbol{g}(x))^{-1} f_{XY}(x, \boldsymbol{g}(x)).$$

where $f_{YY} \doteq \nabla^2_{\boldsymbol{yy}} f(x, \boldsymbol{y}) \in \mathbb{R}^{n \times n}$ and $f_{XY} \doteq \frac{\partial}{\partial x} \nabla_{\boldsymbol{y}} f(x, \boldsymbol{y}) \in \mathbb{R}^n$.

Proof. Similar to Lemma 3.1, we have:

$$f_Y(x, \boldsymbol{g}(x)) \doteq \nabla_Y f(x, \boldsymbol{y})|_{\boldsymbol{y}=\boldsymbol{g}(x)} = 0 \tag{10}$$

$$\frac{d}{dx} f_Y(x, \boldsymbol{g}(x)) = 0 \tag{11}$$

$$\therefore f_{XY}(x, \boldsymbol{g}(x)) + f_{YY}(x, \boldsymbol{g}(x)) \boldsymbol{g}'(x) = 0 \tag{12}$$

$$\frac{d}{dx} \boldsymbol{g}(x) = -f_{YY}(x, \boldsymbol{g}(x))^{-1} f_{XY}(x, \boldsymbol{g}(x)) \tag{13}$$

□

The above lemma assumes that $\boldsymbol{g}(x)$ is parameterized by a single scalar value $x \in \mathbb{R}$. It is trivial to extend the result to multiple parameters $\boldsymbol{x} = (x_1, \ldots, x_m)$ by performing the derivative calculation for each parameter separately. That is,

$$\nabla_{\boldsymbol{x}} \boldsymbol{g}(x_1, \ldots, x_m) = -f_{YY}(\boldsymbol{x}, \boldsymbol{g}(\boldsymbol{x}))^{-1} \begin{bmatrix} f_{X_1 Y}(\boldsymbol{x}, \boldsymbol{g}(\boldsymbol{x})) & \cdots & f_{X_m Y}(\boldsymbol{x}, \boldsymbol{g}(\boldsymbol{x})) \end{bmatrix} \tag{14}$$

Note that the main computational challenge of the inverting the $n \times n$ matrix f_{YY} (or decomposing it to facilitate solving each system of n linear equations) only needs to be done once and can then be reused for the derivative with respect to each parameter. Thus the overhead of computing gradients for multiple parameters is small compared to the cost of computing the gradient for just one parameter. Of course if $\boldsymbol{x}$ (or $\boldsymbol{g}(\boldsymbol{x})$) is changed (e.g., during bi-level optimization) then f_{YY} will be different and its inverse recalculated.

So far we have only considered minimization problems. However, studying the proofs above we see that they do not require that $\boldsymbol{g}(x)$ be a local-minimum point; any stationary point will suffice. Thus, the result extends to the case of argmax problems as well.

Lemma 3.3.: Let $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$ be a continuous function with first and second derivatives. Let $\boldsymbol{g}(x) = \text{argmax}_{\boldsymbol{y} \in \mathbb{R}^n} f(x, \boldsymbol{y})$. Then the vector derivative of $\boldsymbol{g}$ with respect to x is

$$\boldsymbol{g}'(x) = -f_{YY}(x, \boldsymbol{g}(x))^{-1} f_{XY}(x, \boldsymbol{g}(x)).$$

where $f_{YY} \doteq \nabla^2_{\boldsymbol{yy}} f(x, \boldsymbol{y}) \in \mathbb{R}^{n \times n}$ and $f_{XY} \doteq \frac{\partial}{\partial x} \nabla_{\boldsymbol{y}} f(x, \boldsymbol{y}) \in \mathbb{R}^n$.

Proof. Follows from proof of Lemma 3.2. □

3.1 Example: Scalar Mean

In this section we consider the simple example of finding the point whose sum-of-squared distance to all points in the set $\{h_i(x)\}_{i=1}^m$ is minimized. Writing out the problem as a mathematical optimization problem we have $g(x) = \text{argmin}_y f(x, y)$ where $f(x, y) = \sum_{i=1}^m (h_i(x) - y)^2$. Here a well-known analytic solution exists, namely the mean $g(x) = \frac{1}{m} \sum_{i=1}^m h_i(x)$.

Applying Lemma 3.1 we have:

$$f_Y(x, y) = -2 \sum_{i=1}^m (h_i(x) - y) \tag{15}$$

$$f_{XY}(x, y) = -2 \sum_{i=1}^m h_i'(x) \tag{16}$$

$$f_{YY}(x, y) = 2m \tag{17}$$

$$\therefore g'(x) = \frac{1}{m} \sum_{i=1}^m h_i'(x) \tag{18}$$

which agrees with the analytical solution (assuming the derivatives $h_i'(x)$ exist).

3.2 Example: Scalar Function with Three Local Minima

The results above do not require that the function f being optimized have a single global optimal point. Indeed, as discussed above, the results hold for any stationary point (local optima or inflection point). In this example we present a function with up to three stationary points and show that we can calculate the gradient with respect to x at each stationary point.[3] Consider the function

$$f(x, y) = xy^4 + 2x^2y^3 - 12y^2 \tag{19}$$

with the following partial first- and second-order derivatives

$$f_Y(x, y) = 4xy^3 + 6x^2y^2 - 24y \tag{20}$$

$$f_{XY}(x, y) = 4y^3 + 12xy^2 \tag{21}$$

$$f_{YY}(x, y) = 12xy^2 + 12x^2y - 24 \tag{22}$$

Then for any stationary point $g(x)$ of $f(x, y)$, where stationarity is with respect to y for fixed x, we have

$$g'(x) = -\frac{g(x)^3 + 3xg(x)^2}{3xg(x)^2 + 3x^2g(x) - 6} \tag{23}$$

Here the gradient describes how each stationary point $g(x)$ moves locally with an infintisimal change in x. If such a problem, with multiple local minima, were to be used within a bi-level optimization learning problem then care should be taken during each iteration to select corresponding solution points at each iteration of the algorithm.

[3] Technically the function that we present only has three stationary points when $x < -2\left(\frac{4}{3}\right)^{\frac{1}{3}}$ or $x > 0$. It has two stationary points at $x = -2\left(\frac{4}{3}\right)^{\frac{1}{3}}$ and a unique stationary point elsewhere.

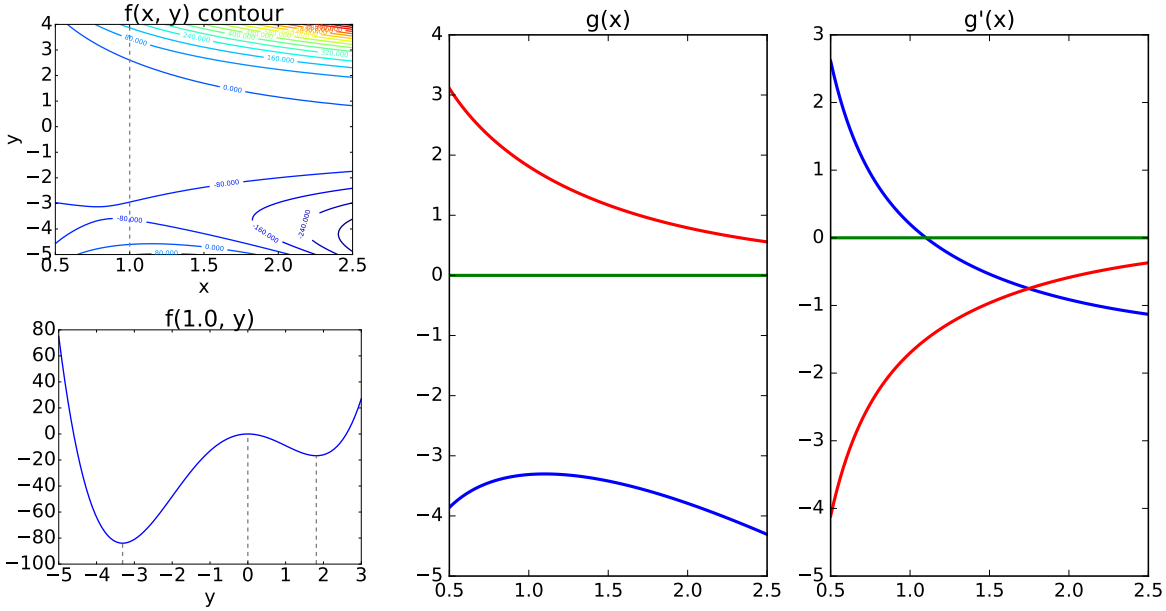


Figure 1: Example of a parameterized scalar function $f(x, y) = xy^4 + 2x^2y^3 - 12y^2$ with three stationary points for any fixed $x > 0$. The top-left panel shows a contour plot of f; the bottom-left panel shows the function at $x = 1$; and the remaining panels show the three solutions for $g(x) = \text{argmin}_y f(x, y)$ and corresponding gradients $g'(x)$ at each stationary point.

The (three) stationary points occur when $\frac{\partial f(x,y)}{\partial y} = 0$, which (in this example) we can compute analytically as

$$g(x) \in \underset{y}{\text{argmin}}\, f(x, y) = \left\{ 0, \frac{-3x^2 \pm \sqrt{9x^4 + 96x}}{4x} \right\} \tag{24}$$

which we use when generating the plots below.

Figure 1 shows a surface plot of $f(x, y)$ and a slice through the surface at $x = 1$. Clearly visible in the plot are the three stationary points. The figure also shows the three values for $g(x)$ and their gradients $g'(x)$ for a range of x. Note that one of the solutions (i.e., $y = 0$) is independent of x.

3.3 Example: Maximum Likelihood of Soft-max Classifier

Now consider the more elaborate example of exploring how the maximum likelihood feature vector of a soft-max classifier changes as a function of the classifier's parameters. Assume m classes and let classifier be parameterized by $\Theta = \{(\boldsymbol{a}_i, b_i)\}_{i=1}^m$. Then we can define the likelihood of feature vector $\boldsymbol{x}$ for the i-th class of a soft-max distribution as

$$\ell_i(\boldsymbol{x}) = P(Y = i \mid \boldsymbol{X} = \boldsymbol{x}; \Theta) \tag{25}$$

$$= \frac{1}{Z(\boldsymbol{x}; \Theta)} \exp\left(\boldsymbol{a}_i^T \boldsymbol{x} + b_i\right) \tag{26}$$

where $Z(\boldsymbol{x}; \Theta) = \sum_{j=1}^m \exp\left(\boldsymbol{a}_j^T \boldsymbol{x} + b_j\right)$ is the partition function.[4]

[4]Note that we use a different notation here to be consistent with the standard notation in machine learning. In particular, the role of $\boldsymbol{x}$ is differs from its appearance elsewhere in this article.

The maximum (log-)likelihood feature vector for class i can be found as

$$\boldsymbol{g}_i(\Theta) = \operatorname*{argmax}_{\boldsymbol{x}\in\mathbb{R}^n} \log \ell_i(\boldsymbol{x}) \tag{27}$$

$$= \operatorname*{argmax}_{\boldsymbol{x}\in\mathbb{R}^n} \left\{ \boldsymbol{a}_i^T \boldsymbol{x} + b_i - \log \left(\sum_{j=1}^{m} \exp\left(\boldsymbol{a}_j^T \boldsymbol{x} + b_j\right) \right) \right\} \tag{28}$$

whose objective is concave and so has a unique global maximum. However, the problem, in general, has no closed-form solution. Yet we can still compute the derivative of the maximum likelihood feature vector $\boldsymbol{g}_i(\Theta)$ with respect to any of the model parameters as follows.

$$\nabla_{\boldsymbol{x}} \log \ell_i(\boldsymbol{x}) = \boldsymbol{a}_i - \sum_{j=1}^{m} \ell_j(\boldsymbol{x}) \boldsymbol{a}_j \tag{29}$$

$$\nabla^2_{\boldsymbol{x}\boldsymbol{x}} \log \ell_i(\boldsymbol{x}) = \sum_{j=1}^{m}\sum_{k=1}^{m} \ell_j(\boldsymbol{x})\ell_k(\boldsymbol{x}) \boldsymbol{a}_j \boldsymbol{a}_k^T - \sum_{j=1}^{m} \ell_j(\boldsymbol{x}) \boldsymbol{a}_j \boldsymbol{a}_j^T \tag{30}$$

$$\frac{\partial}{\partial a_{jk}} \nabla_{\boldsymbol{x}} \log \ell_i(\boldsymbol{x}) = [\![i = j]\!] \boldsymbol{e}_k - \ell_j(\boldsymbol{x}) x_k \left(\boldsymbol{a}_j - \sum_{l=1}^{m} \ell_l(\boldsymbol{x}) \boldsymbol{a}_l \right) - \ell_j(\boldsymbol{x}) \boldsymbol{e}_k \tag{31}$$

$$\frac{\partial}{\partial b_j} \nabla_{\boldsymbol{x}} \log \ell_i(\boldsymbol{x}) = -\ell_j(\boldsymbol{x}) \left(\boldsymbol{a}_j - \sum_{k=1}^{m} \ell_k(\boldsymbol{x}) \boldsymbol{a}_k \right) \tag{32}$$

where $\boldsymbol{e}_k$ is the k-th canonical vector (k-th element one and the rest zero) and $[\![\cdot]\!]$ is the indicator function (or iverson bracket), which takes value 1 when its argument is true and 0 otherwise.

Letting $\boldsymbol{x}^\star = \boldsymbol{g}_i(\Theta)$, $H = \nabla^2_{\boldsymbol{x}\boldsymbol{x}} \log \ell_i(\boldsymbol{x}^\star)$ and $\bar{\boldsymbol{a}} = \sum_{k=1}^{m} \ell_k(\boldsymbol{x}^\star)\boldsymbol{a}_k$, we have

$$\frac{\partial \boldsymbol{g}_i}{\partial a_{jk}} = \begin{cases} H^{-1}(\ell_i(\boldsymbol{x}^\star) - 1)\boldsymbol{e}_k, & i = j \\ \ell_j(\boldsymbol{x}^\star) H^{-1} \left(x_k^\star (\boldsymbol{a}_j - \bar{\boldsymbol{a}}) + \boldsymbol{e}_k \right), & i \neq j \end{cases} \tag{33}$$

$$\frac{\partial \boldsymbol{g}_i}{\partial b_j} = \begin{cases} 0, & i = j \\ \ell_j(\boldsymbol{x}^\star) H^{-1} \left(\boldsymbol{a}_j - \bar{\boldsymbol{a}} \right), & i \neq j \end{cases} \tag{34}$$

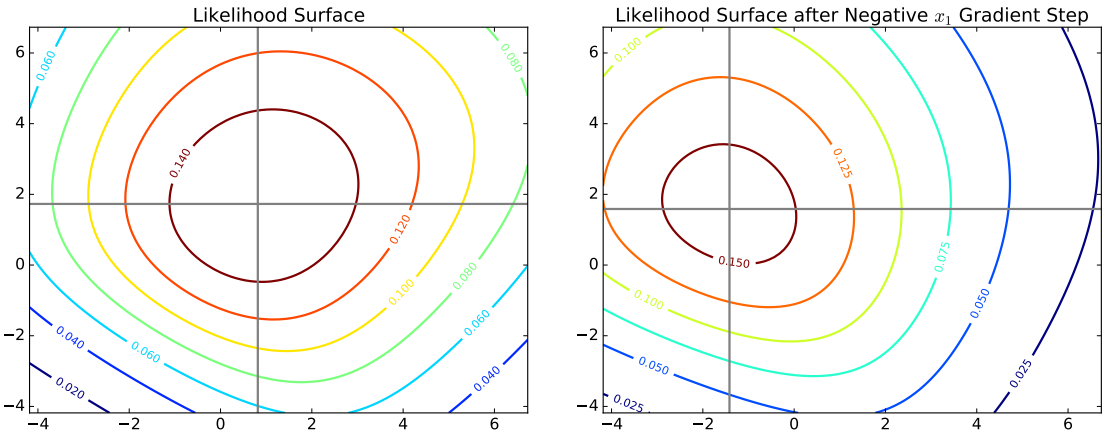
where we have used the fact that $\boldsymbol{a}_i = \bar{\boldsymbol{a}}$ since $\nabla_{\boldsymbol{x}} \log \ell_i(\boldsymbol{x}^\star) = 0$.

Figure 2 shows example maximum-likelihood surfaces for one of the likelihood functions $\ell_i(\boldsymbol{x})$ from a soft-max distribution over ten classes and two-dimensional feature vectors. The parameters of the distribution were chosen randomly. Shown are the initial surface and the surface after adjusting all parameters in the negative gradient direction for the first feature dimension, i.e., $\theta \leftarrow \theta - \eta \boldsymbol{e}_1^T \frac{\partial \boldsymbol{g}_i}{\partial \theta}$ where we have used θ to stand for any of the parameters in $\{a_{ij}, b_i \mid i = 1, \ldots, 10; j = 1, 2\}$.

This example will be developed further below by adding equality and inequality constraints and then applying it in the context of bi-level optimization.

3.4 Invariance Under Monotonic Transformations

This section explores the idea that the optimal point of a function is invariant under certain transformations, e.g., exponentiation. Let $g(x) = \operatorname{argmin}_y f(x, y)$. Now let $\tilde{f}(x, y) = e^{f(x,y)}$ and $\tilde{g}(x) = \operatorname{argmin}_y \tilde{f}(x, y)$. Clearly, $\tilde{g}(x) = g(x)$ (since the exponential function is smooth and monotonically increasing).



(a) $\boldsymbol{x}^\star = (0.817, 1.73)$ (b) $\boldsymbol{x}^\star = (-1.42, 1.59)$

Figure 2: Example maximum-likelihood surfaces $\ell_i(\boldsymbol{x})$ before and after taking a small step on all parameters in the negative gradient direction for $x_1^\star$.

Computing the gradients we see that

$$\tilde{f}_Y(x,y) = e^{f(x,y)} f_Y(x,y) \tag{35}$$

$$\tilde{f}_{XY}(x,y) = e^{f(x,y)} f_{XY}(x,y) + e^{f(x,y)} f_X(x,y) f_Y(x,y) \tag{36}$$

$$\tilde{f}_{YY}(x,y) = e^{f(x,y)} f_{YY}(x,y) + e^{f(x,y)} f_Y^2(x,y) \tag{37}$$

So applying Lemma 3.1, we have

$$\tilde{g}'(x) = -\frac{\tilde{f}_{XY}(x,g(x))}{\tilde{f}_{YY}(x,g(x))} \tag{38}$$

$$= -\frac{e^{f(x,g(x))} f_{XY}(x,g(x)) + e^{f(x,g(x))} f_X(x,g(x)) f_Y(x,g(x))}{e^{f(x,g(x))} f_{YY}(x,g(x)) + e^{f(x,g(x))} f_Y^2(x,g(x))} \tag{39}$$

$$= -\frac{f_{XY}(x,g(x)) + f_X(x,g(x)) f_Y(x,g(x))}{f_{YY}(x,g(x)) + f_Y^2(x,g(x))} \qquad \text{(cancelling } e^{f(x,g(x))}\text{)} \tag{40}$$

$$= -\frac{f_{XY}(x,g(x))}{f_{YY}(x,g(x))} \qquad \text{(since } f_Y(x,g(x)) = 0\text{)} \tag{41}$$

$$= g'(x) \tag{42}$$

Similarly, now assume $f(x,y) > 0$ for all x and y, and let $\tilde{f}(x,y) = \log f(x,y)$ and $\tilde{g}(x) = \operatorname{argmin}_y \tilde{f}(x,y)$. Again we have $\tilde{g}(x) = g(x)$ (since the logarithmic function is smooth and monotonically increasing on the positive reals).

Verifying Lemma 3.1, we have

$$\tilde{f}_Y(x,y) = \frac{1}{f(x,y)} f_Y(x,y) \tag{43}$$

$$\tilde{f}_{XY}(x,y) = \frac{f(x,y) f_{XY}(x,y) - f_Y(x,y) f_X(x,y)}{f^2(x,y)} \tag{44}$$

$$\tilde{f}_{YY}(x,y) = \frac{f(x,y) f_{YY}(x,y) - f_Y^2(x,y)}{f^2(x,y)} \tag{45}$$

and once again we can set $f_Y(x, g(x))$ to zero and cancel terms to get $\tilde{g}'(x) = g'(x)$.

These examples motivate the following lemma, which formalizes the fact that composing a function with a monotonically increasing or monotonically decreasing function does not change its stationary points.

Lemma 3.4.: Let $f : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ be a continuous function with first and second derivatives. Let $h : \mathbb{R} \to \mathbb{R}$ be a smooth monotonically increasing or monotoncially decreasing function and let $g(x) = \text{argmin}_y h(f(x, y))$. Then the derivative of g with respect to x is

$$\frac{dg(x)}{dx} = -\frac{f_{XY}(x, g(x))}{f_{YY}(x, g(x))}$$

where $f_{XY} \doteq \frac{\partial^2 f}{\partial x \partial y}$ and $f_{YY} \doteq \frac{\partial^2 f}{\partial y^2}$.

Proof. Follows from Lemma 3.1 observing that $\frac{\partial h(f(x,y))}{\partial y} = h'(f(x, y))\frac{\partial f(x,y)}{\partial y}$ by the chain rule and that $h'(f(x, y))$ is always non-zero for monotonically increasing or decreasing h. □

4 Constrained Optimization Problems

In this section we extend the results of the argmin and argmax derivatives to problems with linear equality and arbitrary inequality constraints.

4.1 Equality Constraints

Let us introduce linear equality constraints $A\boldsymbol{y} = \boldsymbol{b}$ into the vector version of our minimization problem. We now have $\boldsymbol{g}(x) = \text{argmin}_{\boldsymbol{y}:A\boldsymbol{y}=\boldsymbol{b}} f(x, \boldsymbol{y})$ and wish to find $\boldsymbol{g}'(x)$.

Lemma 4.1.: Let $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$ be a continuous function with first and second derivatives. Let $A \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$. Let $\boldsymbol{y}_0 \in \mathbb{R}^n$ be any vector satisfying $A\boldsymbol{y}_0 = \boldsymbol{b}$ and let the columns of F span the null-space of A. Let $\boldsymbol{z}^\star(x) \in \text{argmin}_{\boldsymbol{z}} f(x, \boldsymbol{y}_0 + F\boldsymbol{z})$ so that $\boldsymbol{g}(x) = \boldsymbol{y}_0 + F\boldsymbol{z}^\star(x)$. Then

$$\boldsymbol{g}'(x) = -F\left(F^T f_{YY}(x, \boldsymbol{g}(x))F\right)^{-1} F^T f_{XY}(x, \boldsymbol{g}(x))$$

where $f_{YY} \doteq \nabla^2_{\boldsymbol{y}\boldsymbol{y}} f(x, \boldsymbol{y}) \in \mathbb{R}^{n \times n}$ and $f_{XY} \doteq \frac{\partial}{\partial x}\nabla_{\boldsymbol{y}} f(x, \boldsymbol{y}) \in \mathbb{R}^n$.

Proof.

$$\nabla_{\boldsymbol{z}} f(x, \boldsymbol{y}_0 + F\boldsymbol{z})|_{\boldsymbol{z}=\boldsymbol{z}^\star(x)} = 0 \tag{46}$$

$$F^T f_Y(x, \boldsymbol{g}(x)) = 0 \tag{47}$$

$$F^T f_{XY}(x, \boldsymbol{g}(x)) + F^T f_{YY}(x, \boldsymbol{g}(x))F\boldsymbol{z}'(x) = 0 \tag{48}$$

$$\therefore \boldsymbol{z}'(x) = -(F^T f_{YY}(x, \boldsymbol{g}(x))F)^{-1}F^T f_{XY}(x, \boldsymbol{g}(x)) \tag{49}$$

$$\therefore \boldsymbol{g}'(x) = -F(F^T f_{YY}(x, \boldsymbol{g}(x))F)^{-1}F^T f_{XY}(x, \boldsymbol{g}(x)) \tag{50}$$

□

Alternatively, we can construct the $\boldsymbol{g}'(x)$ directly as the following lemma shows.

Lemma 4.2.: Let $f : \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}$ be a continuous function with first and second derivatives. Let $A \in \mathbb{R}^{m \times n}$ and $\boldsymbol{b} \in \mathbb{R}^m$ with $\text{rank}(A) = m$. Let $\boldsymbol{g}(x) = \text{argmin}_{\boldsymbol{y}: A\boldsymbol{y}=\boldsymbol{b}} f(x, \boldsymbol{y})$. Let $H = f_{YY}(x, \boldsymbol{g}(x))$. Then

$$\boldsymbol{g}'(x) = \left(H^{-1}A^T \left(AH^{-1}A^T\right)^{-1} AH^{-1} - H^{-1}\right) f_{XY}(x, \boldsymbol{g}(x))$$

where $f_{YY} \doteq \nabla^2_{\boldsymbol{yy}} f(x, \boldsymbol{y}) \in \mathbb{R}^{n \times n}$ and $f_{XY} \doteq \frac{\partial}{\partial x} \nabla_{\boldsymbol{y}} f(x, \boldsymbol{y}) \in \mathbb{R}^n$.

Proof. Consider the Lagrangian $\mathcal{L}$ for the constrained optimization problem,

$$\begin{array}{ll} \underset{\boldsymbol{y} \in \mathbb{R}^n}{\text{minimize}} & f(x, \boldsymbol{y}) \\ \text{subject to} & A\boldsymbol{y} = \boldsymbol{b} \end{array} \tag{51}$$

We have

$$\mathcal{L}(x, \boldsymbol{y}, \boldsymbol{\lambda}) = f(x, \boldsymbol{y}) + \boldsymbol{\lambda}^T (A\boldsymbol{y} - \boldsymbol{b}) \tag{52}$$

Assume $\tilde{\boldsymbol{g}}(x) = (\boldsymbol{y}^\star(x), \boldsymbol{\lambda}^\star(x))$ is a optimal primal-dual pair. Then

$$\begin{bmatrix} \nabla_{\boldsymbol{y}} \mathcal{L}(x, \boldsymbol{y}^\star, \boldsymbol{\lambda}^\star) \\ \nabla_{\boldsymbol{\lambda}} \mathcal{L}(x, \boldsymbol{y}^\star, \boldsymbol{\lambda}^\star) \end{bmatrix} = \begin{bmatrix} f_Y(x, \boldsymbol{y}^\star) + A^T \boldsymbol{\lambda}^\star \\ A\boldsymbol{y}^\star - \boldsymbol{b} \end{bmatrix} = 0 \tag{53}$$

$$\frac{\mathrm{d}}{\mathrm{d}x} \begin{bmatrix} f_Y(x, \boldsymbol{y}^\star) + A^T \boldsymbol{\lambda}^\star \\ A\boldsymbol{y}^\star - \boldsymbol{b} \end{bmatrix} = 0 \tag{54}$$

$$\begin{bmatrix} f_{XY}(x, \boldsymbol{y}^\star) \\ 0 \end{bmatrix} + \begin{bmatrix} f_{YY}(x, \boldsymbol{y}^\star) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{g}}_Y(x) \\ \tilde{\boldsymbol{g}}_\Lambda(x) \end{bmatrix} = 0 \tag{55}$$

$$\therefore \begin{bmatrix} f_{YY}(x, \boldsymbol{y}^\star) & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} \tilde{\boldsymbol{g}}_Y(x) \\ \tilde{\boldsymbol{g}}_\Lambda(x) \end{bmatrix} = \begin{bmatrix} -f_{XY}(x, \boldsymbol{y}^\star) \\ 0 \end{bmatrix} \tag{56}$$

From the first row of the block matrix equation above, we have

$$\tilde{\boldsymbol{g}}_Y(x) = H^{-1} \left(-f_{XY}(x, \boldsymbol{y}^\star) - A^T \tilde{\boldsymbol{g}}_\Lambda(x)\right) \tag{57}$$

where $H = f_{YY}(x, \boldsymbol{y}^\star)$.

Substituting into the second row gives

$$AH^{-1} \left(f_{XY}(x, \boldsymbol{y}^\star) + A^T \tilde{\boldsymbol{g}}_\Lambda(x)\right) = 0 \tag{58}$$

$$\therefore \ \tilde{\boldsymbol{g}}_\Lambda(x) = -\left(AH^{-1}A^T\right)^{-1} AH^{-1} f_{XY}(x, \boldsymbol{y}^\star) \tag{59}$$

which when substituted back into the first row gives the desired result with $\boldsymbol{g}(x) = \boldsymbol{y}^\star$ and $\boldsymbol{g}'(x) = \tilde{\boldsymbol{g}}_Y(x)$. □

Note that for $A = 0$ (and $b = 0$) this reduces to the result from Lemma 3.2. Furthermore, $\boldsymbol{g}'(x)$ is in the null-space of A, which we require if the linear equality constraint is to remain satisfied.

4.2 Inequality Constraints

Consider a family of optimization problems, indexed by x, with inequality constraints. In standard form we have

$$\begin{array}{ll} \text{minimize}_{\boldsymbol{y} \in \mathbb{R}} & f_0(x, \boldsymbol{y}) \\ \text{subject to} & f_i(x, \boldsymbol{y}) \leq 0 \quad i = 1, \ldots, m \end{array} \tag{60}$$

where $f_0(x, \boldsymbol{y})$ is the objective function and $f_i(x, \boldsymbol{y})$ are the inequality constraint functions.

Let $\boldsymbol{g}(x) \in \mathbb{R}^n$ be an optimal solution. We wish to find $\boldsymbol{g}'(x)$, which we approximate using ideas from interior-point methods [Boyd and Vandenberghe, 2004]. Introducing the log-barrier function $\phi(x, \boldsymbol{y}) = \sum_{i=1}^m \log(-f_i(x, \boldsymbol{y}))$, we can approximate the above problem as

$$\text{minimize}_{\boldsymbol{y}} \quad tf_0(x, \boldsymbol{y}) - \sum_{i=1}^m \log(-f_i(x, \boldsymbol{y})) \tag{61}$$

where $t > 0$ is a scaling factor that controls the approximation (or duality gap when the problem is convex). We now have an unconstrained problem and can invoke Lemma 3.2.

For completeness, recall that the gradient and Hessian of the log-barrier function, $\phi(\boldsymbol{z}) = \sum_{i=1}^m \log(-f_i(\boldsymbol{z}))$, are given by

$$\nabla\phi(\boldsymbol{z}) = -\sum_{i=1}^m \frac{1}{f_i(\boldsymbol{z})} \nabla f_i(\boldsymbol{z}), \tag{62}$$

$$\nabla^2\phi(\boldsymbol{z}) = \sum_{i=1}^m \frac{1}{f_i(\boldsymbol{z})^2} \nabla f_i(\boldsymbol{z}) \nabla f_i(\boldsymbol{z})^T - \sum_{i=1}^m \frac{1}{f_i(\boldsymbol{z})} \nabla^2 f_i(\boldsymbol{z}) \tag{63}$$

Thus the gradient of an inequality constrained argmin function can be approximated as

$$\boldsymbol{g}'(x) \approx -\Big(tf_{YY}(x, \boldsymbol{g}(x)) - \phi_{YY}(x, \boldsymbol{g}(x))\Big)^{-1} \Big(tf_{XY}(x, \boldsymbol{g}(x)) - \phi_{XY}(x, \boldsymbol{g}(x))\Big) \tag{64}$$

In many cases the constraint functions f_i will not depend on x and the above expression can be simplified by setting $\phi_{XY}(x, \boldsymbol{y})$ to zero.

4.3 Example: Positivity Constraints

We consider an inequality constrained version of our scalar mean example from above,

$$\begin{array}{rll} g(x) = & \text{argmin}_{y \in \mathbb{R}} & \sum_{i=1}^m (h_i(x) - y)^2 \\ & \text{subject to} & y \geq 0 \end{array} \tag{65}$$

where we have added a positivity constraint on y. This problem has closed-form solution

$$g(x) = \max\left\{0,\ \frac{1}{m}\sum_{i=1}^m h_i(x)\right\} \tag{66}$$

Following Section 4.2 we can construct the approximation

$$g_t(x) = \operatorname*{argmin}_{y \in \mathbb{R}} tf(x, y) - \log(y) \tag{67}$$

where $f(x, y) = \sum_{i=1}^m (h_i(x) - y)^2$. Applying Lemma 3.1 gives

$$\frac{\partial}{\partial y}\left(tf(x, y) - \phi(y)\right) = -2t\sum_{i=1}^m (h_i(x) - y) - \frac{1}{y} \tag{68}$$

$$\frac{\partial^2}{\partial x \partial y}\left(tf(x, y) - \phi(y)\right) = -2t\sum_{i=1}^m h_i'(x) \tag{69}$$

$$\frac{\partial^2}{\partial y^2}\left(tf(x, y) - \phi(y)\right) = 2tm + \frac{1}{y^2} \tag{70}$$

$$\therefore\ g_t'(x) = \frac{2t\sum_{i=1}^m h_i'(x)}{2tm + \frac{1}{g(x)^2}} \tag{71}$$

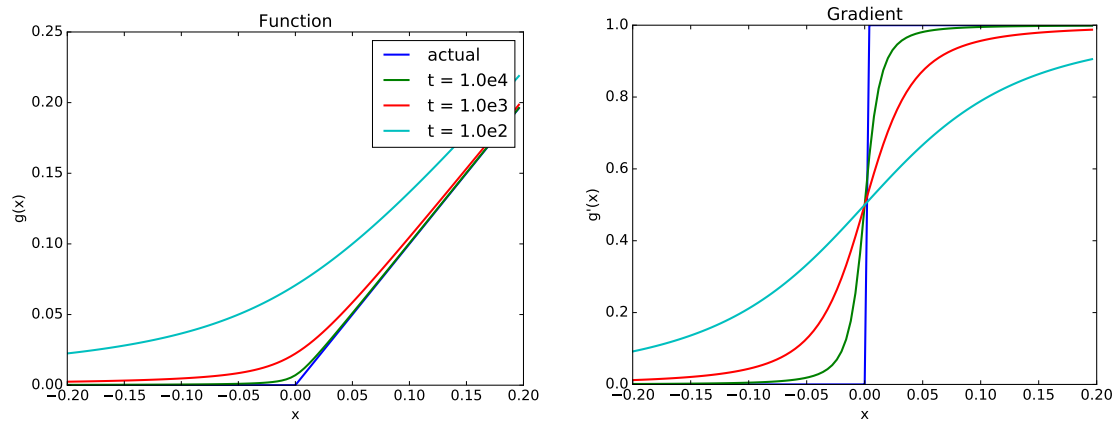


Figure 3: Example graph of both function value and gradient for $g(x) = \text{argmin}_{y \geq 0}(x-y)^2$ and approximations $g_t(x) = \text{argmin}_y\, t(x-y)^2 - \log(y)$ for different values of t. As $t \to \infty$ the approximation converges to the actual function and gradient.

Note that here we can solve the quadratic equation induced by $\frac{\partial}{\partial y}\left(tf(x,y) - \phi(y)\right) = 0$ to obtain a closed-form solution of $g_t(x)$ and hence $g_t'(x)$. We leave this as an exercise.

Observe from Equation 71 that as $t \to \infty$,

$$g_t'(x) = \begin{cases} \frac{1}{m}\sum_{i=1}^m h_i'(x) & \text{if } g(x) > 0 \\ 0 & \text{if } g(x) = 0 \end{cases} \tag{72}$$

We demonstrate this example on a special case with $m = 1$ and $h_1(x) = x$. The true and approximate function and their gradients are plotted in Figure 3.

4.4 Example: Maximum Likelihood of Constrained Soft-max Classifier

Let us now continue our soft-max example from Section 3.3 by adding constraints on the solution. We begin by adding the linear equality constraint $\mathbf{1}^T\boldsymbol{x} = 1$ to give

$$\begin{aligned} \boldsymbol{g}_i(\Theta) = \quad & \text{argmax}_{\boldsymbol{x}\in\mathbb{R}^n} \quad \left\{\boldsymbol{a}_i^T\boldsymbol{x} + b_i - \log\left(\sum_{j=1}^m \exp\left(\boldsymbol{a}_j^T\boldsymbol{x} + b_j\right)\right)\right\} \\ & \text{subject to} \quad \mathbf{1}^T\boldsymbol{x} = 1 \end{aligned} \tag{73}$$

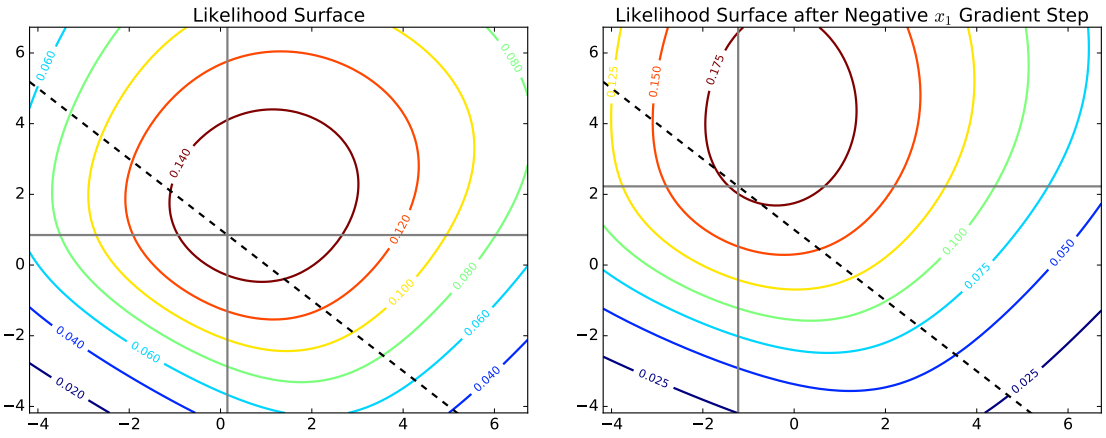
Following Lemma 4.2 and letting $H^\dagger = \left(H^{-1} - \frac{1}{\mathbf{1}^T H^{-1}\mathbf{1}} H^{-1}\mathbf{1}\mathbf{1}^T H^{-1}\right)$ we have the following gradients with respect to each parameter,

$$\frac{\partial \boldsymbol{g}_i}{\partial a_{jk}} = \begin{cases} H^\dagger(\ell_i(\boldsymbol{x}^\star) - 1)\boldsymbol{e}_k, & i = j \\ \ell_j(\boldsymbol{x}^\star)H^\dagger\left(x_k^\star(\boldsymbol{a}_j - \bar{\boldsymbol{a}}) + \boldsymbol{e}_k\right), & i \neq j \end{cases} \tag{74}$$

$$\frac{\partial \boldsymbol{g}_i}{\partial b_j} = \begin{cases} 0, & i = j \\ \ell_j(\boldsymbol{x}^\star)H^\dagger\left(\boldsymbol{a}_j - \bar{\boldsymbol{a}}\right), & i \neq j \end{cases} \tag{75}$$

where H and $\bar{\boldsymbol{a}}$ are as defined in Section 3.3.

Figure 4 shows the constrained solutions before and after taking a gradient step for the same maximum-likelihood surface as described in Section 3.3. Notice that the solution lies along the



(a) $\boldsymbol{x}^\star = (0.151, 0.849)$ (b) $\boldsymbol{x}^\star = (-1.23, 2, 23)$

Figure 4: Example maximum-likelihood surfaces $\ell_i(\boldsymbol{x})$ before and after taking a small step on all parameters in the negative gradient direction for $x_1^\star$ with constraint $\mathbf{1}^T\boldsymbol{x} = 1$.

line $x_1 + x_2 = 1$. Moreover, the sum of gradients for x_1 and x_2 are zero, i.e., the gradient is in the null-space of $\mathbf{1}^T$. To show this it is sufficient to prove that $H^\dagger\mathbf{1} = 0$, which is straightforward.

Next we consider adding the inequality constraint $\|\boldsymbol{x}\|^2 \leq 1$ to the soft-max maximum-likelihood problem. That is, we constrain the solution to lie within a unit ball centered at the origin. The new function is given by

$$\boldsymbol{g}_i(\Theta) = \underset{\boldsymbol{x}\in\mathbb{R}^n}{\operatorname{argmax}} \left\{ \boldsymbol{a}_i^T\boldsymbol{x} + b_i - \log\left(\sum_{j=1}^m \exp\left(\boldsymbol{a}_j^T\boldsymbol{x} + b_j\right)\right)\right\} \tag{76}$$

$$\text{subject to } \|\boldsymbol{x}\|^2 \leq 1 \tag{77}$$

Following Section 4.2 we can compute approximate gradients as

$$\frac{\partial \boldsymbol{g}_i}{\partial a_{jk}} \approx \begin{cases} (H + \nabla^2\phi(\boldsymbol{x}^\star))^{-1}(\ell_i(\boldsymbol{x}^\star) - 1)\boldsymbol{e}_k, & i = j \\ \ell_j(\boldsymbol{x}^\star)(H + \nabla^2\phi(\boldsymbol{x}^\star))^{-1}\left(x_k^\star(\boldsymbol{a}_j - \bar{\boldsymbol{a}}) + \boldsymbol{e}_k\right), & i \neq j \end{cases} \tag{78}$$

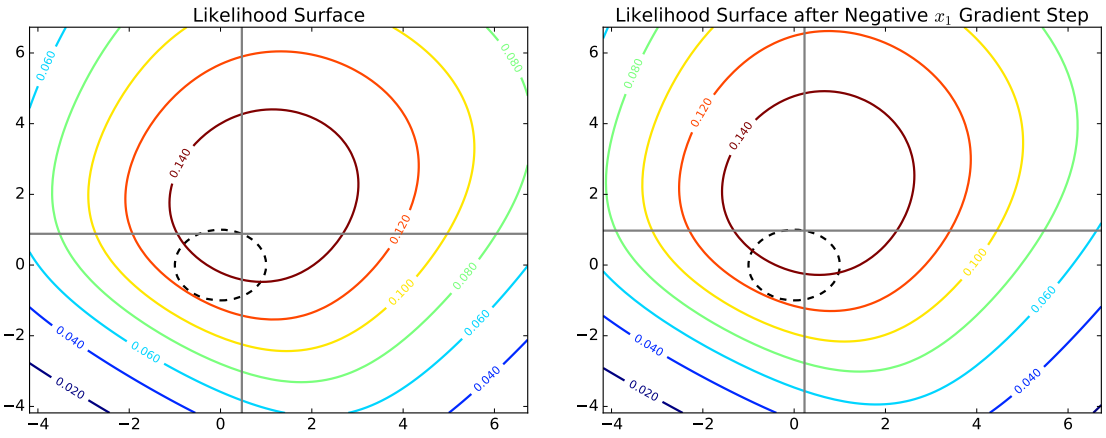
and

$$\frac{\partial \boldsymbol{g}_i}{\partial b_j} \approx \begin{cases} 0, & i = j \\ \ell_j(\boldsymbol{x}^\star)(H + \nabla^2\phi(\boldsymbol{x}^\star))^{-1}\left(\boldsymbol{a}_j - \bar{\boldsymbol{a}}\right), & i \neq j \end{cases} \tag{79}$$

where H and $\bar{\boldsymbol{a}}$ are as defined in Section 3.3, and the second derivative for the log-barrier $\phi(\boldsymbol{x}) = \log\left(1 - \|\boldsymbol{x}\|^2\right)$ is given by

$$\nabla^2\phi(\boldsymbol{x}) = \frac{4}{\left(1 - \|\boldsymbol{x}\|^2\right)^2}\boldsymbol{x}\boldsymbol{x}^T - \frac{2}{1 - \|\boldsymbol{x}\|^2}I_{n\times n} \tag{80}$$

Similar to previous examples, Figure 5 shows the maximum-likelihood surfaces and corresponding solutions before and after taking a gradient step (on all parameters) in the negative x_1 direction.



(a) $\boldsymbol{x}^\star = (0.467, 0.884)$ (b) $\boldsymbol{x}^\star = (0.227, 0.973)$

Figure 5: Example maximum-likelihood surfaces $\ell_i(\boldsymbol{x})$ before and after taking a small step on all parameters in the negative gradient direction for $x_1^\star$ with constraint $\|\boldsymbol{x}\|_2 \leq 1$.

5 Bi-level Optimization Example

We now build on previous examples to demonstrate the application of the above results to the problem of bi-level optimization. We consider the constrained maximum-likelihood problem from Section 4.4 with the goal of optimizing parameter to achieve a desired location for the maximum-likelihood feature vectors. We consider a three class problem over two dimensional space. Here we use the constrained version of the problem since for a three class soft-max classifier over two-dimensional space there will always be a direction in which the likelihood tends to one as the magnitude of the maximum-likelihood feature vector $\boldsymbol{x}$ tends to infinity.

Let the target location for the i-th maximum-likelihood feature vector be denoted by $\boldsymbol{t}_i$ and given. Optimizing the parameters of the classifier to achieve these locations (with the maximum-likelihood feature vectors constrained to the unit ball centered at the origin) can be formalised as

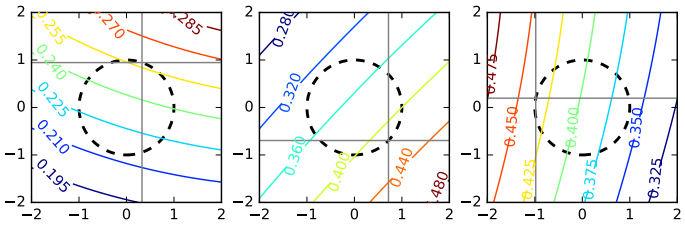
$$\begin{aligned} \text{minimize}_\Theta \quad & \frac{1}{2}\sum_{i=1}^{m} \|\boldsymbol{g}_i(\Theta) - \boldsymbol{t}_i\|^2 \\ \text{subject to} \quad & \boldsymbol{g}_i(\Theta) = \operatorname*{argmax}_{\boldsymbol{x}:\|\boldsymbol{x}\|_2 \leq 1} \log \ell_i(\boldsymbol{x}; \Theta) \end{aligned} \tag{81}$$

Solving by gradient descent gives updates

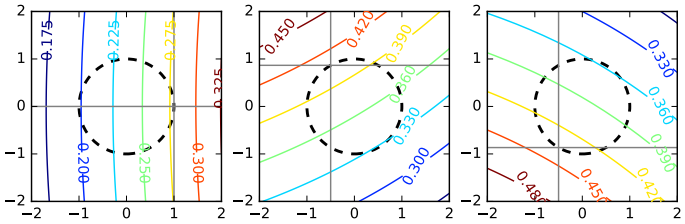
$$\theta^{(t+1)} \leftarrow \theta^{(t)} - \eta \sum_{i=1}^{m} (\boldsymbol{g}_i(\theta) - \boldsymbol{t}_i)^T \boldsymbol{g}_i'(\theta) \tag{82}$$

for any $\theta \in \Theta = \{(a_{ij}, b_i)\}_{i=1}^m$. Here η is the step size.

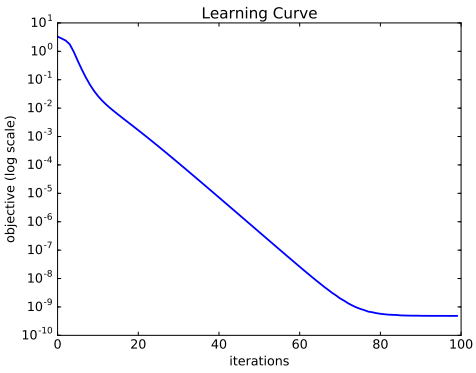
Example likelihood surfaces for the three classes in our example are shown in Figure 6 for both initial parameters and final (optimized) parameters where we have set the target locations to be evenly spaced around the unit circle. Notice that this is achieved by the final parameter settings. Also shown in Figure 6(c) is the learning curve (in log-scale). Here we see a rapid decrease in the objective in the first 20 iterations and final convergence (to within 10^{-9} of the optimal value) in under 100 iterations.



(a) Initial



(b) Final



(c) Learning Curve

Figure 6: Example bi-level optimization to position maximum-likelihood points on inequality constrained soft-max model. Shown are the initial and final maximum-likelihood surfaces $\ell_i(\boldsymbol{x})$ for $i = 1, 2, 3$ and constraint $\|\boldsymbol{x}\|_2 \leq 1$ in (a) and (b), respectively. Also shown is the learning curve in (c) where the objective value is plotted on a log-scale versus iteration number.

6 Discussion

We have presented results for differentiating parameterized argmin and argmax optimization problems with respect to their parameters. This is useful for solving bi-level optimization problems by gradient descent [Bard, 1998]. The results give exact gradients but (i) require that function being optimized (within the argmin or argmax) be smooth and (ii) involve computing a Hessian matrix inverse, which could be expensive for large-scale problems. However, in practice the methods can be applied even on non-smooth functions by approximating the function or perturbing the current solution to a nearby differentiable point. Moreover, for large-scale problems the Hessian matrix can be approximated by a diagonal matrix and still give a descent direction as was recently shown in the context of convolutional neural network (CNN) parameter learning for video recognition via stochastic gradient descent [Fernando and Gould, 2016].

The problem of solving non-smooth large-scale bi-level optimization problems, such as in CNN parameter learning for video recognition, present some interesting directions for future research. First, given that the parameters are likely to be changing slowly for any first-order gradient update it would be worth investigating whether warm-start techniques would be effectivey for speeding up gradient calculations. Second, since large-scale problems often employ stochastic gradient procedures, it may only be necessary to find a descent direction rather than the direction of steepest descent. Such an approach may be more computationally efficient, however it is currently unclear how such a direction could be found (without first computing the true gradient). Last, the results reported herein are based on the optimal solution to the lower-level problem. It would be interesting to explore whether non-exact solutions could still lead to descent directions, which would greatly improve efficiency for large-scale problems, especially during the early iterations where the parameters are likely to be far from their optimal values.

Models that can be trained end-to-end using gradient-based techniques have rapidly become the leading choice for applications in computer vision, natural language understanding, and other areas of artificial intelligence. We hope that the collection of results and examples included in

this technical report will help to develop more expressive models—specifically, ones that include optimization sub-problems—that can still be trained in an end-to-end fashion. And that these models will lead to even greater advances in AI applications into the future.

References

J. F. Bard. *Practical Bilevel Optimization: Algorithms and Applications*. Kluwer Academic Press, 1998.

S. P. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge, 2004.

S. Dempe and S. Franke. On the solution of convex bilevel optimization problems. *Computational Optimization and Applications*, pages 1–19, 2015.

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 (NIPS)*, 2007.

J. Domke. Generic methods for optimization-based modeling. In *Proc. of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2012.

O. Faugeras. *Three-dimensional Computer Vision*. MIT Press, 1993.

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.

B. Fernando, P. Anderson, M. Hutter, and S. Gould. Discriminative hierarchical rank pooling for activity recognition. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

T. Klatzer and T. Pock. Continuous hyper-parameter learning for support vector machines. In *Computer Vision Winter Workshop (CVWW)*, 2015.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

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.

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.

J. Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.