

# CGD: MODIFYING THE LOSS LANDSCAPE BY GRADIENT REGULARIZATION

Anonymous authors

Paper under double-blind review

## ABSTRACT

Line-search methods are commonly used to solve optimization problems. The simplest line search method is the steepest descent where we always move in the direction of the negative gradient. Newton’s method on the other hand is a second-order method that uses the curvature information in the Hessian to pick the descent direction. In this work, we propose a new line-search method called Constrained Gradient Descent (CGD) that implicitly changes the landscape of the objective function for efficient optimization. CGD is formulated as a solution to the constrained version of the original problem where the constraint is on a function of the gradient. We optimize the corresponding Lagrangian function thereby favourably changing the landscape of the objective function. This results in a line search procedure where the Lagrangian penalty acts as a control over the descent direction and can therefore be used to iterate over points that have smaller gradient values, compared to iterates of vanilla steepest descent. We reinterpret and draw parallels with the Explicit Gradient Regularization (EGR) method, discussing its drawbacks and potential enhancements. Numerical experiments are conducted on synthetic test functions to illustrate the performance of CGD and its variants.

## 1 INTRODUCTION

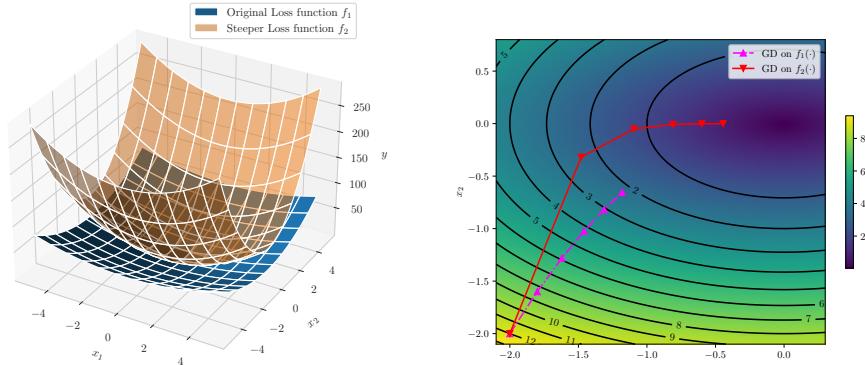


Figure 1: (Left) Loss function  $f_1(\mathbf{x}) = x_1^2 + 2x_2^2$  and a steeper loss function  $f_2(\mathbf{x})$ . (Right) GD trajectories on functions  $f_1(\mathbf{x})$  and  $f_2(\mathbf{x})$ . The two trajectories correspond to 5 iterations of GD on functions  $f_1(\mathbf{x})$  and  $f_2(\mathbf{x})$  respectively for fixed step size  $\alpha = 0.05$ . The contours and the gradient norm heatmap are over the function  $f_1(\mathbf{x})$ .

Deep Neural Networks have been successful at a variety of tasks primarily due to their over-parameterization. However, with such over-parameterization comes the problem of an uneven loss landscape that contains many local maxima and minima with some providing better performance than the others (Neyshabur et al., 2017; Li et al., 2018). As a result, optimizing over the parameter space has become incredibly relevant in the current times. To tackle this problem, one of the approach that is used is to modify the loss/objective function in such a way that the new function is easier to optimize. This is typically done by adding a regularization term that will make the

loss landscape more smooth, and which would in turn reduce the chances of the optimizer reaching sub-optimal minima, thereby improving generalizability. Considerable research in deep learning is now focused on designing loss functions that have smoother landscapes and therefore are easier to optimize (Du et al., 2022; Zhuang et al., 2022; Barrett & Dherin, 2022; Zhao et al., 2022; Karakida et al., 2023).

In this work, we focus on this idea of modifying the objective function for effective numerical optimization and propose an algorithmic paradigm to achieve it. To illustrate the key idea, consider minimizing a loss function  $f_1(\cdot)$  over  $\mathbb{R}^2$  as shown in Figure 1 (*Left*). Additionally consider a steeper loss function  $f_2(\cdot)$  such that for every  $\mathbf{x} \in \mathbb{R}^2$  we have  $f_1(\mathbf{x}) \leq f_2(\mathbf{x})$ . Note that the functions are such that their minima coincides. Now perform gradient descent (GD) with a fixed step size on both these functions. Their respective trajectories are plotted in Figure 1 (*Right*) on the contours of  $f_1$ . It is clear from the figure that the iterates of GD on  $f_2$  are much closer to the minima as compared to iterates of GD on  $f_1$ . We make this observation a focal point of this work and investigate if one can achieve the latter iterates (iterates from GD on  $f_2$ ) on the former function ( $f_1$ ).

Towards this, we propose Constraint gradient descent (CGD), a variant of the GD algorithm which achieves this by constraining the gradient norm to be appropriately small. Instead of solving this constrained optimization problem, we consider the Lagrangian of this function and perform GD on it. The Lagrangian parameter  $\lambda$  controls the penalty on gradient norm and thereby controls the steepness of the modified function. It is on this modified function that we seek to apply GD to possibly attain iterates that are much closer to the local minima as compared to GD iterates. From Figure 1 (*Right*) we additionally see that compared to GD, the CGD iterates (which is nothing but GD iterates on  $f_2$ ) have a lower gradient norm. Since the path to minima is over points with lower gradient norm, this we believe is an attractive feature and even amounts to better generalization properties for high dimensional functions such as loss landscapes of neural networks.

The idea of flat minima and their attractiveness goes long back to Hochreiter & Schmidhuber (1997): An optimal minimum is considered “flat” if the test error changes less in its neighbourhood. Keskar et al. (2017) and Chaudhari et al. (2017) observe better generalization results for neural networks at flat minima. Gradient regularization on the other hand has only been recently studied from a perspective of deep neural networks (DNNs) (Zhao et al., 2022; Karakida et al., 2023; Barrett & Dherin, 2022; Smith et al., 2021) where the square of the  $L_2$  norm is penalized.

While the above methods only penalize the squared  $L_2$ -norm of the gradient, we propose a more general approach where the penalty could be on any positive function of the gradient. This opens doors to investigating potential applications where penalty functions beyond squared  $L_2$  norm might prove more beneficial. A key aim of this work is to understand this gradient regularization from the perspective of a numerical optimization algorithm and identify properties and features that may not have been obvious earlier. We believe CGD and its variants discussed in this work have the ability to find flatter minimas which may prove to be useful for training of neural networks.

We summarize our contributions below:

- We propose a new line-search procedure called Constraint Gradient Descent (CGD) that performs gradient descent on a gradient regularized loss function.
- While CGD requires the Hessian information, we also propose a first-order variant of CGD using finite-difference approximation of the Hessian called *CGD-FD*. We define appropriate stopping criteria in CGD-FD for settings where gradient computation can be expensive.
- We conduct experiments over synthetic test functions to compare the performance of CGD and its variants compared to standard line-search procedures.
- Our work also provides new insights to gradient regularization based methods. In fact, we re-interpret and identify pitfalls in Explicit Gradient Regularization (EGR) Methods using our formulation.

The rest of the paper is organized as follows. In the next section, we recall some preliminaries on line-search methods. We then discuss the CGD algorithm and propose its variants. We then illustrate the performance of our algorithm on several test functions and conclude with a discussion on future directions.

108    **2 NOTATION AND PRELIMINARIES**  
 109

110    The set of real numbers and non-negative real numbers is denoted by  $\mathbb{R}$  and  $\mathbb{R}^+$  respectively. We  
 111    consider a vector  $\mathbf{x} \in \mathbb{R}^n$  as a column vector given by  $\mathbf{x} = [x_1 \ x_2 \ \dots \ x_n]^T$ . We use a  
 112    boldface letter to denote a vector and lowercase letters (with subscripts) to denote its components.  
 113     $L^p$ -norm of a vector  $\mathbf{x}$  is defined as  $\|\mathbf{x}\|_p = (\sum_i |x_i|^p)^{1/p}$ . Setting  $p = 2$  gives us  $L^2$ -norm:  
 114     $\|\mathbf{x}\| \triangleq \|\mathbf{x}\|_2 = \sqrt{\sum_i x_i^2} = \sqrt{\mathbf{x}^T \mathbf{x}}$ .  $I_n$  denotes the identity matrix of size  $n \times n$ . All zero vector of  
 115    length  $n$  is denoted by  $\mathbf{0}_n$ .  
 116

117    **2.1 LINE-SEARCH METHODS**  
 118

119    Let  $f(\mathbf{x})$  be a twice differential function with domain  $\mathcal{D} \subseteq \mathbb{R}^n$  and codomain  $\mathbb{R}$ , i.e.,  $f : \mathcal{D} \rightarrow \mathbb{R}$ .  
 120    Let  $\nabla f(\mathbf{x}_k)$  and  $H(\mathbf{x}_k)$  denote the gradient and Hessian of the function  $f(\mathbf{x})$  evaluated at point  
 121     $\mathbf{x}_k \in \mathcal{D}$ . For the sake of simplicity, we will also use the notation  $\nabla f_k$  and  $H_k$  to denote  $\nabla f(\mathbf{x}_k)$  and  
 122     $H(\mathbf{x}_k)$  respectively. For the given function  $f(\mathbf{x})$ , we consider the problem of finding its minimizer,  
 123    i.e., we wish to find  $\mathbf{x}^* \in \mathcal{D}$  such that

$$124 \quad \mathbf{x}^* = \arg \min_{\mathbf{x} \in \mathcal{D}} f(\mathbf{x}). \quad (1)$$

125    We focus on the situation where  $\mathbf{x}^*$  is obtained using an iterative line-search procedure (for details  
 126    refer Nocedal & Wright (2006, Ch. 3)). The steepest descent (*gradient descent*) method, Newton's  
 127    method, and quasi-Newton's method are some examples of line-search methods. In an iterative  
 128    algorithm, the key idea is to begin with an initial guess  $\mathbf{x}_0$  for the minimizer and generate a sequence  
 129    of vectors  $\mathbf{x}_0, \mathbf{x}_1, \dots$  until convergence (or until desired level of accuracy is achieved). In such  
 130    algorithms,  $\mathbf{x}_{k+1}$  is obtained using  $\mathbf{x}_k$  using a pre-defined update rule. For line-search methods, the  
 131    update rule can be written in a general form as follows,

$$133 \quad \mathbf{x}_{k+1} = \mathbf{x}_k + \alpha \mathbf{p}_k \quad (2)$$

134    where  $\alpha \in \mathbb{R}^+$  is the step size (or learning rate) and  $\mathbf{p}_k \in \mathbb{R}^n$  corresponds to the direction in the  $k$ -th  
 135    iteration. For some line-search methods, we can express  $\mathbf{p}_k$  as  $\mathbf{p}_k = -P_k \nabla f_k$  where  $P_k \in \mathbb{R}^{n \times n}$ .  
 136    For example, for steepest gradient descent method we have  $P_k = -I_n$  and for Newton's method  
 137     $P_k = -H_k^{-1}$ . For  $\mathbf{p}_k$  to be a *descent* direction, the following condition must hold:

$$139 \quad \nabla f_k^T \mathbf{p}_k < 0. \quad (3)$$

141    **3 CONSTRAINED GRADIENT DESCENT**  
 142

143    In this section, we propose an iterative line-search method to find a minimizer  $\mathbf{x}^*$  of the given  
 144    function  $f(\mathbf{x})$  (see Equation 1). We shall refer to our approach as *Constrained Gradient Descent*  
 145    (CGD) method. It is known that if  $\mathbf{x}^*$  is a local minimizer of the function  $f(\mathbf{x})$  then the gradient  
 146    at point  $\mathbf{x}^*$  is equal to zero (Nocedal & Wright, 2006, Ch. 2). The key idea in our approach is to  
 147    focus on the set of  $\mathbf{x} \in \mathcal{D}$  such that  $\nabla f(\mathbf{x})$  is close to zero. For this consider a general constrained  
 148    optimization problem:

$$149 \quad \mathbf{x}^* = \arg \min_{\substack{\mathbf{x} \in \mathcal{D} \\ h(\nabla f(\mathbf{x})) \leq \epsilon}} f(\mathbf{x}) \quad (4)$$

150    where the constraint  $h(\cdot)$  is defined on the gradient and  $\epsilon$  is a small positive real number. The uncon-  
 151    strained optimization problem corresponding to Equation 4 is obtained by penalizing the gradient  
 152    constraint with a Lagrange multiplier  $\lambda > 0$ :

$$155 \quad \mathbf{x}^* = \arg \min_{\mathbf{x} \in \mathcal{D}} \left[ f(\mathbf{x}) + \lambda h(\nabla f(\mathbf{x})) \right] \quad (5)$$

156    Note that  $\epsilon$  doesn't affect the optimization and hence has been removed from the objective. Observe  
 157    that such a formulation provides us with a *modified* loss function to optimize over. Choosing a non-  
 158    negative constraint  $h(\cdot)$  is preferred so that we do not introduce artificial minima points. In this case,  
 159    the modified loss function will become steeper and hence easier to descent on than the original loss  
 160    function. This can also be referred to by *penalization* or using a *gradient penalty* since the objective  
 161    is penalized at points which are not local minima depending on the gradients at these points.

We consider non-decreasing functions over the gradient-norm to be suitable choices for the constraint function  $h(\cdot)$  since the above mentioned qualities are fulfilled by them.

Consider, two modified loss functions using  $L^p$ -norm gradient penalty:

$$g_p(\mathbf{x}) \triangleq f(\mathbf{x}) + \lambda \|\nabla f(\mathbf{x})\|_p \quad (L^p\text{-norm}) \quad (6)$$

$$\hat{g}_p(\mathbf{x}) \triangleq f(\mathbf{x}) + \lambda \|\nabla f(\mathbf{x})\|_p^p \quad (L^p\text{-norm to the power } p) \quad (7)$$

For ease of exposition, we use  $\hat{g}_2(\mathbf{x})$  for most of our discussion and numerical results. We simplify the notation further to denote  $g(\mathbf{x}) \triangleq \hat{g}_2(\mathbf{x})$ . More details on using functions  $g_p(\mathbf{x})$  and  $\hat{g}_p(\mathbf{x})$  can be found in Appendix A. Now consider the following optimization problem,

$$\mathbf{x}^* = \arg \min_{\mathbf{x} \in \mathcal{D}} g(\mathbf{x}) \quad (8)$$

where  $g(\mathbf{x}) \triangleq f(\mathbf{x}) + \lambda \|\nabla f(\mathbf{x})\|^2 = f(\mathbf{x}) + \lambda (\nabla f(\mathbf{x})^T \nabla f(\mathbf{x}))$ . We now apply steepest descent iteration to the optimization problem in Equation 8, given as

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{x}_k - \alpha \nabla g_k \\ &= \mathbf{x}_k - \alpha (\nabla f_k + 2\lambda H_k \nabla f_k) \\ &= \mathbf{x}_k - \alpha B_k \nabla f_k \end{aligned} \quad (9)$$

where  $B_k \triangleq I_n + 2\lambda H_k$ . From Equation 2 note that we have  $\mathbf{p}_k = -B_k \nabla f_k$  and  $P_k = B_k$  which is the matrix corresponding to the direction taken at the  $k$ -th iteration by our CGD method.

Revisiting the example in Figure 1, the function  $f_1(\mathbf{x}) = x_1^2 + 2x_2^2$  was penalized with the square of  $L^2$ -norm of gradient as given in Equation 8. Thus, the modified loss function  $f_2(\mathbf{x}) = x_1^2 + 2x_2^2 + \lambda ((2x_1)^2 + (4x_2)^2) = (1 + 4\lambda)x_1^2 + 2(1 + 8\lambda)x_2^2$  where  $\lambda$  was chosen to be 0.4.

We now provide a lemma that investigates if the penalized objective function has stationary points which are different from the original function and if so characterizes them.

**Lemma 1.** Let  $\mathcal{S}_{\hat{\mathbf{x}}}$  and  $\mathcal{S}_{\mathbf{x}^*}$  be the stationary points of  $f(\mathbf{x})$  and  $g(\mathbf{x})$  as defined in Equation 8. Then,  $\mathcal{S}_{\hat{\mathbf{x}}} \subseteq \mathcal{S}_{\mathbf{x}^*}$ . Furthermore, for any  $\mathbf{x}^* \in \mathcal{S}_{\mathbf{x}^*}$  one of the following is true: (a)  $\mathbf{x}^* \in \mathcal{S}_{\hat{\mathbf{x}}}$  or (b)  $\nabla f(\mathbf{x}^*)$  is an eigenvector of  $H(\mathbf{x}^*)$  with the eigenvalue  $-\frac{1}{2\lambda}$ .

*Proof.* For any  $\hat{\mathbf{x}} \in \mathcal{S}_{\hat{\mathbf{x}}}$  we have  $\nabla f(\hat{\mathbf{x}}) = \mathbf{0}_n$  and hence  $\nabla g(\hat{\mathbf{x}}) = \mathbf{0}_n$  from Equation 9. Thus,  $\mathcal{S}_{\hat{\mathbf{x}}} \subseteq \mathcal{S}_{\mathbf{x}^*}$  trivially holds. Now for any  $\mathbf{x}^* \in \mathcal{S}_{\mathbf{x}^*}$ ,

$$\nabla g(\mathbf{x}^*) = 0 \implies (I_n + 2\lambda H(\mathbf{x}^*)) \nabla f(\mathbf{x}^*) = 0$$

Therefore if  $\nabla f(\mathbf{x}^*) = 0$ , we have  $\mathbf{x}^* \in \mathcal{S}_{\hat{\mathbf{x}}}$ . Otherwise,  $H(\mathbf{x}^*) \nabla f(\mathbf{x}^*) = -\frac{1}{2\lambda} \nabla f(\mathbf{x}^*)$ .  $\square$

The above lemma illustrates that additional stationary points could possibly be introduced and some of these points could also be a local minima. The above lemma also characterizes conditions under which this is true and therefore such points can easily be detected. A perturbation from the current  $\lambda$  in that case results in the iterate to descend further, possibly moving towards a better local minima.

Within the current scheme, the nature of stationary points  $f(\mathbf{x})$  might change in  $g(\mathbf{x})$ . Particularly, the local maxima and saddle points of  $f(\mathbf{x})$  might become local minima in  $g(\mathbf{x})$  depending on the coefficient  $\lambda$ . This, happens due to the fact that at the stationary points, the modified function takes the value of the original function itself (since gradient is zero). But within the neighbourhood, the function value increases (since we are adding gradient-norm at these points). Thus, for a particular local maximum (or saddle point)  $\mathbf{y}$ , the nature of the point changes to that of a local minimum when  $\lambda > \lambda^*$  where  $\lambda^*$  represents a threshold for this behaviour.  $\lambda^*$  depends on the function's curvature and gradient norm changes in the neighbourhood of the point  $\mathbf{y}$  and hence, is different for different local maxima and saddle points.

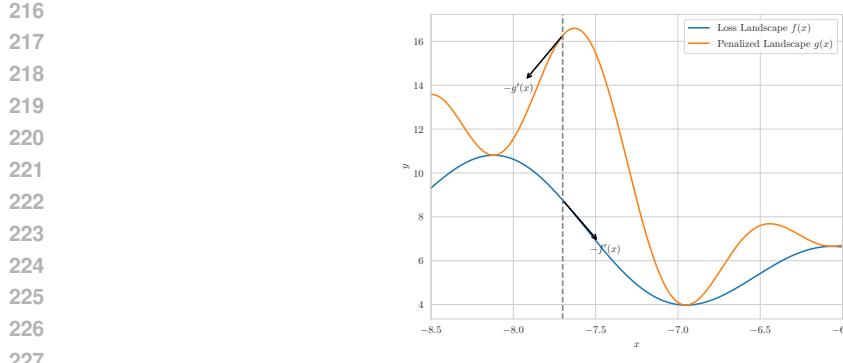


Figure 2: For the given function, observe artificial stationary points being introduced after penalization at  $x \approx -6.5$  and  $x \approx -7.6$ . Both these stationary points turn out to be local maxima. Observe how the local maximum at  $x \approx -8.1$  turns into a local minimum. We also plot the descent directions  $-f'(x)$  and  $-g'(x)$  at  $x = -7.7$  (denoted by dotted line).

Thus within the current scheme, descent on the penalized function might actually cause ascent on the original one (moving towards the local maxima turned into local minima). For example, in Figure 2 we observe that at  $x = -7.7$ , steepest descent along  $-\nabla g(\mathbf{x})$  direction would actually cause an ascent over the original loss function.

To fix this behaviour, we ensure that we only move along the direction  $-\nabla g(\mathbf{x})$  when it is a descent direction. Therefore Equation 3 requires that  $\nabla f(\mathbf{x})^T \nabla g(\mathbf{x}) > 0$  in order for us to move along  $-\nabla g(\mathbf{x})$ . If Equation 3 doesn't hold i.e,  $\nabla f(\mathbf{x})^T \nabla g(\mathbf{x}) \geq 0$  then we set  $\lambda = 0$  at this point and only move along  $-\nabla f(\mathbf{x})$  direction. Note that this simple check also helps us to avoid stopping at artificially introduced stationary points. We summarize CGD in Algorithm 1.

---

**Algorithm 1** Constrained Gradient Descent (CGD)

---

**Input:** Objective function  $f : \mathcal{D} \rightarrow \mathbb{R}$ , initial point  $\mathbf{x}_0$ , max iterations  $T$ , step size  $\alpha$ , regularization coefficient  $\lambda$ .

**Output:** Final point  $\mathbf{x}_T$ .

```

248 1: for iteration  $k = 0, \dots, T - 1$  do
249 2:    $\mathbf{p}_k \leftarrow -(I_n + 2\lambda_k H_k) \nabla f_k$ 
250 3:   if  $\nabla f_k^T \mathbf{p}_k < 0$  then                                ▷ Check if  $\mathbf{p}_k$  is a Descent Direction
251 4:      $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha \mathbf{p}_k$ 
252 5:   else
253 6:      $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k - \alpha \nabla f_k$ 
254 7:   end if
255 8: end for
256 9: return  $\mathbf{x}_T$ 

```

---

### 3.1 CGD-FD: FINITE DIFFERENCE APPROXIMATION OF THE HESSIAN

Note that CGD is a second-order optimization algorithm as it requires the Hessian information to compute each iterate. We improve over this complexity by restricting CGD to be a first-order line search method wherein the Hessian is approximated using a finite difference (Pearlmutter, 1994). Using Taylor series, we know that

$$\nabla f(\mathbf{x}_k + \Delta \mathbf{x}) = \nabla f_k + H_k \Delta \mathbf{x} + \mathcal{O}(\|\Delta \mathbf{x}\|^2).$$

Now let  $\Delta \mathbf{x} = r \mathbf{v}$  where  $r$  is arbitrarily small. Then, we can rewrite the above expression as:

$$H \mathbf{v} = \frac{\nabla f(\mathbf{x}_k + r \mathbf{v}) - \nabla f_k}{r} + \mathcal{O}(r).$$

270 Substituting  $\mathbf{v} = \nabla f_k$  and using this approximation in Equation 9 gives us  
 271

$$\begin{aligned} 272 \quad \nabla g(\mathbf{x}_k) &\approx \nabla f_k + 2\lambda \frac{\nabla f(\mathbf{x}_k + r\nabla f_k) - \nabla f_k}{r} \\ 273 \quad &= (1 - \nu)\nabla f_k + \nu\nabla f(\mathbf{x}_k + r\nabla f_k) \end{aligned} \quad (10)$$

275 where  $\nu = 2\lambda/r$ . We call this version *CGD with Finite Differences* (CGD-FD).  
 276

277 Note now that we require two gradient calls in every iteration of CGD-FD which may be prohibitive.  
 278 Towards this it is only natural to consider a stopping criteria so as to revert back to using steepest  
 279 direction whenever the improvement through CGD-FD iterates is low. This is done to avoid the extra  
 280 gradient evaluations after the optimizer has moved to a sufficiently optimal point in the domain.  
 281 Such criteria are particularly helpful in loss functions where gradient evaluations are costly and  
 282 therefore, budgeted. This also gives us the freedom to play around with which direction to choose  
 283 (steepest or CGD-FD) and quantify when to switch to the other. For instance, we initially only move  
 284 in the directions suggested by CGD-FD and switch to steepest once we make good-enough drop in  
 285 the function value from the initial point.

286 In our experiments, we used a combination of the strategies explained above. We only use CGD-FD  
 287 for the first  $b$  iterations (out of the total budget  $T$ ) to strive for a good-drop in function values initially.  
 288 While doing so, we also ensure that we only move in the direction  $\mathbf{p}_k$  (CGD-FD, Equation 10) if  
 289 it is a descent direction. Otherwise, we stop using CGD-FD updates henceforth and use steepest  
 290 direction from here onwards. We summarize our CGD-FD method in Algorithm 2.

---

291 **Algorithm 2** Constrained Gradient Descent using Finite Differences (CGD-FD)

292 **Input:** Objective function  $f : \mathcal{D} \rightarrow \mathbb{R}$ , initial point  $\mathbf{x}_0$ , max iterations  $T$ , step size  $\alpha$ , regularization  
 293 coefficients  $\lambda$ , stopping threshold  $b$ .

294 **Output:** Final point  $\mathbf{x}_T$ .

```

295 1:  $\nu \leftarrow 2\lambda/r$ 
296 2: USE_CGD  $\leftarrow$  TRUE
297 3: Number of Gradient Evaluations  $c \leftarrow 0$                                  $\triangleright$  Can be interpreted as cost
298 4: for  $k = 0, \dots, T - 1$  do
299 5:   if  $c = T$  then return  $\mathbf{x}_k$                                                $\triangleright$  Budget Exhausted
300 6:   end if
301 7:   if USE_CGD then
302 8:      $\mathbf{p}_k \leftarrow -[(1 - \nu)\nabla f_k + \nu\nabla f(\mathbf{x}_k + r\nabla f_k)]$ 
303 9:      $c \leftarrow c + 2$ 
304 10:    if  $\nabla f_k^T \mathbf{p}_k < 0$  then                                          $\triangleright$  Check if  $\mathbf{p}_k$  is a Descent Direction
305 11:       $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k + \alpha \mathbf{p}_k$ 
306 12:    else
307 13:       $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k - \alpha \nabla f_k$ 
308 14:      USE_CGD  $\leftarrow$  FALSE                                               $\triangleright$  Stop using CGD-FD iterates
309 15:    end if
310 16:  else
311 17:     $\mathbf{x}_{k+1} \leftarrow \mathbf{x}_k - \alpha \nabla f_k$ 
312 18:     $c \leftarrow c + 1$ 
313 19:  end if
314 20:  if  $k \geq b$  then
315 21:    USE_CGD  $\leftarrow$  FALSE
316 22:  end if
317 23: end for
318 24: return  $\mathbf{x}_T$ 

```

---

319 **4 NUMERICAL EXPERIMENTS**

320 We test CGD-FD on synthetic test functions from *Virtual Library of Simulation Experiments: Test*  
 321 *Functions and Datasets*<sup>1</sup>.

322 <sup>1</sup><http://www.sfu.ca/~ssurjano>

For each function, we strive for an initial gain i.e., a good-drop in function value while using CGD-FD iterates. We set the stopping threshold  $b$  as  $T/4$  where  $T$  is the total budget. For our experiments we chose  $T = 40$ .

For the hyperparameter  $\lambda$ , we empirically tested different schedules and strategies for different kinds of functions. We observe that using constant  $\lambda$  works very well with convex functions. However, in non-convex functions an increasing schedule might be useful. The intuition behind an increasing schedule is that initially we want to penalize less and as we move to points with lower gradient norm, we can start to penalize more. We denote this as  $\text{Linear}(a, b)$  which is an increasing linear schedule of  $T$  values going from  $a$  to  $b$ . For the step-size  $\alpha$ , a constant-value was found to be suitable in all our experiments.

To measure the initial drop in function value, we compare the *Improvement* for the first step of CGD-FD vs the first step of steepest descent. Improvement in the first step is defined as:

$$\text{Improvement (in \%)} = \frac{f(\mathbf{x}_0) - f(\mathbf{x}_1)}{f(\mathbf{x}_0)} * 100$$

Table 1 summarizes the hyperparameter values chosen and Table 2 compares the intial improvement across CGD-FD and steepest descent.

Table 1: Choices of  $\alpha$  and  $\lambda$  for synthetic test functions (Budget  $T = 40$ , Stopping threshold  $b = T/4$ ).

Test Function (Dimensions=n)	$\lambda$	$\alpha$
Quadratic function ( $n = 10$ )	0.4	0.01
Rotated hyper-ellipsoid function ( $n = 5$ )	0.5	0.01
Levy function ( $n = 2$ )	Linear(0.01, 0.1)	0.05
Branin function ( $n = 2$ )	0.07	0.01
Griewank function ( $n = 2$ )	40.0	0.01
Matyas function ( $n = 2$ )	10.0	0.01

Table 2: Initial Improvement (in %) across CGD-FD and Steepest Descent (GD).

Test Function (Dimensions=n)	GD	CGD-FD
Quadratic function ( $n = 10$ )	18.89	<b>97.91</b>
Rotated hyper-ellipsoid function ( $n = 5$ )	15.94	<b>82.76</b>
Levy function ( $n = 2$ )	23.73	<b>63.21</b>
Branin function ( $n = 2$ )	37.53	<b>87.07</b>
Griewank function ( $n = 2$ )	0.01	<b>0.08</b>
Matyas function ( $n = 2$ )	1.83	<b>34.40</b>

We plot the function values vs gradient evaluations in Figure 3. Note, that the  $x$ -axis is not iterations but gradient evaluations. We compare the trajectories across points where equal number of gradients have been evaluated.

## 5 EXPLICIT GRADIENT REGULARIZATION (EGR)

Barrett & Dherin (2022) proposed **Explicit Gradient Regularization** (EGR) where the original loss function is regularized with the square of  $L^2$ -norm of the gradient. This regularized objective is then optimized with the intention that the model’s parameters will converge to a *flat-minima* and thus, be more generalizable. However, an understanding of why this happens is missing.

We explain EGR through the example visualized in Figure 4 (*Left*). Based on our formulation explained in Section 3, we can interpret the gradient-regularized function as a more steeper version of the orginal loss function wherein the minima remain same. Due to the gradient penalty, a steeper minima (in the original loss function) would turn more steeper while the increase in steepness

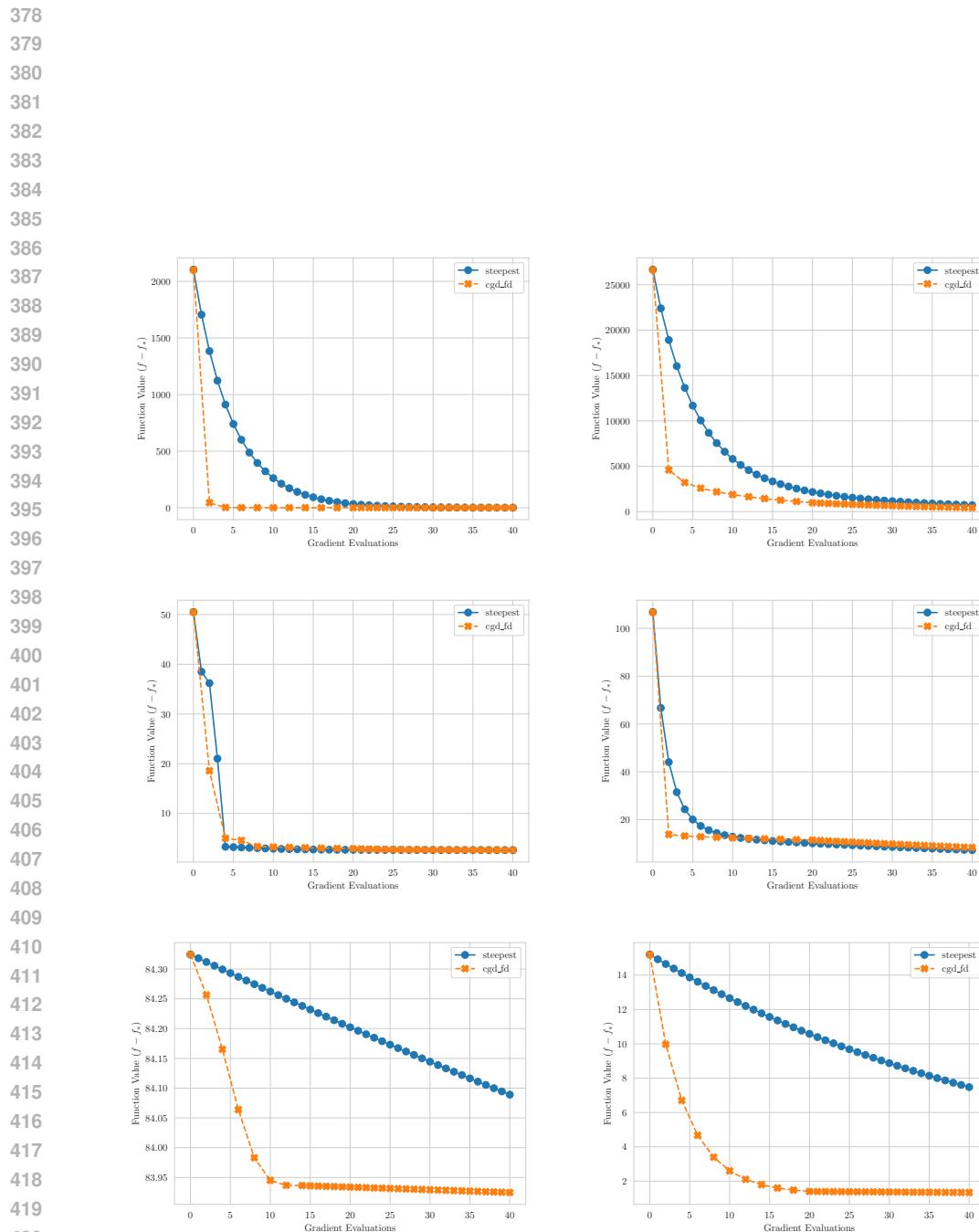


Figure 3: Function value  $f(\cdot) - f^*$  ( $f^*$  being the optimal function value) vs Gradient Evaluations. In raster order (from top to bottom; left to right), the functions are: Quadratic function, Rotated hyper-ellipsoid function, Levy function, Branin function, Griewank function and Matyas function (see Appendix B). Note: The  $x$ -axis of each plot is not iterations but number of gradients evaluated.

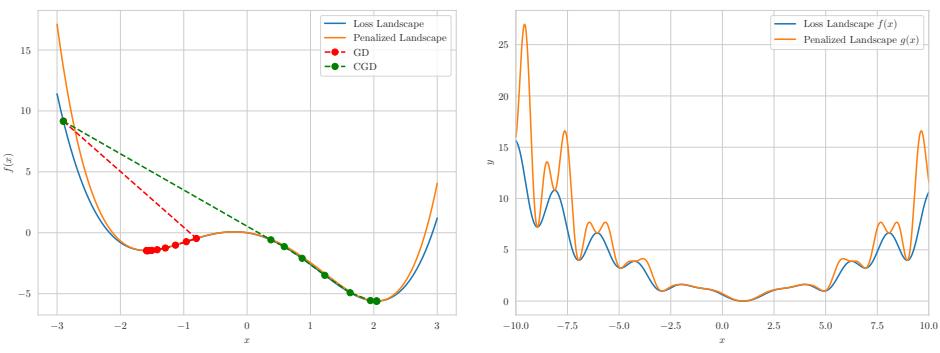


Figure 4: (Left) CGD is able to move to a better local minimum by moving along the negative gradients over the modified loss function. (Right) Local maxima of the original loss function turning into local minima of the penalized loss function.

wouldn't be this high at a flatter minimum. Thus, at a constant step-size, the optimizer will likely overshoot over the sharper minimum due to the extremely high gradient value, while still being able to converge to the less-steep and preferred flat minimum point. In Figure 4 (Left) we see how GD gets stuck at a suboptimal point of the function while CGD (GD over the penalized loss function) is able to avoid this point and converges to a better (more optimal) minimum point.

EGR has the drawbacks of having fictitious minima identified in Lemma 1. Since, the loss landscape of neural networks is highly uneven (Li et al., 2018), it's likely that artificial stationary points are introduced and the optimizer might converge at local-maxima points since their nature changes with regularization. For example, in Figure 4 (Right) we observe artificial stationary points being introduced between local-minima and local-maxima of the original loss function while some local-maxima also turn into local-minima of the penalized loss function. Therefore, one needs to ensure that there are suitable fixes for these scenarios for better performance. Specifically, the direction along of the penalized loss function,  $-\nabla g(\cdot)$  should be a descent direction and that we shouldn't stop at points where  $\nabla g(\cdot) = \mathbf{0}$  but  $\nabla f(\cdot) \neq \mathbf{0}$  as discussed in Lemma 1.

Another downfall with EGR is that it requires hessian evaluation for each mini-batch while training. This might become a costly operation for bigger models and hence some approximation of the hessian should be used. Another thing we observe from the experimental results for CGD-FD, is that the improvement through optimizing the regularized function is mostly only during the initial steps. Hence, one should only use EGR for some initial steps to reach a good-enough starting point while not exhausting the budget for gradient evaluations.

## 6 FUTURE WORKS

In this work, we considered a new line swarch method that penalizes the norm of the gradient and provides iterates that have lower gradient norm compared to vanilla gradient descent. We identify properties of this algorithm, provide a variant that does not require Hessian and illustrate connections to the widely popular explicit gradient regularization literature.

There are several future directions arising from this work. We would like to investigate in greater detail the role of different penalty functions and norms beyond  $L_2$ . We would like to investigate applications of this method in more diverse settings like reinforcement learning and even Bayesian optimization. We hope this work sputters more discussions on gradient regularization helping in neural network generalization.

## REFERENCES

- David G. T. Barrett and Benoit Dherin. Implicit gradient regularization, 2022. URL <https://arxiv.org/abs/2009.11162>.

- 486 Pratik Chaudhari, Anna Choromanska, Stefano Soatto, Yann LeCun, Carlo Baldassi, Christian  
 487 Borgs, Jennifer Chayes, Levent Sagun, and Riccardo Zecchina. Entropy-sgd: Biasing gra-  
 488 dient descent into wide valleys, April 2017. URL <http://arxiv.org/abs/1611.01838>.  
 489 arXiv:1611.01838 [cs, stat].  
 490
- 491 Jiawei Du, Daquan Zhou, Jiashi Feng, Vincent Tan, and Joey Tianyi Zhou. Sharpness-aware training  
 492 for free. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh (eds.), *Advances*  
 493 in *Neural Information Processing Systems*, volume 35, pp. 23439–23451. Curran Associates, Inc.,  
 494 2022. URL [https://proceedings.neurips.cc/paper\\_files/paper/2022/file/948b1c9d660d7286dd767cd07dabd487-Paper-Conference.pdf](https://proceedings.neurips.cc/paper_files/paper/2022/file/948b1c9d660d7286dd767cd07dabd487-Paper-Conference.pdf).  
 495
- 496 Sepp Hochreiter and Jürgen Schmidhuber. Flat Minima. *Neural Computation*, 9(1):1–42, 01  
 497 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.1.1. URL <https://doi.org/10.1162/neco.1997.9.1.1>.  
 498
- 500 Ryo Karakida, Tomomi Takase, Tomohiro Hayase, and Kazuki Osawa. Understanding gradient  
 501 regularization in deep learning: Efficient finite-difference computation and implicit bias. In An-  
 502 dreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan  
 503 Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume  
 504 202 of *Proceedings of Machine Learning Research*, pp. 15809–15827. PMLR, 23–29 Jul 2023.  
 505 URL <https://proceedings.mlr.press/v202/karakida23a.html>.  
 506
- 507 Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Pe-  
 508 ter Tang. On large-batch training for deep learning: Generalization gap and sharp minima, Febru-  
 509 ary 2017. URL <http://arxiv.org/abs/1609.04836>.  
 510
- 511 Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss land-  
 512 scape of neural nets. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi,  
 513 and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Cur-  
 514 ran Associates, Inc., 2018. URL [https://proceedings.neurips.cc/paper\\_files/paper/2018/file/a41b3bb3e6b050b6c9067c67f663b915-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2018/file/a41b3bb3e6b050b6c9067c67f663b915-Paper.pdf).  
 515
- 516 Behnam Neyshabur, Srinadh Bhojanapalli, David Mcallester, and Nati Srebro. Exploring general-  
 517 ization in deep learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vish-  
 518 wanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30.  
 519 Curran Associates, Inc., 2017. URL [https://proceedings.neurips.cc/paper\\_files/paper/2017/file/10ce03a1ed01077e3e289f3e53c72813-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2017/file/10ce03a1ed01077e3e289f3e53c72813-Paper.pdf).  
 520
- 521 Jorge Nocedal and Stephen J Wright. *Numerical Optimization*. Springer, 2nd edition, 2006.  
 522
- 524 Barak A. Pearlmutter. Fast Exact Multiplication by the Hessian. *Neural Computation*, 6(1):147–  
 525 160, 01 1994. ISSN 0899-7667. doi: 10.1162/neco.1994.6.1.147. URL <https://doi.org/10.1162/neco.1994.6.1.147>.  
 526
- 528 Samuel L. Smith, Benoit Dherin, David G. T. Barrett, and Soham De. On the origin of implicit  
 529 regularization in stochastic gradient descent, 2021. URL <https://arxiv.org/abs/2101.12176>.  
 530
- 531 Yang Zhao, Hao Zhang, and Xiuyuan Hu. Penalizing gradient norm for efficiently improving gener-  
 532 alization in deep learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari,  
 533 Gang Niu, and Sivan Sabato (eds.), *Proceedings of the 39th International Conference on Machine*  
 534 *Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 26982–26992. PMLR,  
 535 17–23 Jul 2022. URL <https://proceedings.mlr.press/v162/zhao22i.html>.  
 536
- 537 Juntang Zhuang, Boqing Gong, Liangzhe Yuan, Yin Cui, Hartwig Adam, Nicha Dvornek, Sekhar  
 538 Tatikonda, James Duncan, and Ting Liu. Surrogate gap minimization improves sharpness-aware  
 539 training, 2022. URL <https://arxiv.org/abs/2203.08065>.

540    **A  $L^p$ -NORM GRADIENT PENALTY**  
 541

542    **Lemma 2.** *For the modified functions  $g_p(\mathbf{x})$  and  $\hat{g}_p(\mathbf{x})$  defined in Equations 6 and 7, the gradients  
 543    are given as follows:*

$$\begin{aligned} 545 \quad \nabla g_p(\mathbf{x}) &= \nabla f(\mathbf{x}) + \frac{\lambda}{\|\nabla f(\mathbf{x})\|_p^{p-1}} H(\mathbf{x}) \left( \nabla f(\mathbf{x}) \odot \text{abs}(\nabla f(\mathbf{x}))^{p-2} \right) \\ 546 \\ 547 \quad \nabla \hat{g}_p(\mathbf{x}) &= \nabla f(\mathbf{x}) + \lambda p H(\mathbf{x}) \left( \nabla f(\mathbf{x}) \odot \text{abs}(\nabla f(\mathbf{x}))^{p-2} \right) \end{aligned}$$

549    where  $\odot$  is element-wise product of vectors and  $\text{abs}$  is element-wise absolute  $|\cdot|$  of the vector.  
 550

551    *Proof.* We know that  $L^p$ -norm of  $\mathbf{x}$  is  $\|\mathbf{x}\|_p = (\sum_i |x_i|^p)^{1/p}$ . We first find  $\nabla \|\mathbf{x}\|_p$ . For some  $x_k$ :

$$\begin{aligned} 554 \quad \frac{\partial \|\mathbf{x}\|_p}{\partial x_k} &= \frac{1}{p} \left( \sum_i |x_i|^p \right)^{\frac{1}{p}-1} \frac{\partial}{\partial x_k} \left\{ \sum_i |x_i|^p \right\} \\ 555 \\ 556 \quad &= \frac{1}{p} \left( \sum_i |x_i|^p \right)^{\frac{1}{p}-1} \left( \sum_i \frac{\partial |x_i|^p}{\partial x_k} \right) \\ 557 \\ 558 \quad &= \frac{1}{p} \left( \sum_i |x_i|^p \right)^{\frac{1}{p}-1} \left( \sum_i p|x_i|^{p-1} \frac{\partial |x_i|}{\partial x_k} \right) \\ 559 \\ 560 \quad &= \left( \sum_i |x_i|^p \right)^{\frac{1}{p}-1} \left( \sum_i |x_i|^{p-1} \frac{x_i}{|x_i|} \frac{\partial x_i}{\partial x_k} \right) \\ 561 \\ 562 \quad &= \left( \sum_i |x_i|^p \right)^{\frac{1-p}{p}} \left( \sum_i x_i |x_i|^{p-2} \frac{\partial x_i}{\partial x_k} \right) \\ 563 \\ 564 \quad &= \|\mathbf{x}\|_p^{1-p} \left( \sum_i x_i |x_i|^{p-2} \frac{\partial x_i}{\partial x_k} \right) \text{ or } \frac{1}{\|\mathbf{x}\|_p^{p-1}} \left( \sum_i x_i |x_i|^{p-2} \frac{\partial x_i}{\partial x_k} \right) \\ 565 \\ 566 \\ 567 \\ 568 \\ 569 \\ 570 \\ 571 \end{aligned}$$

572    Using this result we can obtain  $\nabla \|\nabla f(\mathbf{x})\|_p$  by chain rule. Note that  $\nabla f \triangleq \nabla f(\mathbf{x})$  in the following  
 573    expressions:  
 574

$$\begin{aligned} 575 \quad \nabla \|\nabla f\|_p &= \frac{1}{\|\nabla f\|_p^{p-1}} \begin{bmatrix} \sum_i \frac{\partial f}{\partial x_i} \left| \frac{\partial f}{\partial x_i} \right|^{p-2} \frac{\partial^2 f}{\partial x_i \partial x_1} \\ \vdots \\ \sum_i \frac{\partial f}{\partial x_i} \left| \frac{\partial f}{\partial x_i} \right|^{p-2} \frac{\partial^2 f}{\partial x_i \partial x_n} \end{bmatrix} \\ 576 \\ 577 \\ 578 \\ 579 \\ 580 \quad &= \frac{1}{\|\nabla f\|_p^{p-1}} \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \vdots & \vdots & & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \frac{\partial^2 f}{\partial x_2 \partial x_n} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \begin{bmatrix} \frac{\partial f}{\partial x_1} \left| \frac{\partial f}{\partial x_1} \right|^{p-2} \\ \vdots \\ \frac{\partial f}{\partial x_n} \left| \frac{\partial f}{\partial x_n} \right|^{p-2} \end{bmatrix} \\ 581 \\ 582 \\ 583 \\ 584 \\ 585 \quad \implies \nabla \|\nabla f(\mathbf{x})\|_p &= \frac{1}{\|\nabla f\|_p^{p-1}} H(\mathbf{x}) \left( \nabla f(\mathbf{x}) \odot \text{abs}(\nabla f(\mathbf{x}))^{p-2} \right) \end{aligned} \tag{11}$$

588    We use the expression obtained in Equation 11 in computing derivative of  $g_p(\mathbf{x})$  and  $\hat{g}_p(\mathbf{x})$  (through  
 589    chain-rule) as follows:  
 590

$$\begin{aligned} 591 \quad \nabla g_p(\mathbf{x}) &= \nabla f(\mathbf{x}) + \lambda \nabla \|\nabla f(\mathbf{x})\|_p, \text{ and} \\ 592 \quad \nabla \hat{g}_p(\mathbf{x}) &= \nabla f(\mathbf{x}) + \lambda p \|\nabla f(\mathbf{x})\|_p^{p-1} \nabla \|\nabla f(\mathbf{x})\|_p. \end{aligned}$$

593    which on expansion proves the lemma.  $\square$

594     **Corollary 1** (CGD with  $L^1$ -norm). Consider modified loss functions  $g_1(\mathbf{x})$  and  $\hat{g}_1(\mathbf{x})$  (as stated in  
 595     Equations 6 and 7) that use  $L^1$ -norm gradient penalty. Then,  
 596

$$597 \quad \nabla g_1(\mathbf{x}) = \nabla \hat{g}_1(\mathbf{x}) = \nabla f(\mathbf{x}) + \lambda H(\mathbf{x}) \operatorname{sign}(\nabla f(\mathbf{x}))$$

598     **Corollary 2** (CGD with  $L^2$ -norm). Similarly,  $g_2(\mathbf{x})$  and  $\hat{g}_2(\mathbf{x})$  with  $L^2$ -norm gradient penalty have  
 599     the following gradients:  
 600

$$601 \quad \nabla g_2(\mathbf{x}) = \nabla f(\mathbf{x}) + \frac{\lambda}{\|\nabla f(\mathbf{x})\|} H(\mathbf{x}) \nabla f(\mathbf{x})$$

$$602 \quad \nabla \hat{g}_2(\mathbf{x}) = \nabla f(\mathbf{x}) + 2\lambda H(\mathbf{x}) \nabla f(\mathbf{x})$$

## 605     B EXPERIMENTAL RESULTS

607     The functions used for the experiments are given as follows:  
 608

- Quadratic Function (dimensions  $n$ )

$$610 \quad f(\mathbf{x}) = \frac{1}{2} \mathbf{x}^T A \mathbf{x} - \mathbf{b}^T \mathbf{x} + c$$

613     where  $A$  is a positive-definite matrix,  $\mathbf{b}$  is any vector.  $\mathbf{x}^* = A^{-1}\mathbf{b}$  and  $f^* = f(\mathbf{x}^*)$ . In our  
 614     experiments, we generate  $A$  randomly as follows:

$$615 \quad A = \frac{1}{2}(U + U^T) + nI_n$$

618     where  $U \sim \mathcal{U}[0, 1]^{n \times n}$ , a matrix where all values are uniformly sampled from the interval  
 619      $[0, 1]$ . We take  $\mathbf{b} = [1, 2, \dots, n]^T$  and  $c = 0.5$ .

- Rotated hyper-ellipsoid Function (dimensions  $n$ )

$$622 \quad f(\mathbf{x}) = \sum_{i=1}^n \sum_{j=1}^i x_j^2$$

625      $f^* = 0$  at  $\mathbf{x}^* = \mathbf{0}_n$ .

- Matyas Function (dimensions  $n = 2$ )

$$627 \quad f(\mathbf{x}) = 0.26(x_1^2 + x_2^2) - 0.48x_1x_2$$

629      $f^* = 0$  at  $x^* = \mathbf{0}_n$ .

- Branin Function (dimensions  $n = 2$ )

$$632 \quad f(\mathbf{x}) = \left( x_2 - \frac{5.1}{4\pi^2} x_1^2 + \frac{5}{\pi} x_1 - 6 \right)^2 + 10 \left( 1 - \frac{1}{8\pi} \right) \cos x_1 + 10$$

634      $f^* = 0.397887$  at  $\mathbf{x}^* = [-\pi, 12.275]^T, [\pi, 2.275]^T, [9.42478, 2.475]^T$ .

- Levy Function (dimensions  $n$ )

$$637 \quad f(\mathbf{x}) = \sin^2(\pi w_1) + \sum_{i=1}^{n-1} (w_i - 1)^2 (1 + 10 \sin^2(\pi w_i + 1)) + (w_n - 1)^2 (1 + \sin^2(2\pi w_n))$$

640     where  $w_i = 1 + (x_i - 1)/4$  for all  $i$ .  $f^* = 0$  at  $\mathbf{x}^* = (1, \dots, 1)$ .

- Griewank Function (dimensions  $n$ )

$$643 \quad f(\mathbf{x}) = \sum_{i=1}^n \frac{x_i^2}{4000} - \prod_{i=1}^n \cos \left( \frac{x_i}{\sqrt{i}} \right) + 1$$

646      $f^* = 0$  at  $x^* = \mathbf{0}_n$ .