Explicit Gradient Learning for Black-Box Optimization

Elad Sarafian* [1] **Mor Sinay*** [1] **Yoram Louzoun** [1] **Noa Agmon** [1] **Sarit Kraus** [1]

Abstract

Black-Box Optimization (BBO) methods can find optimal policies for systems that interact with complex environments with no analytical representation. As such, they are of interest in many Artificial Intelligence (AI) domains. Yet classical BBO methods fall short in high-dimensional non-convex problems. They are thus often overlooked in real-world AI tasks. Here we present a BBO method, termed Explicit Gradient Learning (EGL), that is designed to optimize high-dimensional ill-behaved functions. We derive EGL by finding weak spots in methods that fit the objective function with a parametric Neural Network (NN) model and obtain the gradient signal by calculating the parametric gradient. Instead of fitting the function, EGL trains a NN to estimate the objective gradient directly. We prove the convergence of EGL to a stationary point and its robustness in the optimization of integrable functions. We evaluate EGL and achieve state-of-the-art results in two challenging problems: (1) the COCO test suite against an assortment of standard BBO methods; and (2) in a high-dimensional non-convex image generation task.

1. Introduction

Optimization problems are prevalent in many artificial intelligence applications, from search-and-rescue optimal deployment (Zhen et al., 2014) to triage policy in emergency rooms (Rosemarin et al., 2019) to hyperparameter tuning in machine learning (Bardenet et al., 2013). In these tasks, the objective is to find a policy that minimizes a cost or maximizes a reward. Evaluating the cost of a single policy is a complicated and often costly process that usually has no analytical representation, e.g., due to interaction with real-world physics or numerical simulation. *Black-Box Optimization* (BBO) algorithms (Audet & Hare, 2017; Golovin et al., 2017) are designed to solve such problems, when the analytical formulation is missing, by repeatedly querying the Black-Box function and searching for an optimal solution while minimizing the number of queries (budget).

Related Work BBO problems have been studied in multiple fields with diverse approaches. Many works investigated *derivative–free* methods (Rios & Sahinidis, 2013), from the classic Nelder–Mead algorithm (Nelder & Mead, 1965) and Powell's method (Powell, 1964) to more recent evolutionary algorithms such as CMA-ES (Hansen, 2006). Another line of research is *derivative-based* algorithms, which first approximate the gradient and then apply line-search methods such as the Conjugate Gradient (CG) Method (Shewchuk et al., 1994) and Quasi-Newton Methods, e.g. BFGS (Nocedal & Wright, 2006). Other model-based methods such as SLSQP (Bonnans et al., 2006) and COBYLA (Powell, 2007) iteratively solve quadratic or linear approximations of the objective function. Some variants apply *trust-region* methods and iteratively find an optimum within a trusted subset of the domain (Conn et al., 2009; Chen et al., 2018). Another line of research is more focused on stochastic discrete problems, e.g. Bayesian methods (Snoek et al., 2015), and multi-armed bandit problems (Flaxman et al., 2004).

More recent works have studied the applications of NNs in BBO algorithms. (Mania et al., 2018; Vemula et al., 2019) showed that Markov Decision Processes could be solved by applying random search optimization over the weights of a parameterized policy. (Sener & Koltun, 2020) suggested learning an intermediate low dimensional manifold with a NN to reduce the complexity of the random search. (Maheswaranathan et al., 2018) suggested learning the objective and using the parametric gradient with respect to the inputs (Lillicrap et al., 2015) to guide a random search. (Saremi, 2019) studied NN architectures for learning gradients.

Our contribution In this paper, we suggest a new derivative-based algorithm, *Explicit Gradient Learning* (EGL), that learns a surrogate function for the gradient by averaging the numerical directional derivatives over small volumes bounded by ε radius. We control the accuracy of our model by controlling the ε radius parameter. We then use trust-regions and dynamic scaling of the objective function

*Equal contribution. [1]Department of Computer Science, Bar-Ilan University, Israel. Correspondence to: Elad Sarafian, Mor Sinay <elad.sarafian@gmail.com, mor.sinay@gmail.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

to fine-tune the convergence process to the optimal solution. This results in a theoretically guaranteed convergence of EGL to a stationary point. We compared the performance of EGL to eight different BBO algorithms in rigorous simulations in the COCO test suite (Hansen et al., 2019) and show that EGL outperforms all others in terms of final accuracy. EGL was further evaluated in a high-dimensional non-convex domain, involving searching in the latent space of a Generative Adversarial Network (GAN) (Pan et al., 2019) to generate an image with specific characteristics. EGL again outperformed existing algorithms in terms of accuracy and appearance, where other BBO methods failed to converge to a solution in a reasonable amount of time.

The paper is organized as follows: BBO background and motivation for the EGL algorithm are presented in Sections 2 and 3. The detailed description of the EGL algorithm and its theoretical analysis are shown in Section 4. The empirical evaluation of EGL's performance and comparison to state-of-the-art BBO algorithms are described in Section 5, and we conclude in Section 6. The proofs for all of the theoretical statements are found in the Appendix.

2. Background

A function $f : \Omega \to \mathbb{R}$, $\Omega \subseteq \mathbb{R}^n$ is considered to be a Black-Box if one can evaluate $y = f(x)$ at $x \in \Omega$, but has no prior knowledge of its analytical form. A BBO algorithm seeks to find $x^* = \arg\min_{x\in\Omega} f(x)$, typically with as few evaluations as possible (Audet & Hare, 2017). Since, in general, it is not possible to converge to the optimal value with a finite number of evaluations, we define a budget C and seek to find as good a solution as possible $x^\star$ with less than C evaluations (Hansen et al., 2010).

Many BBO methods operate with a two-phase iterative algorithm: (1) search or collect data with some heuristic; and (2) update a model to obtain a new candidate solution and improve the heuristic. Traditionally, BBO methods are divided into derivative-free and derivative-based methods. The former group relies on statistical models (Balandat et al., 2019), physical models (Van Laarhoven & Aarts, 1987), or Evolutionary Strategies (Back, 1996) to define the search pattern and are not restricted to continuous domains, so they can also be applied to discrete variables. Our algorithm, EGL, falls under the latter category, which relies on a gradient estimation to determine the search direction (Bertsekas & Scientific, 2015). Formally, derivative-based methods are restricted to differentiable functions, but here we show that EGL can be applied successfully whenever the objective function is merely locally integrable.

Recently, with the Reinforcement Learning renaissance (Silver et al., 2017; Schulman et al., 2015), new algorithms have been suggested for the problem of continuous action control (e.g., robotics control). Many of these can be viewed in the context of BBO as derivative-based methods applied with NN parametric models. One of the most prominent algorithms is DDPG (Lillicrap et al., 2015). DDPG iteratively collects data with some (often naive) exploration strategy and then fits a local NN parametric model f_θ around a candidate solution x_k. To update the candidate, it approximates the gradient ∇f at x_k with the parametric gradient ∇f_θ and then applies a gradient descent step (Ruder, 2016).[1] Algorithm 1 outlines the DDPG steps in the BBO formulation. We denote it as Indirect Gradient Learning (IGL) since it does not directly learn the gradient ∇f. In the next section, we will develop arguments as to why one should learn the gradient explicitly instead of using the parametric gradient.

Algorithm 1 Indirect Gradient Learning

Input: x_0, α, C
$k = 0$
while *budget* $C > 0$ **do**
  Build Local Model:
    Collect data $\mathcal{D}_k = \{(x_k + \varepsilon n_i, y_i)\}_{i=1}^m,\ n_i \sim \mathcal{N}(0, \mathbf{I})$
    Fit a model f_{θ_k} with
    $\theta_k = \arg\min_\theta \sum_{i=1}^m |f_\theta(x_k + \varepsilon n_i) - y_i|^2$
  Gradient Descent:
    $x_{k+1} \leftarrow x_k - \alpha \nabla f_{\theta_k}(x_k)$
    $k \leftarrow k + 1$
return x_k

3. Motivation

In Algorithm 1 only the gradient information $\nabla f(x_k)$ is required to update the next x_k candidate. However, the gradient function is never learned directly, and it is only inferred from the parametric model without any clear guarantee of its veracity. Hence we seek a method that learns the gradient function ∇f explicitly. Clearly, directly learning the gradient is infeasible since the Black-Box only outputs the $f(x)$ values. Instead, our approach would be to learn a surrogate function for ∇f, termed the *mean-gradient*, by sampling pairs of observations $\{(x_i, y_i), (x_j, y_j)\}_{i,j}$ and averaging the numerical directional derivatives over small volumes. This section formally defines the mean-gradient and then formulates two arguments that motivate its use, instead of ∇f_θ. We then illustrate these arguments using 1D and 2D examples taken from the COCO test suite.

3.1. The Mean-Gradient

For any differentiable function f with a continuous gradient, the first order Taylor expression is

$$f(x + \tau) = f(x) + \nabla f(x) \cdot \tau + O(\|\tau\|^2). \quad (1)$$

[1] f_θ is differentiable. Thus, it can be differentiated with respect to its input x, as done in adversarial training (Yuan et al., 2019).

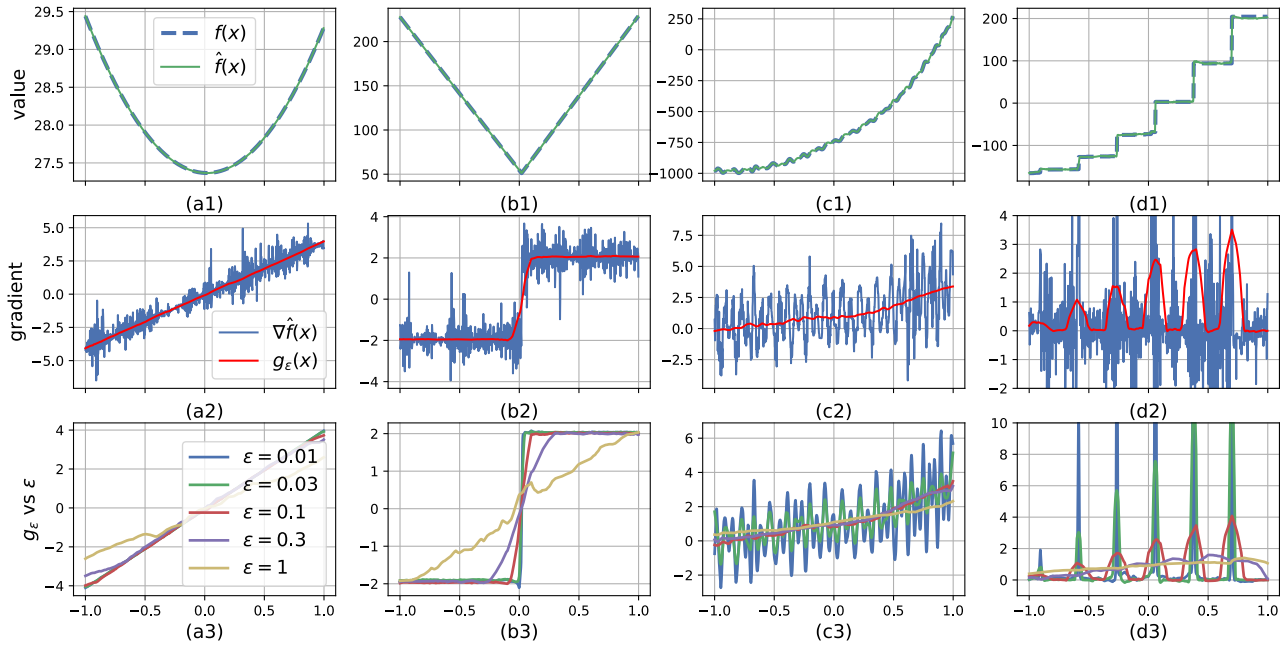

Figure 1. Comparing indirect gradient learning and explicit gradient learning for 4 typical functions: (a) parabolic; (b) piece-wise linear; (c) multiple local minima; (d) step function.

Thus, locally around x, the directional derivative satisfies $\nabla f(x) \cdot \tau \approx f(x+\tau) - f(x)$. We define the mean-gradient as the function that minimizes the Mean-Square-Error (MSE) of such approximations in a vicinity of x.

Definition 1. *The mean-gradient at x with $\varepsilon > 0$ averaging radius is*

$$g_\varepsilon(x) = \arg\min_{g\in\mathbb{R}^n} \int\limits_{V_\varepsilon(x)} |g\cdot\tau - f(x+\tau) + f(x)|^2 d\tau \quad (2)$$

where $V_\varepsilon(x) \subset \mathbb{R}^n$ is a convex subset s.t. $\|x' - x\| \le \varepsilon$ for all $x' \in V_\varepsilon(x)$ and the integral domain is over τ s.t. $x+\tau \in V_\varepsilon(x)$.

Proposition 1 (controllable accuracy)**.** *For any differentiable function f with a continuous gradient, there is $\kappa_g > 0$, so that for any $\varepsilon > 0$ the mean-gradient satisfies $\|g_\varepsilon(x) - \nabla f(x)\| \le \kappa_g \varepsilon$ for all $x \in \Omega$.*

In other words, the mean-gradient has a controllable accuracy parameter ε s.t. reducing ε improves the gradient approximation. As explained in Sec. 4.2, this property is crucial in obtaining the convergence of EGL to stationary points. Unlike the mean-gradient, the parametric gradient has no such parameter and the gradient accuracy is not directly controlled. Even for zero MSE error s.t. $f_\theta(x_i) \equiv y_i$ for all of the samples in the replay buffer (Mnih et al., 2015), there is no guarantee of the parametric gradient accuracy. On the contrary, overfitting the objective may severely hurt the gradients approximation.

Moreover, the parametric gradient can be discontinuous even when the parametric model has a Lipschitz continuous (Hansen & Jaumard, 1995) gradient. For example, a commonly used NN with ReLU activation is only piece-wise differentiable. This leads to very erratic gradients even for smooth objective functions. If the objective function is not smooth (e.g., for noise-like functions or in singular points), the gradient noise is exacerbated. On the other hand, the next proposition suggests that, due to the integral over $V_\varepsilon(x)$ that smooths the gradient, the mean gradient is smooth whenever the objective function is continuous.

Proposition 2 (continuity)**.** *If f is continuous in V and $V_\varepsilon(x) \subset V$ then g_ε is a continuous function at x.*

The mean-gradient is not necessarily continuous in discontinuity points of f. One can obtain an even smoother surrogate for ∇f by slightly modifying the mean-gradient definition

$$g^p_\varepsilon(x) = \arg\min_{g\in\mathbb{R}^n} \iint\limits_{V_\varepsilon(x)B_p(x)} |g\cdot(\tau - s) - f(\tau) + f(s)|^2 ds d\tau,$$

where the integral domains are $s \in B_p(x)$ and $\tau \in V_\varepsilon(x)$. Here, $B_p(x) \subset V_\varepsilon(x)$ is an n-ball perturbation set with $p < \varepsilon$ radius that dithers the reference point (which is fixed at x in Definition 1). We term this modified version the *perturbed mean-gradient*. Remarkably, while g^p_ε is still a controllably accurate model for Lipschitz continuous gradients, as the integrand is x independent, g^p_ε is continuous

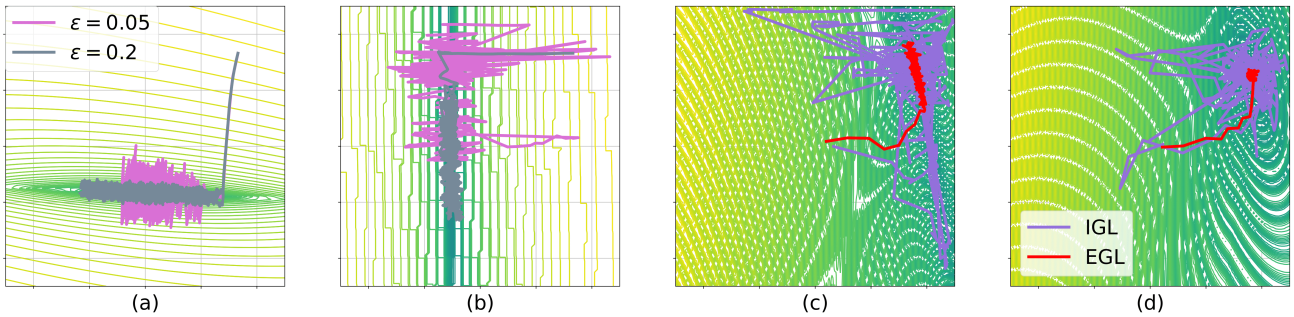

Figure 2. Visualizing explicit gradient learning with different ε for various 2D problems from COCO test suite: (a) sharp-ridge problem 194; (b) step-ellipsoid problem 97. Comparing EGL and IGL: (c) Schaffer F7 C1000 problem 255; (d) Schaffer F7 C10 problem 240.

whenever f is merely integrable. In practice, as g_ε is learned with a Lipschitz continuous model, we find that both forms are continuous for integrable functions. Nevertheless, Sec. 5 shows that g^p_ε adds a small gain to the EGL performance. Next, we demonstrate how the EGL smoothness (as opposed to ∇f_θ) leads to more stable and efficient trajectories in both continuous and discontinuous objective functions.

3.2. Illustrative Examples

To demonstrate these properties of the mean-gradient, we consider 1D [2] and 2D problems from the COCO test suite. We start by examining 4 typical 1D functions: (a) parabolic; (b) piece-wise linear; (c) multiple local minima; and (d) step function. We fit f with a NN and compare its parametric gradient to the mean-gradient, learned with another NN. The NN model is identical for both functions and is based on our spline embedding architecture (see Sec. 4.4). The results are presented in Fig. 1. The 1st row shows that the fit $\hat{f}$ is very strong s.t. the error is almost indistinguishable to the naked eye. Nevertheless, the calculated parametric gradient (2nd row) is noisy, even for smooth functions, as the NN architecture is piece-wise linear. Occasionally, there are even spikes that change the gradient sign. Traversing the surface curves with such a function is very unstable and inefficient. On the other hand, due to the smoothing parameter $\varepsilon = 0.1$ and since the NN is Lipschitz continuous, the mean-gradient is always smooth, even for singularity points and discontinuous gradients.

In the 3rd row we evaluate g_ε for different size ε parameters. We see that by setting ε sufficiently high, the gradient becomes smooth enough, so there is no problem descending over steps and multiple local minima functions. In practice, we may use this property and start the descent trajectory with a high ε. This way, the optimization process does not commit too early to a local minimum and searches for regions with lower valleys. After refining ε the process will settle into a local minimum, which in practice would be much lower than minima found around the initial point.

The next experiment (Fig. 2) is executed on 2D problems from the COCO test suite. In Fig. 2(a-b) we present the gradient-descent steps with the mean-gradient and a constant ε. In 2(a) the objective has a singular minimum, similar to the $|x|$ function's minimum (Fig.1(a)). As ε gets smaller, the gradients near the minimum get larger and there is a need to reduce the learning-rate (α) in order to converge. 2(b) presents a step function. Again, we observe that for a smaller ε the gradient has high spikes in the discontinuity points, leading to a noisier trajectory. For that purpose, the EGL algorithm decays both α and ε during the learning process (see Sec. 4 for details). This lends much smoother trajectories, as can be seen in Fig. 2(c-d). Here we executed both EGL and IGL with the same α, ε decay pattern. We observe that the IGL trajectories are noisy and inefficient since the parametric gradient error is not bounded. On the other hand, EGL smoothly travels through a ravine (Fig. 2(c)) and converges to a global minimum (Fig. 2(d)).

4. Design & Analysis

In this section, we lay out the practical EGL algorithm and analyze its asymptotic properties.

4.1. Monte-Carlo Approximation

To learn the mean-gradient, one may evaluate the integral in Eq. (2) with Monte-Carlo samples and learn a model that minimizes this term. Formally, for a model $g_\theta : \Omega \to \mathbb{R}^n$ and a dataset $\mathcal{D}_k = \{(x_i, y_i)\}_{i=1}^m$, define the loss function

$$\mathcal{L}_{k,\varepsilon}(\theta) = \sum_{i=1}^{m} \sum_{x_j \in V_\varepsilon(x_i)} |(x_j - x_i) \cdot g_\theta(x_i) - y_j + y_i|^2 \quad (3)$$

and learn $\theta^*_k = \arg\min_\theta \mathcal{L}_{k,\varepsilon}(\theta)$, e.g. with gradient descent. This formulation can be used to estimate the mean-gradient for any x. Yet, practically, in each optimization iteration, we only care about estimating it in close proximity to the

[2] Since COCO does not have built-in 1D problems; we generated 1D problems based on 2D problems with $f_{1D}(x) : f_{2D}(x, x)$

current candidate solution x_k. Therefore, we assume that the dataset $\mathcal{D}_k$ holds samples only from $V_\varepsilon(x_k)$.

The accuracy of the learned model $g_{\theta_k^*}$ heavily depends on the number and locations of the evaluation points and the specific parameterization for g_θ. Still, for f with a Lipschitz continuous gradient, i.e. $f \in \mathcal{C}^{+1}$, we can set bounds for the model accuracy in $V_\varepsilon(x_k)$ with respect to ε. For that purpose, we require a set of at least $m \geq n+1$ evaluation points in $\mathcal{D}_k$ which satisfy the following poised set definition.

Definition 2 (poised set for regression)**.** *Let $\mathcal{D}_k = \{(x_i, y_i)\}_1^m$, $m \geq n+1$ s.t. $x_i \in V_\varepsilon(x_k)$ for all i. Define the matrix $\tilde{X}_i \in \mathbb{M}^{m \times n}$ s.t. the j-th row is $x_i - x_j$. Now define $\tilde{X} = \left(\tilde{X}_1^T \cdots \tilde{X}_m^T \right)^T$. The set $\mathcal{D}_k$ is a poised set for regression in x_k if the matrix $\tilde{X}$ has rank n.*

Intuitively, a set is poised if its difference vectors $x_i - x_j$ span $\mathbb{R}^n$. For the poised set, and a constant parameterization, the solution of Eq. (3) is unique and it is equal to the Least-Squares (LS) minimizer. If f has a Lipschitz continuous gradient, then, with an admissible parametric model, the error between $g_\theta(x)$ and $\nabla f(x)$ can be proportional to ε. We formalize this in the following theorem and corollary.

Theorem 1. *Let $\mathcal{D}_k$ be a poised set in $V_\varepsilon(x_k)$. The regression problem*

$$g^{MSE} = \arg\min_g \sum_{i,j \in \mathcal{D}_k} |(x_j - x_i) \cdot g - y_j + y_i|^2 \quad (4)$$

has the unique solution $g^{MSE} = (\tilde{X}^T \tilde{X})^{-1} \tilde{X}^T \delta$, where $\delta \in \mathbb{R}^{m^2}$ s.t. $\delta_{i \cdot (m-1)+j} = y_j - y_i$. Further, if $f \in \mathcal{C}^{1+}$ and $g_\theta \in \mathcal{C}^0$ is a parameterization with (equal or) lower regression loss than g^{MSE}, the following holds for all $x \in V_\varepsilon(x_k)$:

$$\|\nabla f(x) - g_\theta(x)\| \leq \kappa_g \varepsilon \quad (5)$$

Corollary 1. *For the $\mathcal{D}_k$ poised set, any Lipschitz continuous parameterization of the form $g_\theta(x) = F(Wx) + b$ is a controllably accurate model (Audet & Hare, 2017) in $V_\varepsilon(x_k)$ for the optimal set of parameters θ_k^*.*

This is obvious as we can simply set $W = 0$ and $b = g^{MSE}$. In this work, we are interested in using NNs which are much stronger parameterizations than constant models. If the NN satisfies the Lipschitz continuity property (e.g., with spectral normalization) and has at least a biased output layer, then its optimal set of parameters θ_k^* has lower regression loss than g^{MSE} and it is therefore a controllably accurate model.[3]

Finally, to incorporate learning of the perturbed mean gradient, we slightly modify Eq. (3). Note that we cannot directly dither the reference point x_i as we would need to collect more samples. Instead, we may dither the g_θ argument by evaluating it in $\bar{x}_{ir} = x_i + n_r$ where n_r is uniformly sampled in an n-ball with radius p. Thus, the perturbed loss function for small p s.t. $p \ll \varepsilon$ is

$$\mathcal{L}_{k,\varepsilon,p}(\theta) = \sum_{i,j \in \mathcal{D}_k, n_r} |(x_j - x_i) \cdot g_\theta(\bar{x}_{ir}) - y_j + y_i|^2 \quad (6)$$

4.2. Asymptotic Analysis

In this part, we assume that the function f has a Lipschitz continuous gradient s.t. $\|\nabla f(x) - \nabla f(x')\| \leq \kappa_f \|x - x'\|$. In this case, the classical gradient descent theorem states that the update $x_{k+1} = x_k - \alpha \nabla f(x_k)$ with learning parameter $\alpha \leq \frac{1}{\kappa_f}$ converges to a stationary point x^* s.t. $\|f(x_k)\| \to 0$ for $k \to \infty$ (Nesterov, 2013; Lee et al., 2016).

EGL descends over a surrogate function of the gradient that contains some amount of error. Far from x^*, where $\|\nabla f\|$ is large, the error in g_ε is small enough s.t. every new candidate improves the solution, i.e. $f(x_{k+1}) \leq f(x_k)$. If the stationary point x^* is locally convex, then, as x_k gets closer to x^*, the gradient $\|\nabla f\|$ decreases, and eventually the error, i.e. $g_\varepsilon(x_k) - \nabla f(x_k)$, becomes so significant that improvement is no longer guaranteed. Nevertheless, our analysis shows that a proper choice of ε_k yields monotonic decreasing steps that converge to a stationary point.

Theorem 2. *Let $f : \Omega \to \mathbb{R}$ have a Lipschitz continuous gradient and a Lipschitz constant κ_f. Suppose a controllable mean-gradient model g_ε with error constant κ_g, the gradient descent iteration $x_{k+1} = x_k - \alpha g_\varepsilon(x_k)$ with α s.t. $\frac{5\varepsilon}{\|\nabla f(x_k)\|} \leq \alpha \leq \min(\frac{1}{\kappa_g}, \frac{1}{\kappa_f})$ guarantees a monotonically decreasing step s.t. $f(x_{k+1}) \leq f(x_k) - 2.25\frac{\varepsilon^2}{\alpha}$.*

Corollary 2. *If Theorem 2 is satisfied for all k, the gradient descent iteration converges to a stationary point x^* s.t. $\frac{1}{K}\sum_{k=1}^K \|f(x_k)\|^2 \leq \frac{12|f(x_0) - f(x^*)|}{\alpha_K K}$.*

Utilizing Theorem 2 we can design an algorithm that converges to a stationary point. For that purpose we must make sure that the learning rate abides the requirement $\alpha \leq \min(\frac{1}{\kappa_g}, \frac{1}{\kappa_f})$. Since this factor cannot be easily estimated, a practical solution is to decay α during the optimization process. However, to converge, the ratio $\frac{\varepsilon}{\alpha}$ must also decay to zero. This means that ε must decay to zero faster than α. Algorithm 2 provides both of these conditions, so it is guaranteed to converge to a local minimum for $\bar{\varepsilon} \to 0$.

This analysis assumes a Lipschitz continuous gradient, but since EGL is also designed for non-smooth functions, our practical algorithm alleviates the requirement for strictly monotonically decreasing steps. Instead, it calculates a running mean over the last candidates to determine when to decrease α and ε. The complete practical EGL algorithm is found in the Appendix Sec. E. It also includes input and output mapping and scaling, as described in the next section.

[3] Provided that the optimization process recovers θ_k^*, which is not necessarily true in practice.

Algorithm 2 Convergent EGL

Input: $x_0, \alpha, \varepsilon, \gamma_\alpha < 1, \gamma_\varepsilon < 1, \bar{\varepsilon}$
$k = 0$
while $\varepsilon \leq \bar{\varepsilon}$ **do**
  Build Model:
    Collect data $\{(x_i, y_i)\}_1^m, x_i \in V_\varepsilon(x_k)$
    Learn a local model $g_\varepsilon(x_k)$
  Gradient Descent:
    $x_{k+1} \leftarrow x_k - \alpha g_\varepsilon(x_k)$
    if $f(x_{k+1}) > f(x_k) - 2.25\frac{\varepsilon^2}{\alpha}$ **then**
      $\alpha \leftarrow \gamma_\alpha \alpha$
      $\varepsilon \leftarrow \gamma_\alpha \gamma_\varepsilon \varepsilon$
  $k \leftarrow k + 1$
return x_k

4.3. Dynamic Mappings

Unlike supervised learning with a constant dataset, BBO is a dynamic problem. The input and output statistics change over time as the optimization progresses. There are several sources for this drift. First, traversing via g_ε changes the input's first moment by updating the center of the samples x_i and the output's first moment by collecting smaller costs y_i. At the same time, squeezing ε and α over time decreases the second moment statistics by reducing the variety in $\{(x_i, y_i)\}$. NNs are sensitive to such distribution changes (Brownlee, 2018) and require tweaking hyperparameters such as learning rate and initial weight distribution to maintain high performance. Default numbers (e.g. learning rate of 10^{-3}) usually work best when the input data is normalized. To regulate the statistics over the entire optimization process, we apply a method of double dynamic mappings for both input and output values.

From the input perspective, our dynamic mapping resembles *Trust-Region* methods (Conn et al., 2000; Nocedal & Wright, 2006). Instead of searching for x^* in the entire Ω domain by reducing ε and α over time, we fix ε and α and search for $x_j^\star$ in a sub-region Ω_j. After finding the best candidate solution in this sub-region we shrink Ω_j by a factor of $\gamma_\alpha > 0$ and ε by a factor of $\gamma_\varepsilon > 0$. To keep the input statistics regulated, we maintain a bijective mapping $h_j : \Omega_j \to \mathbb{R}^n$: that scales the x values to an unconstrained domain $\tilde{x}$ with approximately constant first and second moments.

In this work, Ω is assumed to be a rectangular box that denotes the upper and lower bounds for each entry in x. Thus we consider element-wise bijective mappings of the form $h_j(x) = (h_j^1(x^1), ..., h_j^n(x^n))$ and each new sub-region, $\Omega_{j+1} \subset \Omega_j$, is a smaller rectangular box centered at $x_j^\star$. In the unconstrained domain $\tilde{x}$, we can use the initial learning rate, yet, due to the squeezing factor, the effective learning rate is decayed by the γ_α factor. Equivalently, the effective accuracy parameter ε is reduced by a factor of $\gamma_\alpha \times \gamma_\varepsilon$. In other words, we use the same NN parameters to learn a zoomed-in problem of the original objective function.

In order to regulate the output statistics, we define a scalar, monotonically increasing, invertible mapping $r_k(y)$ which maps the y_i samples in the dataset $\mathcal{D}_k$ to approximately constant statistics. Since traversing with g_ε can significantly change the statistics even in the same sub-region, r_k must be dynamic and cannot be held fixed for the entire j sub-problem. Combining both input and output mappings, we obtain a modified set of samples $\{(\tilde{x}_i, \tilde{y}_i)\}_i = \{(h_j(x_i), r_k(y_i))\}_i$, so effectively we learn a modified mean-gradient, denoted as $\tilde{g}_{\varepsilon_{jk}}$, of a compressed and scaled function $\tilde{y} = \tilde{f}_{jk}(\tilde{x}) = r_k \circ f \circ h_j^{-1}(\tilde{x})$.

If both input and output mappings are linear, then the true mean-gradient is proportional to the modified mean-gradient

$$g_\varepsilon(x) = \left(\frac{\partial r_k}{\partial y}\right)^{-1} \nabla h_j(x) \odot \tilde{g}_{\varepsilon_{jk}}(\tilde{x}) \tag{7}$$

Even when the mappings are approximately linear inside $V_{\varepsilon_{jk}}(\tilde{x}_k)$, $g_\varepsilon(x_k)$ can be estimated according to Eq. (7). Hence, to maintain a controllably accurate model, we seek mappings that preserve the linearity as much as possible. On the other hand, strictly linear mappings may be insufficient since they are susceptible to outliers. After experimenting with several functions, we found a sweet spot: a composition of a linear mapping followed by a squash function that compresses only the outliers (see details in Appendix Sec. D). With such mappings, we make sure both that the model is controllably accurate and that the learning hyperparameters are adequate for the entire optimization process.

4.4. Spline Embedding

Many black-box applications, including the experiments in Sec. 5, have no clear inductive bias which can be exploited, as e.g., CNNs are suitable for natural images (Cohen & Shashua, 2016). In such cases, it is common to use feed-forward NNs. However, we found out that the quality of fitting objectives and gradients with such NN is unsatisfactory, especially for low dimensional problems. One method to augment the input layer is through learnable embeddings (Zhang et al., 2016). However, usually, embeddings are applied for categorical input and they do not preserve order. Since our input domain is continuous, we wish to design Lipschitz continuous learnable embeddings $s_\theta(x)$ s.t. for two inputs x_1 and x_2 the calculated embedding satisfies $\|s(x_1) - s(x_2)\| \leq \kappa_s \|x_1 - x_2\|$ for some $\kappa_s > 0$.

We chose to do so by learning a set of one-dimensional splines (Reinsch, 1967). A spline is a piece-wise polynomial defined on a set of disjoint intervals with smoothness conditions in the intersection points, denoted as knots. Learnable splines were also suggested in (Fey et al., 2018) for the problem of learning over irregular grids. We used piece-wise

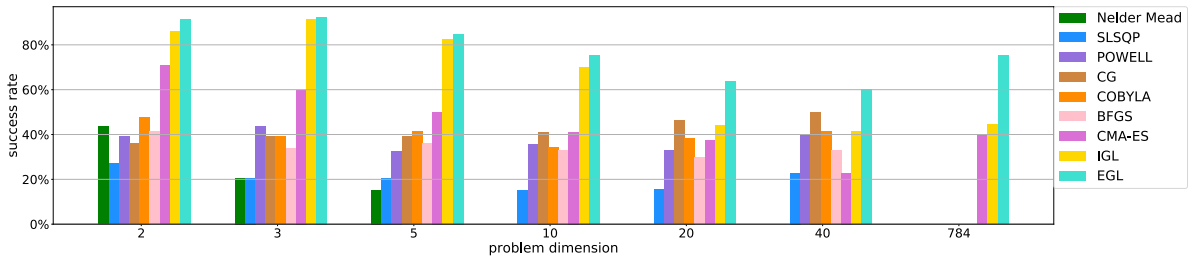

Figure 3. Comparing the success rate of EGL and other BBO algorithms for a budget $C = 150 \cdot 10^3$.

linear splines of the form

$$s_\theta(x) = \frac{\theta_i}{h_i}(t_{i+1} - x) + \frac{\theta_{i+1}}{h_i}(x - t_i) \tag{8}$$

where θ has $k+1$ elements (k is the number of knots), t_i is the position of the i-th knot and $h_i = t_{i+1} - t_i$. Instead of fitting θ to the data as usually done in classical spline applications, we learned these parameters as part of the optimization process, similar to categorical embeddings. Each entry in the input vector was expanded by a paramterization of several one-dimensional splines. We found this architecture particularly suitable for modeling complex functions defined over unstructured input domain. Please refer to the Appendix Sec. C for a full description and an empirical evaluation of fitting objective functions with spline-net.

5. Empirical Evaluation

In this section we describe the rigorous empirical analysis of EGL against various BBO methods in the COCO test suite and in a search task over the latent space of a GAN model. The code is available at `http://github.com/MorSinay/BBO`.

5.1. The COCO test suite

We tested EGL on the COCO test suite, a platform for systematic comparison of real-parameter global optimizers. COCO provides Black-Box functions in several dimensions (2,3,5,10,20,40), where each dimension comprises 360 distinct problems. To test higher dimensions, we created an extra set of 784D problems by composing a pre-trained encoder (Kingma & Welling, 2013) of FashionMnist images, each with 784 pixels (Xiao et al., 2017), with a 10D COCO problem as follows: $f_{784D}(x) : f_{10D}(Encoder(x))$.

We compared EGL with seven baselines, implemented on Scipy and Cma Python packages: Nedler Mead, SLSQP, POWELL, CG, COBYLA, BFGS, CMA-ES and with our IGL implementation based on the DDPG approach. For the 784D problems we evaluated only CG, CMA-ES and IGL which yielded results in a reasonable amount of time. We examined two network models. To compare against other baselines (Figues 3, 4 and 5(a)), we used our spline embedding model (details in Appendix Sec. C). Since spline-net is more time consuming, for the ablation tests in Fig. 5(b-d), we used a lighter Fully Connected (FC) net. For additional information, including a hyperparameters list, refer to Appendix Sec. F.

Fig. 3 presents the success rate of each algorithm with respect to the dimension number and for a budget of $C = 150 \cdot 10^3$. A single test-run is considered successful if: (1) $y_{best} - y^* \leq 1$; and (2) $\frac{y_{best} - y^*}{y_0 - y^*} \leq 10^{-2}$. Here, y_0 is the initial score $f(x_0)$, $y_{best} = \min y_k$ is the best-observed value for that run, and y^* is the minimal value obtained from all the baselines' test-runs. This definition guarantees that a run is successful only if its best-observed value is near y^*, both in terms of absolute distance and in terms of relative improvement with respect to the initial score. The results show that EGL outperforms all other baselines for any dimension. Most importantly, as the dimension number increases, the performance gap between EGL and all other baselines grows larger.

In Fig. 4 we present two performance profiles (Dolan & Moré, 2002) for the 40D and 10D problem sets: (1) Time-To-Solution (TTS); and (2) Best Observed (BO). TTS measures the percentage of solved problems with respect to the time step. We find that both EGL and IGL have a cold start behavior. This is a result of our design choice to start the training with a warm-up phase where we evaluate 384 points around x_0 and train the networks before executing the first gradient descent step. However, as the training process progresses, EGL outperforms all other methods and peaks at $C = 150 \cdot 10^3$ with the success rate of Fig. 3. The BO profile measures the percentage of problems whose best-observed value after $150 \cdot 10^3$ time steps is better than $\Delta y \in [y_0, y^*]$. We find that for all of the possible thresholds Δy, EGL solves more problems than any other method.

Fig. 5 visualizes the learning process of each method. To do that, we first calculate a scaled distance between the best value at time t and the optimal value $\Delta y^t_{best} = \frac{\min_{k \leq t} y_k - y^*}{y_0 - y^*}$. We then average this number for each t, over all runs in the same dimension problem set. This distance

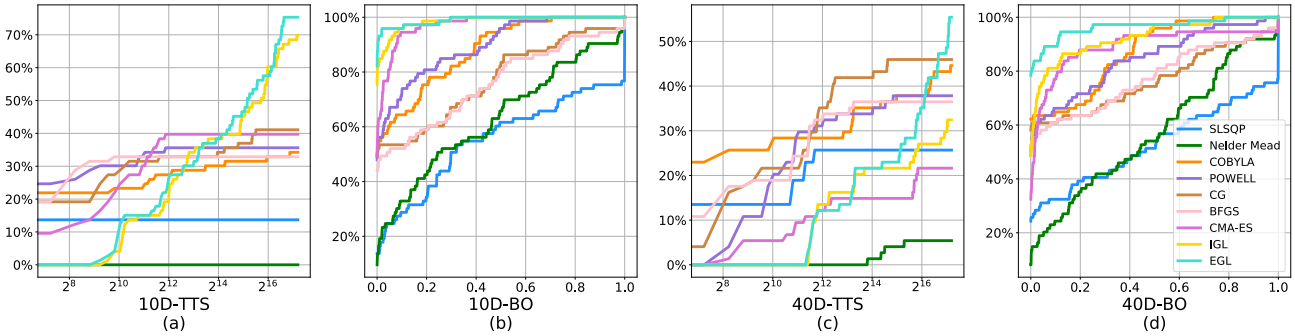

Figure 4. Performance Profile for 10D and 40D: Time-To-Solution (TTS) and Best-Observed (BO) after $150 \cdot 10^3$ steps. In BO the x-axis linearly maps $[y_0, y^*]$ to $[0, 1]$.

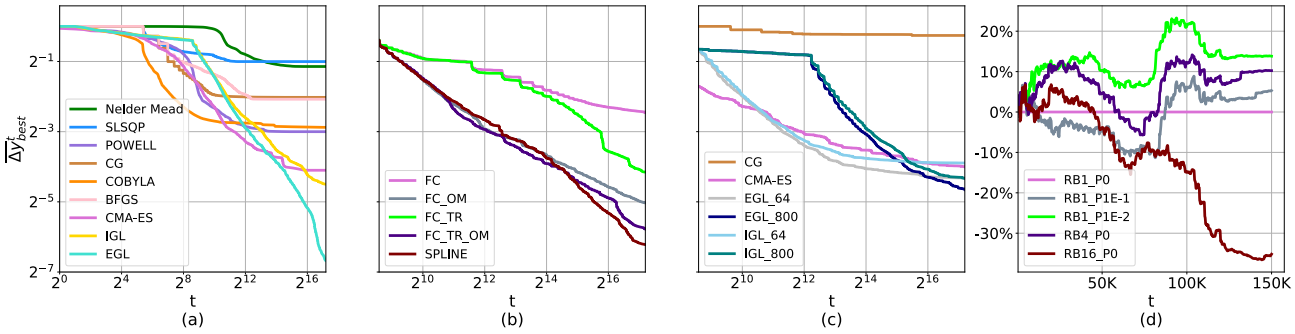

Figure 5. The scaled distance Δy^t_{best} as a function of $t \in [1, .., C]$ for: (a) EGL and baselines on 40D, (b) trust-region and output mapping ablation test, (c) EGL with different m samples and baselines on 784D, (d) the perturbed mean-gradient and long replay buffer on 40D.

is scaled to $[0, 1]$ and the results are presented on a log-log scale. Fig. 5(a) presents $\overline{\Delta y}^t_{best}$ on the 40D set. The elbow pattern in step 384 marks the switch from x_0 to x_1. Unlike the baselines that settle on a local minimum, EGL monotonically decreases during the entire optimization process. It overtakes the best baseline (CMA-ES) after $\sim 10^4$ steps.

Fig. 5(b) demonstrates the advantage of output-mapping (OM) and trust-region (TR). Here, we compared an FC net, trained with OM and TR (`FC_TR_OM`) against the variations `FC`, `FC_TR` and `FC_OM`. The results show a clear advantage to using both OM and TR, yet, while OM is crucial for the entire optimization process, the TR advantage materializes only near the minimal value. In addition, Fig. 5(b) manifests the gain of the spline-net (`SPLINE`) on top of `FC_TR_OM`.

The next experiment is executed on the high dimensional 784D problem set. In Fig. 5(c) we compare the performance of different numbers of exploration points $m = \{64, 800\}$ against CMA-ES, CG and the IGL baselines. While Theorem 1 guarantees a controllably accurate model when sampling $m \geq n+1$ points, remarkably, EGL generalized to an outstanding performance and outperformed all other baselines even for $m = 64 \ll n$. Nevertheless, as expected, more exploration points converged to a better final value. In practice, the choice of m should correspond to the allocated budget size. More exploration around each candidate comes with the cost of fewer gradient descent steps; thus, for low budgets, one should typically choose a small m.

Next, we tested the gain of two possible modifications to EGL: (1) perturbations with g^p_ε; and (2) training g_θ with a larger replay buffer (RB) with exploration points from the last L candidates. These tests were all executed with the same random seed. Fig. 5(d) presents the performance gain $(1 - \overline{\Delta y}^t_{best}/\overline{\Delta y}^t_{RB1\_P0})$ as a function of t for several different runs. Small perturbations $p = 0.01\varepsilon$ improved the performance (14%), possibly since they also regulate the NN training, yet, too large perturbations of $p = 0.1\varepsilon$ yielded inconsistent results. We also observed that a too long RB of $L = 16$ largely hurt the performance, probably since it adds high values to the RB which leads to a more compressed output mapping. However, moderate RB of $L = 4$ had a positive impact (10%), probably since more exploration points near the current candidate reduce the controllable accuracy factor κ_g and thus the accuracy of g_ε improves.

5.2. Searching the latent space of generative models

To examine EGL in a high-dimensional, complex, non-convex and noisy domain, we experimented with the task of searching the latent space of an image generative model

Figure 6. Searching latent space of generative models with EGL and IGL. Note that *the target image is not revealed to the optimizer*, only the face attributes and landmark points. The left-hand number is the average minimal value for each algorithm over 64 different problems.

(Volz et al., 2018). Generative models learn to map between a latent predefined distribution z to a complex real-world distribution x (e.g. images or audio). Given a trained Black-Box generator, while it is easy to sample from the distribution of x by sampling from z, it is not straightforward to generate an image with some desired characteristics. For that purpose, one may apply a BBO to search the latent space for a hidden representation z^* that generates an image with the desired traits x^*. Here, we used a face generative model and optimized z^* to generate an image with a required set of face attributes, landmark points and quality.

We trained a generator & discriminator for the CelebA dataset (Liu et al., 2015) based on the BigGAN (Brock et al., 2018) architecture and a classifier for the CelebA attributes. For the face landmark points, we used a pre-trained model (Kazemi & Sullivan, 2014). The BBO was trained to minimize the following objective:

$$f_{al}(z) = \lambda_a \mathcal{L}_a(G(z)) + \lambda_l \mathcal{L}_l(G(z)) + \lambda_g \tanh(D(G(z)))$$

Where: (1) $\mathcal{L}_a$ is the Cross-Entropy loss between the generated face attributes as measured by the classifier and the desired set of attributes a; (2) $\mathcal{L}_l$ is the MSE between the generated landmark points and the desired set of landmarks l ; and (3) $D(G(z))$ is the discriminator output, positive for low-quality images and negative for high-equality images.

Since each evaluation of z is costly, we limited the budget C to only 10^4 evaluations and the number of exploration points in each step was only $m = 32 \ll 512$. Such m violates the requirement in Theorem 1. However, we found that *in practice*, for finite budget and high dimensions, it is better to take more gradient steps with a less accurate gradient. To increase the sample efficiency we found that instead of sampling points in an n-ball around the candidate x_k, it is better to sample points in a cone with an apex at x_k and a vector equal to the mean-gradient estimation at x_k, i.e. $g(x_k)$. We term this exploration strategy as gradient-guided-exploration. For further details, please refer to the Appendix, Sec. H.

Due to the high-dimensional problem, classic methods such as CG and even CMA-ES fail to generate satisfying faces (see Appendix Sec. G for sample images). In Fig. 6 we compare the images generated by EGL and IGL. Generally, the quality of the results depends on the image target style. Some face traits which are more frequent in the CelebA dataset lead to a better face quality. Some faces, specifically with attributes such as a beard or a hat, are much harder to find, probably due to the suboptimality of the generator, which usually reduces the variety of images found in the dataset (Bau et al., 2019). Nevertheless, the results show that EGL produces better images, both visually and according to the final cost value. We observed that for hard targets, IGL frequently fails to find any plausible solutions while EGL yields non-perfect yet much more satisfactory candidates.

6. Conclusions

We presented EGL, a derivative-based BBO algorithm that achieves state-of-the-art results on a wide range of optimization problems. The essence of its success is a learnable function that estimates the mean gradient with a controllable smoothness factor. Starting with a high smoothness factor, let EGL find global areas in the function with low valleys. Gradually decreasing it lets EGL converge to a local minimum. The concept of EGL can be generalized to other related fields, such as sequential decision-making problems (i.e. Reinforcement Learning), by directly learning the gradient of the Q-function. We also demonstrated the use of EGL in an applicative high-dimensional Black-Box problem, searching the latent space of generative models.

Acknowledgements

This work was supported in part by the Ministry of Science & Technology, Israel.

References

Audet, C. and Hare, W. *Derivative-free and blackbox optimization*. Springer, 2017.

Back, T. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996.

Balandat, M., Karrer, B., Jiang, D. R., Daulton, S., Letham, B., Wilson, A. G., and Bakshy, E. BoTorch: Programmable Bayesian Optimization in PyTorch. *arxiv e-prints*, 2019. URL http://arxiv.org/abs/1910.06403.

Bardenet, R., Brendel, M., Kégl, B., and Sebag, M. Collaborative hyperparameter tuning. In *International conference on machine learning*, pp. 199–207, 2013.

Bau, D., Zhu, J.-Y., Wulff, J., Peebles, W., Strobelt, H., Zhou, B., and Torralba, A. Seeing what a gan cannot generate. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4502–4511, 2019.

Bertsekas, D. P. and Scientific, A. *Convex optimization algorithms*. Athena Scientific Belmont, 2015.

Bonnans, J.-F., Gilbert, J. C., Lemaréchal, C., and Sagastizábal, C. A. *Numerical optimization: theoretical and practical aspects*. Springer Science & Business Media, 2006.

Brock, A., Donahue, J., and Simonyan, K. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

Brownlee, J. *Better Deep Learning: Train Faster, Reduce Overfitting, and Make Better Predictions*. Machine Learning Mastery, 2018.

Chen, R., Menickelly, M., and Scheinberg, K. Stochastic optimization using a trust-region method and random models. *Mathematical Programming*, 169(2):447–487, 2018.

Cohen, N. and Shashua, A. Inductive bias of deep convolutional networks through pooling geometry. *arXiv preprint arXiv:1605.06743*, 2016.

Conn, A. R., Gould, N. I., and Toint, P. L. *Trust region methods*. SIAM, 2000.

Conn, A. R., Scheinberg, K., and Vicente, L. N. *Introduction to derivative-free optimization*, volume 8. Siam, 2009.

Dolan, E. D. and Moré, J. J. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

Fey, M., Eric Lenssen, J., Weichert, F., and Müller, H. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 869–877, 2018.

Flaxman, A. D., Kalai, A. T., and McMahan, H. B. Online convex optimization in the bandit setting: gradient descent without a gradient. *arXiv preprint cs/0408007*, 2004.

Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J., and Sculley, D. Google vizier: A service for black-box optimization. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1487–1495, 2017.

Hansen, N. The cma evolution strategy: a comparing review. In *Towards a new evolutionary computation*, pp. 75–102. Springer, 2006.

Hansen, N., Auger, A., Ros, R., Finck, S., and Pošík, P. Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009. In *Proceedings of the 12th annual conference companion on Genetic and evolutionary computation*, pp. 1689–1696, 2010.

Hansen, N., Brockhoff, D., Mersmann, O., Tusar, T., Tusar, D., ElHara, O. A., Sampaio, P. R., Atamna, A., Varelas, K., Batu, U., Nguyen, D. M., Matzner, F., and Auger, A. COmparing Continuous Optimizers: numbbo/COCO on Github, March 2019. URL https://doi.org/10.5281/zenodo.2594848.

Hansen, P. and Jaumard, B. Lipschitz optimization. In *Handbook of global optimization*, pp. 407–493. Springer, 1995.

Kazemi, V. and Sullivan, J. One millisecond face alignment with an ensemble of regression trees. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1867–1874, 2014.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Lee, J. D., Simchowitz, M., Jordan, M. I., and Recht, B. Gradient descent converges to minimizers. *arXiv preprint arXiv:1602.04915*, 2016.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

Liu, Z., Luo, P., Wang, X., and Tang, X. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

Maheswaranathan, N., Metz, L., Tucker, G., Choi, D., and Sohl-Dickstein, J. Guided evolutionary strategies: Augmenting random search with surrogate gradients. *arXiv preprint arXiv:1806.10230*, 2018.

Mania, H., Guy, A., and Recht, B. Simple random search provides a competitive approach to reinforcement learning. *arXiv preprint arXiv:1803.07055*, 2018.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540): 529–533, 2015.

Nelder, J. A. and Mead, R. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

Nesterov, Y. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

Nocedal, J. and Wright, S. *Numerical optimization*. Springer Science & Business Media, 2006.

Pan, Z., Yu, W., Yi, X., Khan, A., Yuan, F., and Zheng, Y. Recent progress on generative adversarial networks (gans): A survey. *IEEE Access*, 7:36322–36333, 2019.

Powell, M. J. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The computer journal*, 7(2):155–162, 1964.

Powell, M. J. A view of algorithms for optimization without derivatives. *Mathematics Today-Bulletin of the Institute of Mathematics and its Applications*, 43(5):170–174, 2007.

Reinsch, C. H. Smoothing by spline functions. *Numerische mathematik*, 10(3):177–183, 1967.

Rios, L. M. and Sahinidis, N. V. Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, 56(3): 1247–1293, 2013.

Rosemarin, H., Rosenfeld, A., and Kraus, S. Emergency department online patient-caregiver scheduling. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 695–701, 2019.

Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Saremi, S. On approximating ∇f with neural networks. *arXiv preprint arXiv:1910.12744*, 2019.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. Trust region policy optimization. In *International Conference on Machine Learning*, pp. 1889–1897, 2015.

Sener, O. and Koltun, V. Learning to guide random search. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=B1gHokBKwS.

Shewchuk, J. R. et al. An introduction to the conjugate gradient method without the agonizing pain, 1994.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., Patwary, M., Prabhat, M., and Adams, R. Scalable bayesian optimization using deep neural networks. In *International conference on machine learning*, pp. 2171–2180, 2015.

Van Laarhoven, P. J. and Aarts, E. H. Simulated annealing. In *Simulated annealing: Theory and applications*, pp. 7–15. Springer, 1987.

Vemula, A., Sun, W., and Bagnell, J. A. Contrasting exploration in parameter and action space: A zeroth-order optimization perspective. *arXiv preprint arXiv:1901.11503*, 2019.

Volz, V., Schrum, J., Liu, J., Lucas, S. M., Smith, A., and Risi, S. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 221–228, 2018.

Xiao, H., Rasul, K., and Vollgraf, R. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms, 2017.

Yuan, X., He, P., Zhu, Q., and Li, X. Adversarial examples: Attacks and defenses for deep learning. *IEEE transactions on neural networks and learning systems*, 30(9): 2805–2824, 2019.

Zhang, W., Du, T., and Wang, J. Deep learning over multi-field categorical data. In *European conference on information retrieval*, pp. 45–57. Springer, 2016.

Zhen, L., Wang, K., Hu, H., and Chang, D. A simulation optimization framework for ambulance deployment and relocation problems. *Computers & Industrial Engineering*, 72:12–23, 2014.