Reproducibility Challenge NeurIPS 2019 Report on "Competitive Gradient Descent"

Gopi Kishan
Computer Science and Engineering
Indian Institute of Technology Roorkee
Roorkee, Uttarakhand
gkishan@cs.iitr.ac.in

Abstract

This is a report for reproducibility challenge of NeurIPS'19 on the paper *Competitive gradient descent* (Schäfer et al., 2019). The paper introduces a novel algorithm for the numerical computation of Nash equilibria of competitive two-player games. It avoids oscillatory and divergent behaviors seen in alternating gradient descent.

The purpose of this report is to critically examine the reproducibility of the work by (Schäfer et al., 2019), within the framework of the NeurIPS 2019 Reproducibility Challenge. The experiments replicated in this report confirms the results of the original study. Moreover, this project offers a Python(Pytorch based) implementation of the proposed CGD algorithm which can be found at the following public git repository: (`https://github.com/GopiKishan14/Reproducibility_Challenge_NeurIPS_2019`)

1 Introduction and Motivation

The original paper introduces a new algorithm for the numerical computation of Nash equilibria of competitive two-player games. Authors suggest their method is a natural generalization of gradient descent to the two-player scenario where the update is given by the Nash equilibrium of a regularized bilinear local approximation of the underlying game. It avoids oscillatory and divergent behaviors seen in alternating gradient descent.

The paper proposes several experiments to establish the robustness of their method. This project aims at replicating their results.

The paper provides a detailed comparison to methods based on *optimism* and *consensus* on the properties of convergence and stability of various discussed methods using numerical experiments and rigorous analysis

In order to understand these terms, comparison and proposed method and examine the results of the experiments, next section gives a necessary background of the original paper.

2 Background

The traditional optimization is concerned with a single agent trying to optimize a cost function. It can be seen as $\min_{x \in \mathbb{R}^m} f(x)$. The agent has a clear objective to find ("Good local") minimum of f. Gradeint Descent (and its varients) are reliable Algorithmic Baseline for this purpose.

Preprint. Under review.

The paper talks about **Competitive optimization**. Competitive optimization extends this problem to the setting of multiple agents each trying to minimize their own cost function, which in general depends on the actions of all agents.

The paper deals with the case of two such agents:

$$\min_{x\in\mathbb{R}^m} f(x,y), \quad \min_{y\in\mathbb{R}^n} g(x,y) \tag{1}$$

for two functions $f, g : \mathbb{R}^m \times \mathbb{R}^n \longrightarrow \mathbb{R}$.

In single agent optimization, the solution of the problem consists of the minimizer of the cost function. In competitive optimization, the right definition of *solution* is less obvious, but often one is interested in computing Nash– or strategic equilibria: Pairs of strategies, such that no player can decrease their costs by unilaterally changing their strategies. If f and g are not convex, finding a global Nash equilibrium is typically impossible and instead we hope to find a "good" local Nash equilibrium.

Gradient descent/ascent and the cycling problem: For differentiable objective functions, the most naive approach to solving (1) is gradient descent ascent (GDA), whereby both players independently change their strategy in the direction of steepest descent of their cost function. Unfortunately, this procedure features oscillatory or divergent behavior even in the simple case of a bilinear game ($f(x,y) = x^\top y = -g(x,y)$) (see Figure 3). In game-theoretic terms, GDA lets both players choose their new strategy optimally with respect to the last move of the other player. Thus, the cycling behaviour of GDA is not surprising: It is the analogue of *"Rock! Paper! Scissors! Rock! Paper! Scissors! Rock! Paper!..."* in the eponymous hand game. While gradient descent is a reliable basic *workhorse* for single-agent optimization, GDA can not play the same role for competitive optimization. At the moment, the lack of such a *workhorse* greatly hinders the broader adoption of methods based on competition.

2.1 Competitive gradient descent

Authors propose a novel algorithm, which they call *competitive gradient descent* (CGD), for the solution of competitive optimization problems $\min_{x\in\mathbb{R}^m} f(x,y)$, $\min_{y\in\mathbb{R}^n} g(x,y)$, where they have access to function evaluations, gradients, and Hessian-vector products of the objective functions. [1]

Algorithm 1: Competitive Gradient Descent (CGD)

for $0 \le k \le N-1$ **do**

$$x_{k+1} = x_k - \eta\left(\mathrm{Id} - \eta^2 D^2_{xy} f D^2_{yx} g\right)^{-1}\left(\nabla_x f - \eta D^2_{xy} f \nabla_y g\right);$$

$$y_{k+1} = y_k - \eta\left(\mathrm{Id} - \eta^2 D^2_{yx} g D^2_{xy} f\right)^{-1}\left(\nabla_y g - \eta D^2_{yx} g \nabla_x f\right);$$

return (x_N, y_N);

To motivate this algorithm, authors remind us that gradient descent with stepsize η applied to the function $f : \mathbb{R}^m \longrightarrow \mathbb{R}$ can be written as

$$x_{k+1} = \mathrm{argmin}_{x\in\mathbb{R}^m}(x^\top - x_k^\top)\nabla_x f(x_k) + \frac{1}{2\eta}\|x - x_k\|^2.$$

This models a (single) player solving a local linear approximation of the (minimization) game, subject to a quadratic penalty that expresses her limited confidence in the global accuracy of the model. The natural generalization of this idea to the competitive case should then be given by the two players solving a local approximation of the true game, both subject to a quadratic penalty that expresses their limited confidence in the accuracy of the local approximation.
In order to implement this idea, we need to find the appropriate way to generalize the linear approximation in the single agent setting to the competitive setting.

Authors suggest to use a *bilinear* approximation in the two-player setting. Since the bilinear approximation is the lowest order approximation that can capture some interaction between the two players, they argue that the natural generalization of gradient descent to competitive optimization

[1] Here and in the following, all derivatives are evaluated in the point (x_k, y_k)

is not GDA, but rather the update rule $(x_{k+1}, y_{k+1}) = (x_k, y_k) + (x, y)$, where (x, y) is a Nash equilibrium of the game [2]

$$\begin{aligned} &\min_{x\in\mathbb{R}^m} x^\top \nabla_x f + x^\top D^2_{xy} f y + y^\top \nabla_y f + \frac{1}{2\eta} x^\top x \\ &\min_{y\in\mathbb{R}^n} y^\top \nabla_y g + y^\top D^2_{yx} g x + x^\top \nabla_x g + \frac{1}{2\eta} y^\top y. \end{aligned} \tag{2}$$

Indeed, the (unique) Nash equilibrium of the Game (2) can be computed in closed form.

Theorem 2.1. *Among all (possibly randomized) strategies with finite first moment, the only Nash equilibrium of the Game* (2) *is given by*

$$\begin{aligned} x &= -\eta \left(\mathrm{Id} - \eta^2 D^2_{xy} f D^2_{yx} g\right)^{-1} \left(\nabla_x f - \eta D^2_{xy} f \nabla_y g\right) \\ y &= -\eta \left(\mathrm{Id} - \eta^2 D^2_{yx} g D^2_{xy} f\right)^{-1} \left(\nabla_y g - \eta D^2_{yx} g \nabla_x f\right), \end{aligned}$$

given that the matrix inverses in the above expression exist. [3]

An elegant proof of the above theorem is presented in the original paper (Schäfer et al., 2019).

According to Theorem 2.1, the Game (2) has exactly one optimal pair of strategies, which is deterministic. Thus, we can use these strategies as an update rule, generalizing the idea of local optimality from the single– to the multi agent setting and obtaining Algorithm 1.

What I think that they think that I think ... that they do: Another game-theoretic interpretation of CGD follows from the observation that its update rule can be written as

$$\begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix} = -\begin{pmatrix} \mathrm{Id} & \eta D^2_{xy} f \\ \eta D^2_{yx} g & \mathrm{Id} \end{pmatrix}^{-1} \begin{pmatrix} \nabla_x f \\ \nabla_y g \end{pmatrix}.$$

Applying the expansion $\lambda_{\max}(A) < 1 \Rightarrow (\mathrm{Id} - A)^{-1} = \lim_{N\to\infty} \sum_{k=0}^{N} A^k$ to the above equation, we observe that:

- The first partial sum ($N = 0$) corresponds to the optimal strategy if the other player's strategy stays constant (GDA).
- The second partial sum ($N = 1$) corresponds to the optimal strategy if the other player thinks that the other player's strategy stays constant (LCGD, see Figure 1).
- The third partial sum ($N = 2$) corresponds to the optimal strategy if the other player thinks that the other player thinks that the other player's strategy stays constant, and so forth, until the Nash equilibrium is recovered in the limit.

2.2 Comparison with other methods

As illustrated in Figure 1, these six algorithms amount to different subsets of the following four terms.

1. The *gradient term* $-\nabla_x f$, $\nabla_y f$ which corresponds to the most immediate way in which the players can improve their cost.
2. The *competitive term* $-D_{xy} f \nabla_y f$, $D_{yx} f \nabla_x f$ which can be interpreted either as anticipating the other player to use the naive (GDA) strategy, or as decreasing the other players influence (by decreasing their gradient).
3. The *consensus term* $\pm D^2_{xx} \nabla_x f$, $\mp D^2_{yy} \nabla_y f$ that determines whether the players prefer to decrease their gradient ($\pm = +$) or to increase it ($\pm = -$). The former corresponds the players seeking consensus, whereas the latter can be seen as the opposite of consensus. (It also corresponds to an approximate Newton's method. [4])

[2] We could alternatively use the penalty $(x^\top x + y^\top y)/(2\eta)$ for both players, without changing the solution.

[3] We note that the matrix inverses exist for all but one value of η, and for all η in the case of a zero sum game.

[4] Applying a damped and regularized Newton's method to the optimization problem of Player 1 would amount to choosing $x_{k+1} = x_k - \eta(\mathrm{Id} + \eta D^2_{xx})^{-1} f \nabla_x f \approx x_k - \eta(\nabla_x f - \eta D^2_{xx} f \nabla_x f)$, for $\|\eta D^2_{xx} f\| \ll 1$.

$$
\begin{array}{lll}
\text{GDA:} & \Delta x = & -\nabla_x f \\
\text{LCGD:} & \Delta x = & -\nabla_x f \quad -\eta D^2_{xy} f \nabla_y f \\
\text{SGA:} & \Delta x = & -\nabla_x f \quad -\gamma D^2_{xy} f \nabla_y f \\
\text{ConOpt:} & \Delta x = & -\nabla_x f \quad -\gamma D^2_{xy} f \nabla_y f \quad -\gamma D^2_{xx} f \nabla_x f \\
\text{OGDA:} & \Delta x \approx & -\nabla_x f \quad -\eta D^2_{xy} f \nabla_y f \quad +\eta D^2_{xx} f \nabla_x f \\
\text{CGD:} & \Delta x = \left(\mathrm{Id} + \eta^2 D^2_{xy} f D^2_{yx} f\right)^{-1} & \left(-\nabla_x f \quad -\eta D^2_{xy} f \nabla_y f \right)
\end{array}
$$

Figure 1: The update rules of the first player for (from top to bottom) GDA, LCGD, ConOpt, OGDA, and CGD, in a zero-sum game ($f = -g$).

4. The *equilibrium term* $(\mathrm{Id} + \eta^2 D^2_{xy} D^2_{yx} f)^{-1}$, $(\mathrm{Id} + \eta^2 D^2_{yx} D^2_{xy} f)^{-1}$, which arises from the players solving for the Nash equilibrium. This term lets each player prefer strategies that are less vulnerable to the actions of the other player.

Further Discussion:

Consensus optimization **(ConOpt)** (Mescheder et al., 2017), penalises the players for non-convergence by adding the squared norm of the gradient at the next location, $\gamma \|\nabla_x f(x_{k+1}, y_{k+1}), \nabla_x f(x_{k+1}, y_{k+1})\|^2$ to both player's loss function (here $\gamma \geq 0$ is a hyperparameter).
(Daskalakis et al., 2017) proposed to modify GDA as

$$
\begin{aligned}
\Delta x &= -\left(\nabla_x f(x_k, y_k) + \left(\nabla_x f(x_k, y_k) - \nabla_x f(x_{k-1}, y_{k-1})\right)\right) \\
\Delta y &= -\left(\nabla_y g(x_k, y_k) + \left(\nabla_y g(x_k, y_k) - \nabla_y g(x_{k-1}, y_{k-1})\right)\right),
\end{aligned}
$$

which we will refer to as ***optimistic gradient descent ascent*** **(OGDA)**. By interpreting the differences appearing in the update rule as finite difference approximations to Hessian vector products, we see that (to leading order) OGDA corresponds to yet another second order correction of GDA (see Figure 1).

It will also be instructive to compare the algorithms to ***linearized competitive gradient descent*** **(LCGD)**, which is obtained by skipping the matrix inverse in CGD (which corresponds to taking only the leading order term in the limit $\eta D^2_{xy} f \to 0$).

3 Experiments and Replications

This section contains the contribution from this project and defines the experiment.
A link to public Github repository (`https://github.com/GopiKishan14/Reproducibility_Challenge_NeurIPS_2019`) describes the implementation details of *Competitive Gradient Descent*.

A brief discussion on the implementation of CGD by Authors.
Computing Hessian vector products: First, our algorithm requires products of the mixed Hessian $v \mapsto D_{xy} f v$, $v \mapsto D_{yx} g v$, which we want to compute using automatic differentiation. Many AD frameworks, like Autograd (`https://github.com/HIPS/autograd`) and ForwardDiff(`https://github.com/JuliaDiff/ForwardDiff.jl`, (Revels et al., 2016)) together with ReverseDiff(`https://github.com/JuliaDiff/ReverseDiff.jl`) support this procedure.

Matrix inversion for the equilibrium term: Similar to a *truncated Newton's method* (Nocedal and Wright, 2006), we propose to use iterative methods to approximate the inverse-matrix vector products arising in the equilibrium term 4. We will focus on zero-sum games, where the matrix is always symmetric positive definite, making the conjugate gradient (CG) algorithm the method of choice. We suggest terminating the iterative solver after a given relative decrease of the residual is achieved ($\|Mx - y\| \leq \epsilon \|x\|$ for a small parameter ϵ, when solving the system $Mx = y$). In our experiments we choose $\epsilon = 10^{-6}$. Given the strategy Δx of one player, Δy is the optimal counter strategy which can be found without solving another system of equations.

3.1 Experiment 1

The author suggest that each of the terms (section 2.2) used in the definition of the algorithm is responsible for a different feature of the corresponding algorithm which can illustrated by applying the algorithms to three prototypical test cases.

1. **Test Case 1:** We first consider the bilinear problem $f(x, y) = \alpha xy$ (see Figure 3).
   Replication and Findings: As suggested, GDA fails on this problem, for any value of stepsize η.
   For $\alpha = 1.0$, all the other methods converge exponentially towards the equilibrium, with ConOpt and SGA converging at a faster rate.
   For $\alpha = 3.0$, OGDA diverges, while ConvOpt and SGA begin to oscillate widely.
   For $\alpha = 6.0$, all methods but CGD diverge. This result is compared and summed up in Figure (3)
   However, for $\alpha = 3.0$, if we decrease γ *to* 0.5 ConOpt and SGA converges. ConOpt and SGA does not converge for any value of γ if $\alpha = 6.0$. Figure 4 Left.
   For $\alpha = 6.0$, if stepsize η is increased to $\eta = 0.9$, CGD converges faster. Hence, establishing the claim that CGD convergence rate increases with increase in stepsize η. Figure 4 Middle.

   Interestingly, CGD fails for the problem if initial points set on $y = 5/3x$ for $\alpha = 3.0$ and $\eta = 0.2$. But LCGD still converges. Figure (4) Right.

2. **Test case 2:** In order to explore the effect of the consensus Term 3, we now consider the convex-concave problem $f(x, y) = \alpha(x^2 - y^2)$ (see Figure 6).
   Replication and Findings: For $\alpha = 1.0$, all algorithms converge at an exponential rate, with ConOpt converging the fastest, and OGDA the slowest.

   As we increase α to $\alpha = 3.0$, the OGDA and ConOpt start failing (diverge), while the remaining algorithms still converge at an exponential rate. Upon increasing α further to $\alpha = 6.0$, all algorithms diverge. (See Figure 6)
   Figure 7 shows the robustness of CGD, while other methods can also be tuned for convergance at $\alpha = 3.0$ and $\eta = 0.2$ on decreasing γ.

3. **Test case 3:** We further investigate the effect of the consensus Term 3 by considering the concave-convex problem $f(x, y) = \alpha(-x^2 + y^2)$ (see Figure 6).
   Replication and Findings: The critical point $(0, 0)$ does not correspond to a Nash-equilibrium, since both players are playing their *worst possible strategy*. Thus it is highly undesirable for an algorithm to converge to this critical point.
   However for $\alpha = 1.0$, ConOpt does converge to $(0, 0)$ which provides an example of the consensus regularization introducing spurious solutions. The other algorithms, instead, diverge away towards infinity, as would be expected. In particular, we see that SGA is correcting the problematic behavior of ConOpt, while maintaining its better convergence rate in the first example. As we increase α to $\alpha \in \{3.0, 6.0\}$, the radius of attraction of $(0, 0)$ under ConOpt decreases and thus ConOpt diverges from the starting point $(0.5, 0.5)$, as well. (See Figure 6)
   Figure 8 shows the robustness of CGD, while ConOpt can also be tuned for convergence at $\alpha = 1.0$ and $\eta = 0.2$ on decreasing γ $\gamma \in \{1.0, 0.5, 0.2\}$ (divergence desired).

Conclusion **:** The first test case illustrates that this is not just a lack of theory, but corresponds to an actual failure mode of the existing algorithms. Introducing the competitive term (2) is enough to fix the cycling behaviour of GDA, OGDA and ConOpt (for small enough η).

In the Test Case 2 (where convergence is desired), OGDA converges in a smaller parameter range than GDA and SGA, while only diverging slightly faster in the Test Case 3 (where divergence is desired). ConOpt, on the other hand, converges faster than GDA in the Test Case 2, for $\alpha = 1.0$ however, it diverges faster for the remaining values of α and, what is more problematic, it converges to a spurious solution in the Test Case 3 for $\alpha = 1.0$.

Based on the findings of Test Case 2 and 3, the consensus term (3) with either sign does not seem to systematically improve the performance of the algorithm (see Fig 7 and 8), which is why the authors suggest to only use the competitive term (that is, use LOLA/LCGD, or CGD, or SGA).

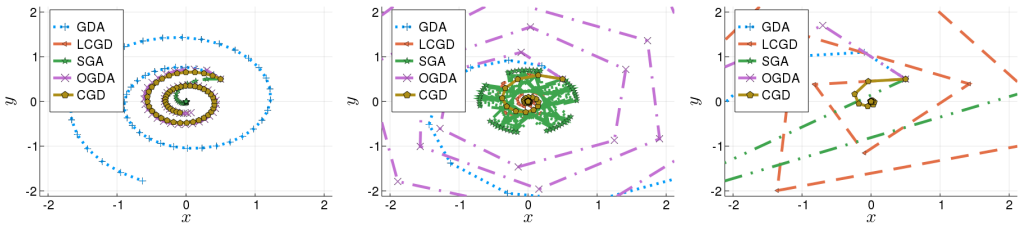

Figure 2: Original Paper Result

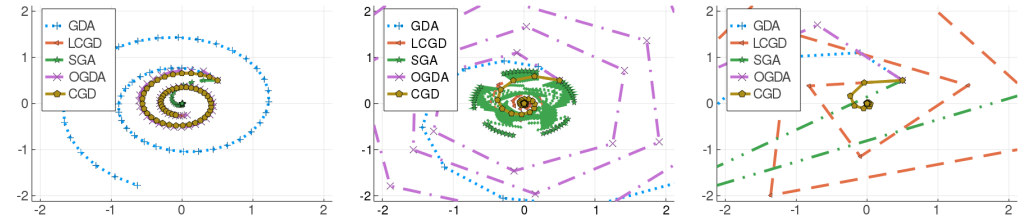

Figure 3: Replicated Result: The first 50 iterations of GDA, LCGD, ConOpt, OGDA, and CGD with parameters $\eta = 0.2$ and $\gamma = 1.0$. The objective function is $f(x, y) = \alpha x^\top y$ for, from left to right, $\alpha \in \{1.0, 3.0, 6.0\}$. (Note that ConOpt and SGA coincide on a bilinear problem)

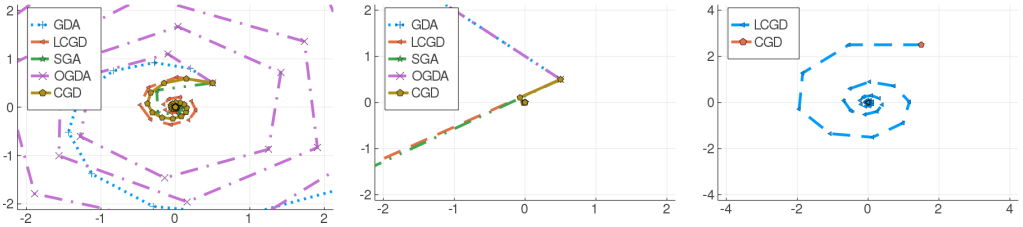

Figure 4: Results for Experiment 3.1 Test 1. (Refer to the part for explanation)

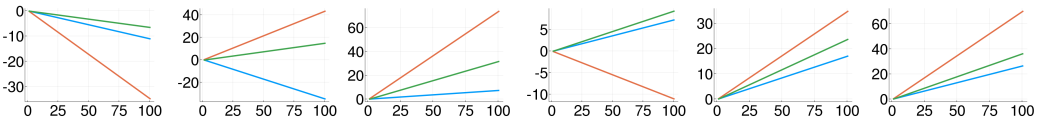

Figure 5: Original Paper result

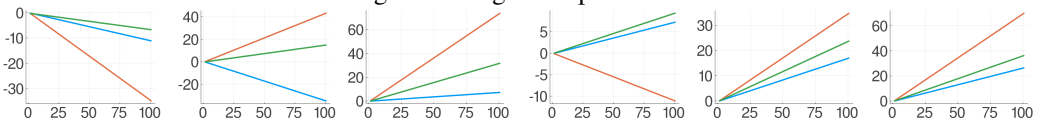

Figure 6: Replicated result: We measure the (non-)convergence to equilibrium in the separable convex-concave– ($f(x, y) = \alpha(x^2 - y^2)$, left three plots) and concave convex problem ($f(x, y) = \alpha(-x^2 + y^2)$, right three plots), for $\alpha \in \{1.0, 3.0, 6.0\}$. (Color coding given by GDA, SGA, LCGD, CGD, ConOpt, OGDA, the y-axis measures $\log_{10}(\|(x_k, y_k)\|)$ and the x-axis the number of iterations k. Note that convergence is desired for the first problem, while *divergence* is desired for the second problem.

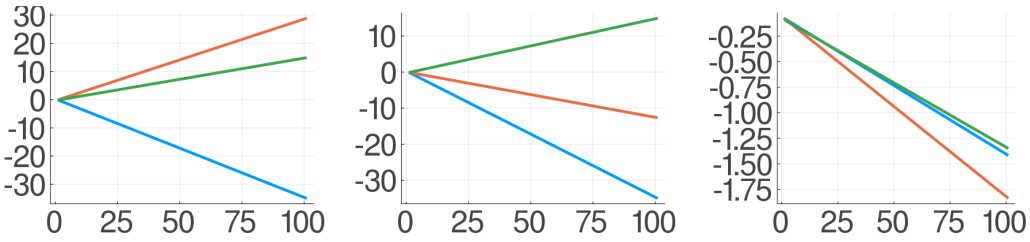

Figure 7: Results for Experiment 3.1 Test 2. Shows the robustness of CGD, while other methods can also be tuned for convergence at $\alpha = 3.0$ and $\eta = 0.2$ on decreasing γ. (convergence desired)

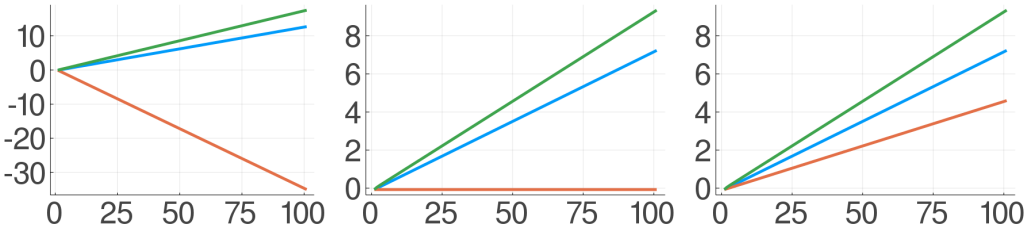

Figure 8: Results for Experiment 3.1 Test 3. Shows the robustness of CGD, while ConOpt can also be tuned for convergence at $\alpha = 1.0$ and $\eta = 0.2$ on decreasing γ.. $\gamma \in \{1.0, 0.5, 0.2\}$ (divergence desired)

3.2 Experiment 2: Estimating a covariance matrix

To show that CGD is also competitive in terms of computational complexity, authors consider the noiseless case of the covariance estimation example used by Daskalakis et al. (2017).

Authors consider the problem $-g(V, W) = f(W, V) = \sum_{ijk} W_{ij} \left(\hat{\Sigma}_{ij} - (V\hat{\Sigma}V^\top)_{i,j}\right)$, where the $\hat{\Sigma}$ are empirical covariance matrices obtained from samples distributed according to $\mathcal{N}(0, \Sigma)$. For the experiments, the matrix Σ is created as $\Sigma = UU^T$, where the entries of $U \in \mathbb{R}^{d\times d}$ are distributed i.i.d. standard Gaussian.

Replication and Findings: As described in the original paper, this project considers the algorithms OGDA, SGA, ConOpt, and CGD for deterministic case $\hat{\Sigma} = \Sigma$, corresponding to the limit of large sample size, with $\gamma = 1.0$, $\epsilon = 10^{-6}$. Let dimensions of Σ, $d \in \{20, 40, 60\}$ and let the stepsizes range over $\eta \in \{0.005, 0.025, 0.1, 0.4\}$.
Authors suggest to evaluate the algorithms according to the trade-off between the number of forward evaluations and the corresponding reduction of the residual $\|W + W^\top\|_{\text{FRO}}/2 + \|UU^\top - VV^\top\|_{\text{FRO}}$, starting with a random initial guess (the same for all algorithms) obtained as $W_1 = \delta W$, $V_1 = U + \delta V$, where the entries of $\delta W, \delta V$ are i.i.d uniformly distributed in $[-0.5, 0.5]$. Authors count the number of "forward passes" per outer iteration as follows.

- OGDA: 2
- SGA: 4
- ConOpt: 6
- CGD: 4 + 2 $*$ number of CG iterations

The results are summarized in Figure 9 and Figure 10. We see consistently that for the same stepsize, CGD has convergence rate comparable to that of OGDA. However, as we increase the stepsize the other methods start diverging, thus allowing CGD to achieve significantly better convergence rates by using larger stepsizes. For larger dimensions ($d \in \{40, 60\}$) OGDA, SGA, and ConOpt become even more unstable such that OGDA with the smallest stepsize is the only other method that still converges, although at a much slower rate than CGD with larger stepsizes.

Conclusion **:** On studying the tradeoff between the number of evaluations of the forward model and the residual, this project supports that larger stepsize, the convergence rate of CGD is better than

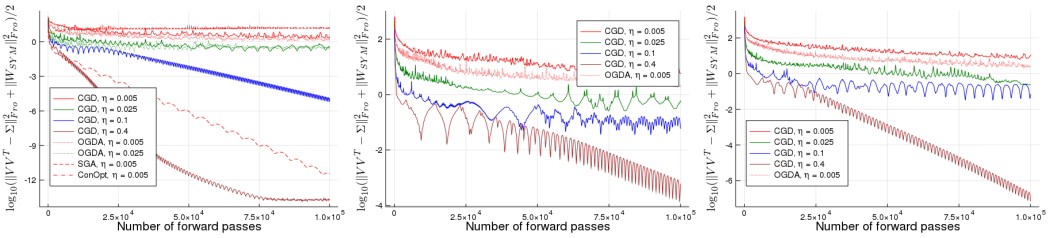

Figure 9: Original Paper result:

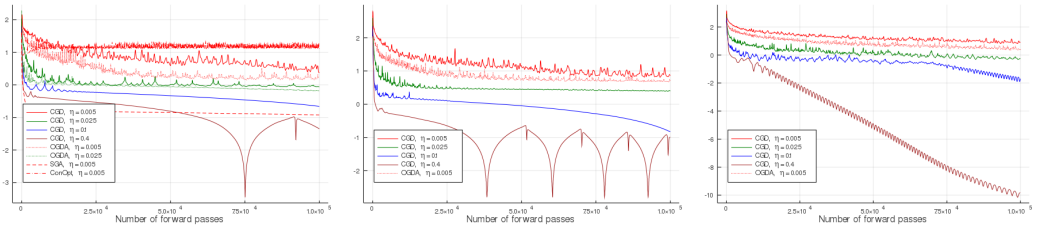

Figure 10: Replication result: Plot of the decay of the residual after a given number of model evaluations, for increasing problem sizes and $\eta \in \{0.005, 0.025, 0.1, 0.4\}$.

the other methods and for comparable stepsize, the convergence rate of CGD is similar to the other methods.

3.3 Experiment 3: Fitting a bimodal distribution

To further study the robustness of the CGD, authors propose to use a simple GAN to fit a Gaussian mixture model with two modes, in two dimensions.

Replication and Findings We use a GAN to fit a Gaussian mixture of two Gaussian random variables with means $\mu_1 = (0, 1)^\top$ and $\mu_2 = (2^{-1/2}, 2^{-1/2})^\top$, and standard deviation $\sigma = 0.1$ Generator and discriminator are given by dense neural nets with four hidden layers of 128 units each that are initialized as orthonormal matrices, and ReLU as nonlinearities after each hidden layer. The generator uses 512-variate standard Gaussian noise as input, and both networks use a linear projection as their final layer. At each step, the discriminator is shown 256 real, and 256 fake examples. We interpret the output of the discriminator as a logit and use sigmoidal crossentropy as a loss function.The results are summarized in Figure 11 and Figure 12.

We apply SGA, ConOpt ($\gamma = 1.0$), OGDA, and CGD for stepsize $\eta \in \{0.4, 0.1, 0.025, 0.005\}$ together with RMSProp ($\rho = 0.9$). In each case, CGD produces an reasonable approximation of the input distribution without any mode collapse. In contrast, all other methods diverge after some initial cycling behaviour.

Conclusion **:** On all methods, the generator and discriminator are initially chasing each other across the strategy space, producing the typical cycling pattern. When using SGA, ConOpt, or OGDA, however, eventually the algorithm diverges with the generator either mapping all the mass far away from the mode, or collapsing the generating map to become zero. Therefore, we also tried decreasing the stepsize to 0.001, which however did not prevent the divergence. For CGD, after some initial cycles the generator starts splitting the mass and distributes is roughly evenly among the two modes.

This project supports authors' claim that CGD is significantly more robust than existing methods for competitive optimization.

4 Conclusion and Acknowledgement

This project mostly successfully replicates the work of original paper. It supports the properties of introduced *Competitive Gradient Descent* method for Competitive optimization through the pro-

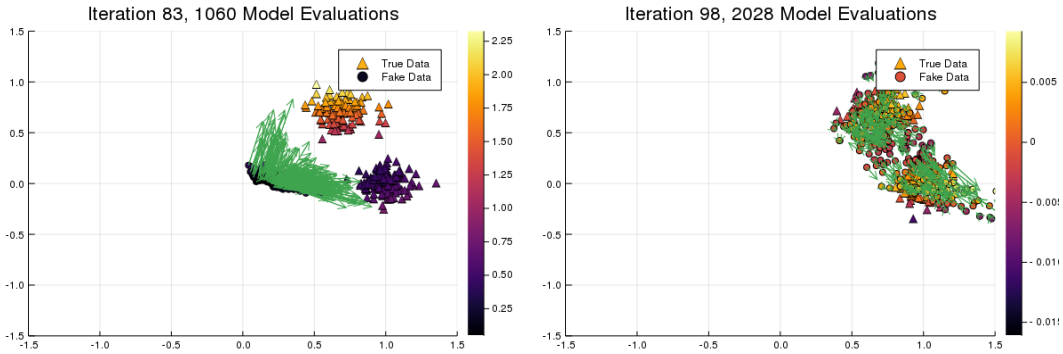

Figure 11: Original Paper result:

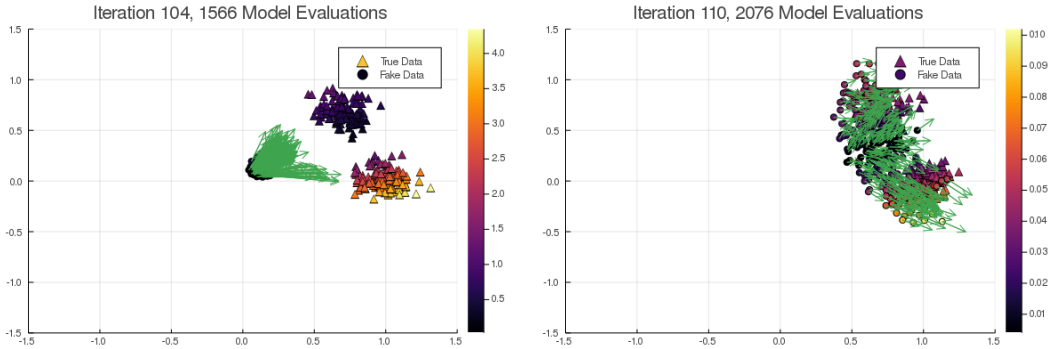

Figure 12: Replication result: Under CGD, the mass eventually distributes evenly among the two modes.

posed experiments. The three experiments test different aspects of the algorithms. Experiment 1 is based on removal of cyclic behaviour and importance of different definition terms. It also tests convergence/divergence of the algorithms on different zero-sum games. Experiment 2 evaluates and compares computational complexity of the CGDA w.r.t other algorithms. Experiment 3 studies the robustness of CGDA by fitting bimodal distribution using GAN.

This project acknowledges the code of the original paper written in Julia used for replication. The original paper describes the experiments very well and their code is also very understandable. It also acknowledges the support of Code Ocean for providing compute resources and prof. Debasish Ghose, Department of Aerospace Engineering, Indian Institute of Science.

References

Florian Schäfer, and Anima Anandkumar (2019). Competitive Gradient Descent. In *Advances in Neural Information Processing Systems. arXiv:1905.12103v2*

Daskalakis, C., Ilyas, A., Syrgkanis, V., and Zeng, H. (2017). Training gans with optimism. *arXiv preprint arXiv:1711.00141*.

Mescheder, L., Nowozin, S., and Geiger, A. (2017). The numerics of gans. In *Advances in Neural Information Processing Systems*, pages 1825–1835.

Nocedal, J. and Wright, S. J. (2006). *Numerical optimization*. Springer Series in Operations Research and Financial Engineering. Springer, New York, second edition.

Revels, J., Lubin, M., and Papamarkou, T. (2016). Forward-mode automatic differentiation in julia. *arXiv:1607.07892 [cs.MS]*.