Visual Analysis of Optimization Algorithms

Juliette Love
CS205L Winter 2020

Abstract

For this project, I built an interactive visualization system to analyze the performance of different optimization algorithms. It aims to help the user understand which algorithms perform best on which types of functions, and with which types of initializations. The user can input a function in three dimensions and an initial guess (x,y), and the program will display the paths of different optimization algorithms as they search for a minimum on that function. In this paper, I consider a few common types of three-dimensional functions, and use this system to analyze which algorithms are best suited to optimization problems on each one.

1. Introduction

The rise in popularity of machine learning has been accompanied with a rise in the number of optimization algorithms in common practice, most of which are variants on Gradient Descent. Each of these algorithms incorporates information about the function differently, and thus it is not always clear which of these algorithms will be most effective for a given problem. In fact, the definition of efficacy is not always clearthere are different considerations when evaluating algorithm performance, such as speed, precision, and robustness to local minima. Especially because the differences in design between the algorithms are not always obvious, it is difficult to tell without trial and error how these differences will affect how each algorithm will perform along each of these metrics.

This project is meant to help people understand these algorithms in a visual way. It expands upon existing work in this area in that it is both visual and highly customizablemost projects with similar goals provide, at best, animations of these algorithms running on set functions and initializations. By contrast, this system allows the user to test the performance of each algorithm on a function of their choice, as well as under different initializations for the same function. At present, the user can see the path of five algorithms-Gradient Descent, Gradient Descent with momentum, AdaGrad, RMSProp, and Adam.

2. Related Work

Two of the most similar and most sophisticated software in this space that I have encountered are described below. I aim to address the shortcomings in both of these projects with my visualization tool.

2.1. *Optimizer Visualizations*

This project[11] showcases the performance of these algorithms on different types of functions. It displays the performance of seven algorithms on three types of functions: one with shallow valleys, one with steep valleys, and one with a shallow initial gradient. However, it only provides one preset function for each of these function types, and does not allow the user to input a custom function. This makes it less useful as a practical toolfor users with problems that may differ significantly from these three types, it is difficult to extrapolate any relevant information about the algorithms relative performance. Additionally, the project uses a single initialization location for all functions. The initialization can be very important in choosing the right optimizer; for example, if you are confident that your optimization problem can be initialized very close to the actual solution, you might want to choose a method that favors precision over speed.

2.2. *Interactive Visualization of Optimization Algorithms in Deep Learning*

This project[3] provides interactivity and allows for exploration by allowing the user to choose the initialization point with a mouse click. It renders a canvas with a three-dimensional function displayed as a heatmap, and traces the path of four optimization algorithms from the user-defined initialization point. However, it provides this interactivity only on a single test function, and analyzes these algorithms on only two more [note: provides links to the other two for testing]. As before, this makes it more difficult for a user with an optimization problem that differs from these preset functions to practically apply the insights from this project.

3. Methods, Data, and Software Libraries

3.1. Algorithms

The system renders 5 optimization algorithms, which I coded from scratch. Here is a brief overview of each:

3.1.1 Gradient Descent

First is vanilla gradient descent, where each update is defined by $x^{t+1} = x^t - \nabla_f(x^t)$ for learning rate α. This is the most standard form of gradient descentgiven an initial point x, the algorithm moves opposite the direction of the steepest gradient until reaching some stopping condition where $x^{t+1} - x^t < \epsilon$.

3.1.2 Gradient Descent with Momentum

Next is a version of gradient descent with an added momentum term; the formula for this is given by

$$v^{t+1} = \gamma v^t - \nabla_f(x^t) \tag{1}$$

$$x^{t+1} = x^t + v^t \tag{2}$$

for an additional momentum parameter γ. Gradient Descent with momentum often arrives at a solution more quickly than vanilla Gradient Descent because it can incorporate information about past gradient values in order to make quicker parameter updates. It can also escape local minima, unlike vanilla Gradient Descent. However, the use of momentum can sometimes lead to overshooting the target minimum, or an increase in oscillation, particularly if the gradient near the minimum is steep.

3.1.3 AdaGrad

Adagrad[2] expands upon Gradient Descent by introducing an adaptive learning rate: instead of a constant learning rate applied to each parameter, it varies the learning rate based on the gradient in each direction. It relies on a cache V with an entry for each parameter p:

$$V_p^{t+1} = \nabla_f(x^t)^2 \tag{3}$$

$$x^{t+1} = x^t - \alpha \nabla_f(x^t) \frac{1}{V_p^t + \epsilon} \tag{4}$$

With vanilla Gradient Descent, if the gradient is steep in one direction and shallow in another, the speed of convergence will be limited by the shallowest gradient direction. AdaGrad solves this problem by updating the parameters more quickly in directions where the gradient is smaller.

3.1.4 RMSProp

Adagrad faces the issue of a rapidly-decaying learning rate as the denominator grows, causing parameters with large gradients to update very slowly after some time. RMSProp, or Root Mean Square Propagation, addresses this issue by decaying the denominator as well with a decay term β:

$$V_p^{t+1} = \beta V_p^t + (1-\beta)\nabla_f(x^t)^2 \tag{5}$$

$$x^{t+1} = x^t - \alpha \nabla_f(x^t) \frac{1}{\sqrt{V_p^t + \epsilon}} \tag{6}$$

3.1.5 Adam

Adam expands upon RMSProp by utilizing the gradient history, balancing the improvements of the adaptive learning rate algorithms with those of a momentum term. For momentum parameter β_1 and velocity parameter β_2, the updates are defined by:

$$m^{t+1} = \beta_1 m^t + (1-\beta_1)\nabla_f(x^t) \tag{7}$$

$$v^{t+1} = \beta_2 v^t + (1-\beta_2)\nabla_f(x^t)^2 \tag{8}$$

$$x^{t+1} = x^t - \alpha m^t \frac{1}{\sqrt{v^t + \epsilon}} \tag{9}$$

3.2. Software Pipeline

The system (Fig. 1) displays in the browser and consists of a Python backend connected via Flask to a JavaScript/HTML/CSS frontend. A simple HTML input allows the user to submit a function; the function is displayed using D3.js with the d3-contour extension[6]. The user inputs initializations via a click on the svg rendered in the previous step. The function and initializations are passed to a Python script which runs each algorithm until convergence, stopping after 1000 steps if it has not yet converged. For each algorithm, it calculates and stores the values of the parameters at each time step, as well as the number of steps until convergence. This Python script works for functions with any number of variables; however, given the difficulty of rendering high-dimensional functions in two dimensions, the JavaScript function renderer only accepts functions of up to two variables.

The algorithms and gradient calculations are written from scratch; additional safeguards are implemented throughout that address underflow and overflow errors, which arise most often when the input function does not have a minimum. The parameter values for each algorithm at each step are packaged together and sent back to the JavaScript frontend, which uses D3.js to animate the results. Additional controls allow the user to change the zoom level on the function as well as the animation speed.

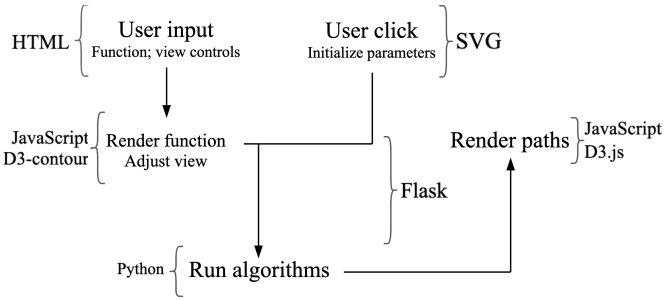

Figure 1. Software Pipeline

4. Experiments, Results, and Analysis

We can use this system to evaluate the performance of these algorithms on various function types, and with different initializations. I tested the system on a variety of different function types, relying on commonly-used test functions when possible. The visualization works best when both the x and y coordinates of the initializations and desired minima are roughly within the range $[-2, 2]$, so for some of commonly-used functions I had to scale the inputs accordingly.

A note on hyperparameter initialization: for all of these functions, the values of any hyperparameters were taken from the widely-accepted default values, usually original papers from these algorithms. I used $\gamma = 0.9$, $\beta = 0.9$, and $\beta_1 = 0.9, \beta_2 = 0.99$ for Gradient Descent with momentum[8], RMSProp[10], and Adam[5], respectively.

4.1. Convex, soft minimum

Here I relied on a simple test function

$$f(x, y) = x^2 + y^2 \tag{10}$$

This is a convex function with a single (local and global) minimum, located at the origin. All algorithms find the minimum with relative ease, diverging little from the most direct path and converging in fewer than 500 steps given an initialization at a Euclidean distance of around 2 from the origin (Fig. 2). However, different algorithms perform slightly better on this function depending on the location of the initialization. Since the function is symmetrical in all directions around the z-axis, we can analyze these algorithms based on the distance between the initialization and the minimum.

Although the relative performance of these algorithms is dependent on their parameter values, we can see that their performance scales differently with distance (Fig. 3). Gradient Descent and Gradient Descent with momentum both appear to scale logarithmically with distance; RMSProp and Adam appear relatively linear; AdaGrad scales exponentially. Thus, the tool suggests that for initializations close

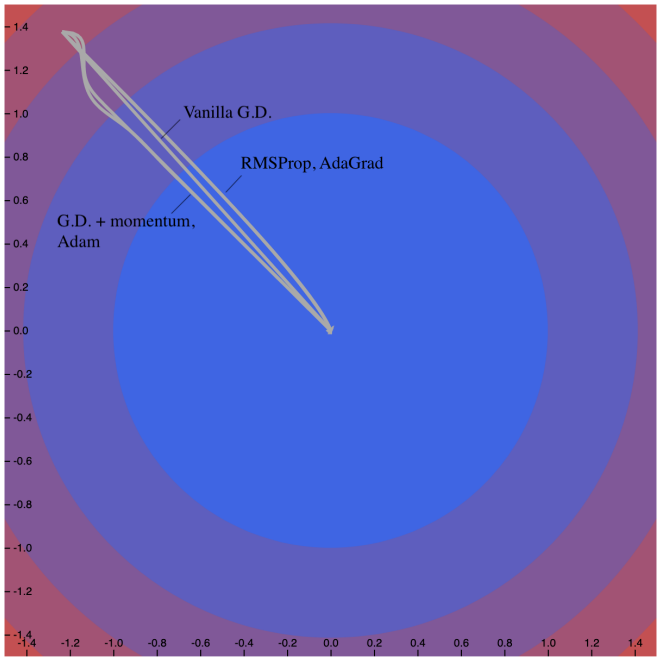

Figure 2. Convergence on $f(x, y) = x^2 + y^2$

to the true minimum for this type of function, AdaGrad and RMSProp are most effective, while simpler versions of Gradient Descent would be more appropriate at larger distances.

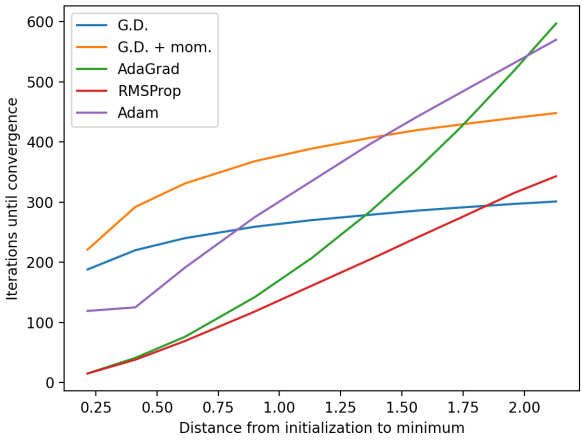

Figure 3. Speed of convergence on $f(x, y) = x^2 + y^2$

Using this tool we can also see that the underperformance of Gradient Descent with momentum relative to vanilla Gradient Descent is due to the oscillation of the momentum algorithm around the minimum; although it nears

the minimum more quickly, it takes longer to converge. This could be solved by using dynamic step size updates; thus, this system could spark a user to include a decaying learning rate if using a momentum-based optimization approach.

4.2. Convex, sharp minimum: Penholder function

The efficacy of these algorithms changes when the minimum is no longer a smooth basin but rather a sharp point (Fig. 4). The Penholder[7] function is a commonly-cited example, defined by

$$f(x, y) = -e^{1-\frac{(x^2+y^2)^{0.5}}{\pi}} \tag{11}$$

Most notably, Adam fails to converge on this function, since the step size is too large to get close enough to the minimum to trigger the stopping condition (Fig. 5). Additionally, the flatness of the Penholder function at distances far from the minimum means that the performance of each of these slows exponentially with distance.

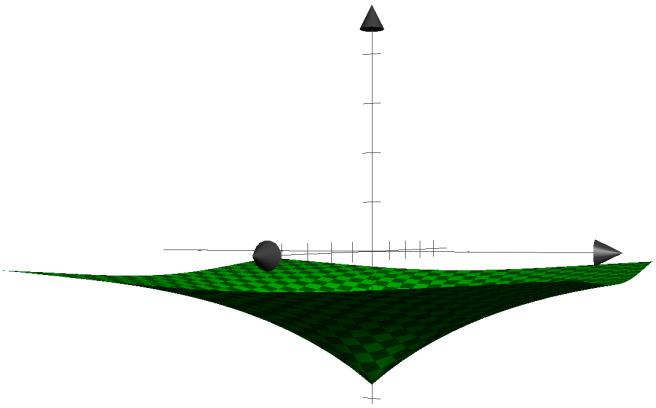

Figure 4. Penholder function

4.3. Flat, narrow basin: Booth function

Asymmetry between the x and y dimensions yields more possibilities for analysis. For this we use a variant on the Booth[1] function:

$$f(x, y) = (2x)^2 + (y - 1)^2 \tag{12}$$

Here we can see that Gradient Descent takes a longer path, perpendicularly intersecting each of the contour lines. RMSProp and AdaGrad take nearly identical, more direct paths; Adam slightly overshoots the minimum before correcting and taking a direct path (Fig. 6). When the gradient direction shifts, Gradient Descent with momentum oscillates significantly in the direction perpendicular to the gradient.

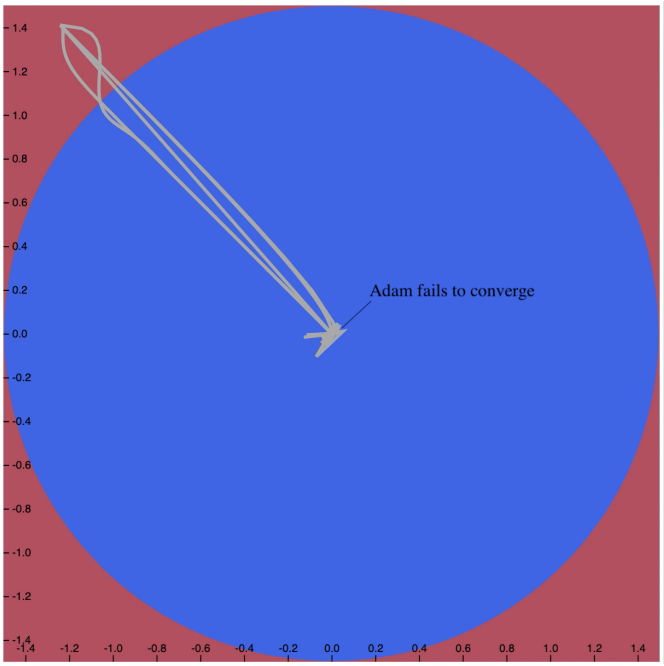

Figure 5. Algorithm paths on Penholder function

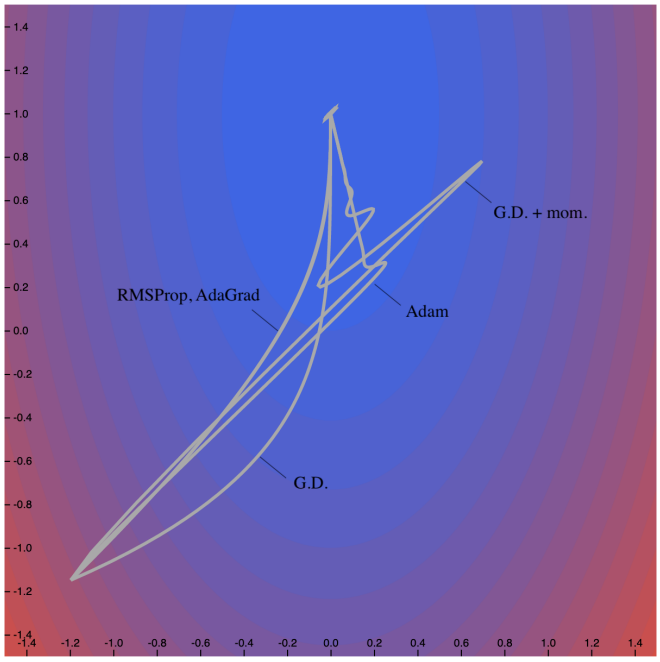

Figure 6. Algorithm paths on Booth function

4.4. Two global minima

We can also explore the performance of these algorithms on the function

$$f(x, y) = (x^2 + y^2)^2 - (x + y)^2 \tag{13}$$

which contains two global minima equidistant from the origin. When initialized roughly equidistant from the minima, the algorithms dont all find the same minimum (Fig. 7). When initialized along the axis of the minima, all algorithms find the closer minimum except for Gradient Descent with momentum, for which the momentum carries it out of the nearer minimum and into the further one. While RMSProp and AdaGrad appear to take a direct path, they take roughly twice as long as the other three functions to converge on both initializations.

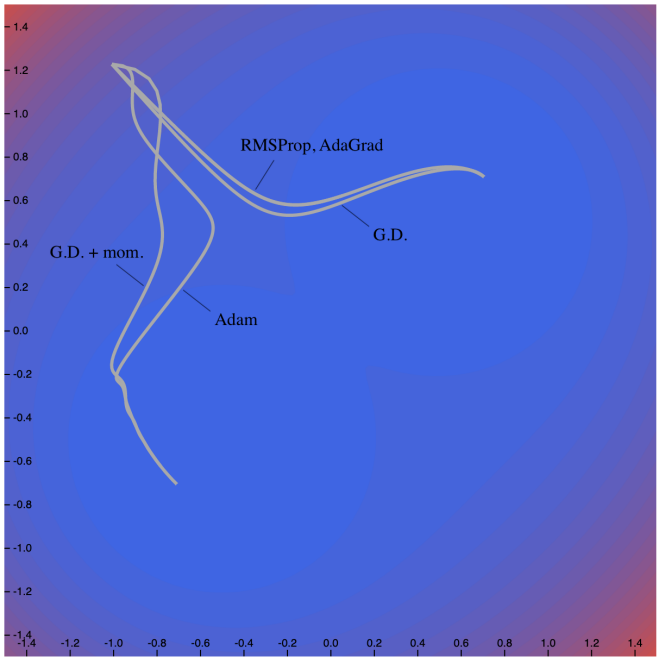

Figure 7. Convergence on two global minima

4.5. Local and global minima

We can define a similar function as the previous example, but include a $+x$ term that introduces one local and one global minimum (Fig. 8):

$$f(x, y) = (x^2 + y^2)^2 - (x + y)^2 + x \tag{14}$$

When initialized near the midpoint perpendicular to the axis of the minima, all of the algorithms are able to converge to the correct minimum. When initialized near the local minimum, Gradient Descent with momentum is now the only algorithm able to find the global minimum, due to the momentum carrying it in and out of the shallow basin; the others become trapped in the local minimum (Fig. 9). However, by changing the steepness of the minima or the distance of the initialization point along the axis of the minima, it is possible for Adam to reach the global minimum as well. For most initializations on this function, we can see that Adam and Gradient Descent with momentum seem to follow similar paths, and converge at similar rates. Vanilla Gradient Descent, RMSProp, and AdaGrad also follow similar paths to each other, but their rates of convergence are less similar (Fig. 10).

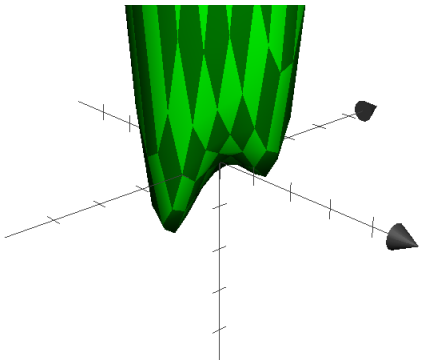

Figure 8. Local and global minima: $f(x, y) = (x^2 + y^2)^2 - (x + y)^2 + x$

4.6. Asymmetrical in x and y: Himmelblau function

We can also evaluate the performance of these algorithms on functions with multiple local minima unevenly distributed around x and y (Fig. 11). The Himmelblau[4] function is one example:

$$f(x, y) = (x^2 + y - 11)^2 + (x + y^2 - 7)^2 \tag{15}$$

We can see that the algorithms typically find the closest local minimum, regardless of whether it is the global optimum (Fig. 12).

4.7. More complex curves: Rastrigin function

The Rastrigin[9] function

$$f(x, y) = 20 * e^{-0.5\sqrt{0.2(x^2+y^2)}} - e^{0.5\cos(2\pi y)} \tag{16}$$

is another commonly-used test function for optimization (Fig. 13). It contains many grooves with local minima, but a central global minimum. We can see that performance on this function is highly dependent upon the initialization- for some initializations, all algorithms will converge to the true minimum (Fig. 14), sometimes only Gradient Descent with momentum and Adam will find the true minimum (Fig. 15), and sometimes all of the algorithms will get trapped in a local minimum.

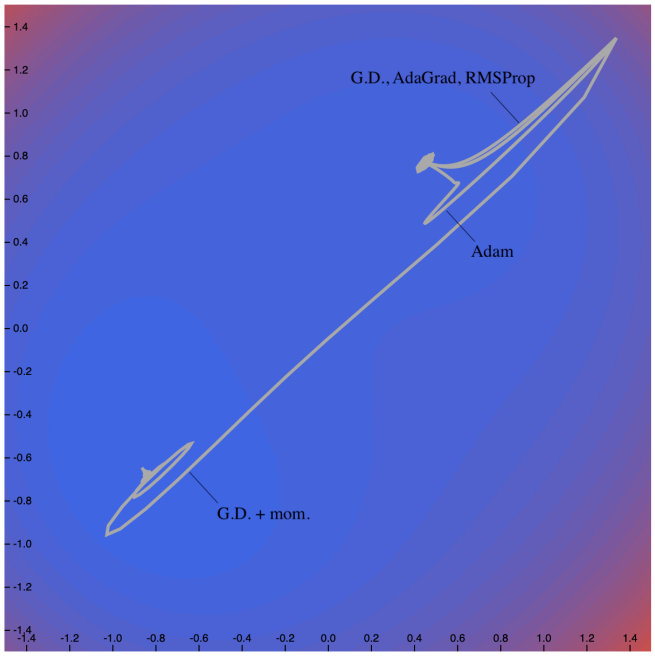

Figure 9. Gradient descent with momentum converges on global minimum

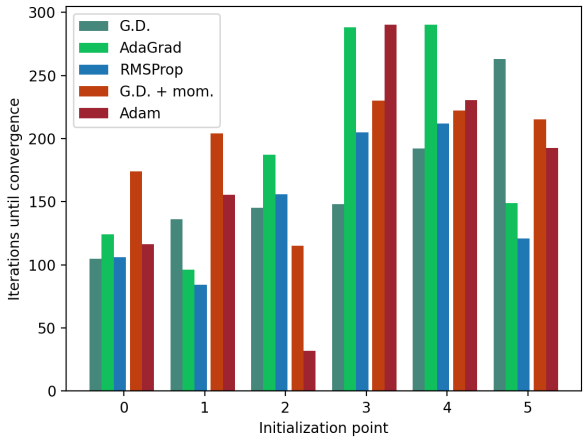

Figure 10. Iterations until convergence for six possible initializations on function with local and global minima

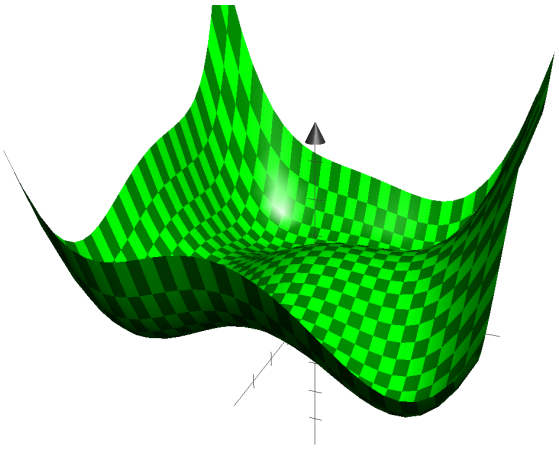

Figure 11. Himmelblau function

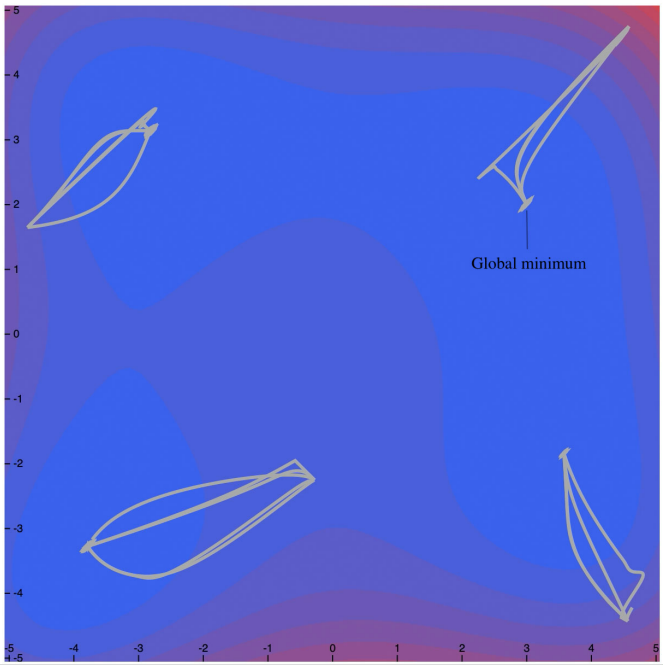

Figure 12. Paths of convergence to local minima for different initializations on Himmelblau function

5. Conclusion

These explorations demonstrate the utility of the tool in showing the general paths of convergence taken by these algorithms, as well as evaluating for which initializations each algorithm is likely to yield successful results on a given function. It also shows the importance of both function type and initialization location on the performance of these optimization methods. However, there are a few avenues that could be addressed in future work. Firstly, comparing the speed of convergence between algorithms is difficult given the differences in hyperparameters between them. Although commonly-accepted parameter values were chosen, the ideal parameter values for these various methods are dependent on the function under optimization. There is an additional feature (a toggle for which is not included in the interactive app) that will run each algorithm multiple times with different values for the hyperparameters, and return the

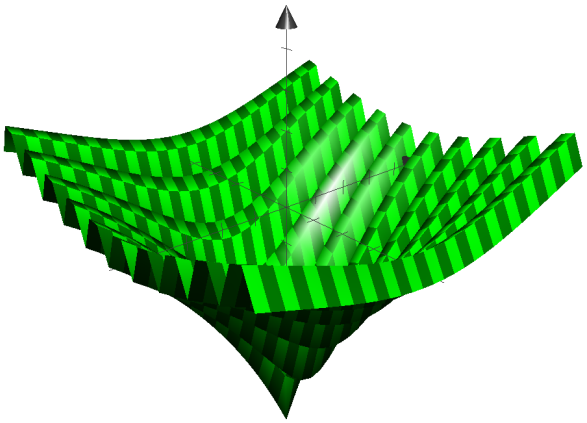

Figure 13. Rastigrin function

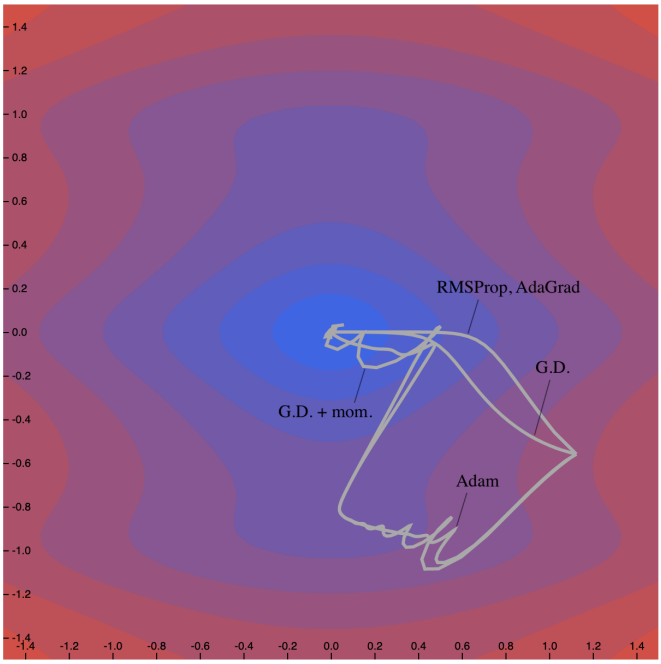

Figure 14. Paths of convergence to global minima on Rastigrin function

hyperparameter values that lead to the fastest convergence. Allowing the user to change these hyperparameters is one possible extension of this work, but there are lots of other features that affect the performance of optimization algorithms as well, such as introducing a decaying step size. The user experience could also be enhanced to make the

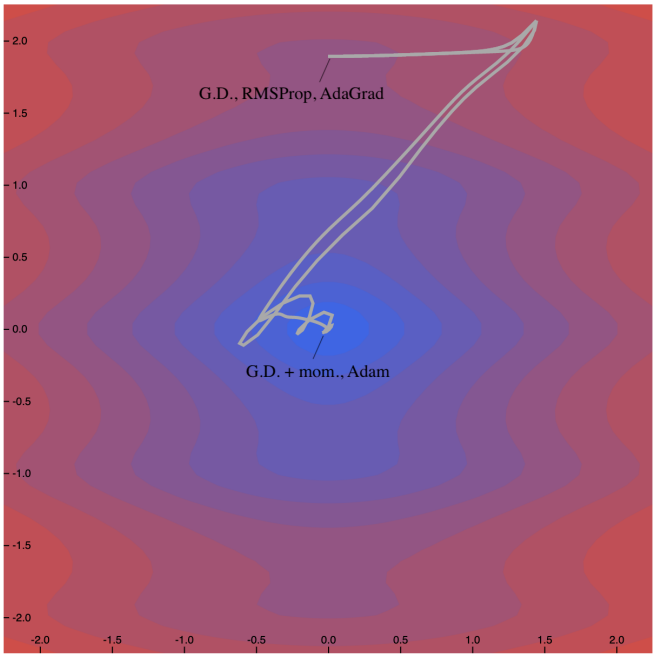

Figure 15. Paths of convergence to local and global minima on Rastigrin function

tool more robust, such as by expanding the view controls to allow for more functions to be effectively displayed. In summary, this work is able to provide novel interaction with optimization algorithm performance, but the complexities that dictate the performance of algorithms provides many possibilities for developing a more advanced system.

References

[1] Derek Bingham. Booth function, Jun 2013.

[2] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

[3] Emilien Dupont. Interactive visualization of optimization algorithms in deep learning, 2018.

[4] David M. Himmelblau. *Applied nonlinear programming*. McGraw-Hill, 1976.

[5] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[6] Mike Bostock. d3-contour.

[7] Sudhanshu K Mishra. Some new test functions for global optimization and performance of repulsive particle swarm method. *Available at SSRN 926132*, 2006.

[8] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[9] Leonard Andreevič Rastrigin. Systems of extremal control. *Nauka*, 1974.

[10] Tijmen Tieleman and Geoffrey Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*, 4(2):26–31, 2012.

[11] Jaewan Yun. Visualize gradient descent optimization algorithms in tensorflow, 2018.