A comparative evaluation of gradient-based optimization algorithms for training Terrain GAN

Kai Qin
Dept. of Electrical and Computer Engineering
University of Texas at Austin
Austin, TX
kai.qin@utexas.edu

Yi Han
Dept. of Electrical and Computer Engineering
University of Texas at Austin
Austin, TX
yh5598@utexas.edu

***Abstract*—This report is for the term project of EE381V Introduction to Optimization**

I. Introduction

GAN has been used in numerous senarios in industrial applications such as Anime generation, human-face generation, video generation. In this study, we focus on terrain generation using GAN especially in game terrain generation with training data from height map taken by satelite using radar sensors. The goal of this project can be separated into two parts. The first one is to learn different gradient-based optimization algorithms such as SGD, SGD with momentum, RMSProp, and ADAM. The second part is comparative evaluation of the performence of each optimizer.

ADAM was used in the original paper of DC GAN and LSGAN, and thus was our first choice optimizer. However,

II. SGD with momentum

Assuming the audiences are familiar with the general ideas of gradient descent, where we "decrease the loss function by moving in the direction of the negative gradient of an entire training set." "The vanilla gradient descent may be accelerated considerably by using Stochastic gradient descent to follow the gradient of randomly selected minibatches.""While SGD remains a very popluar algorithm, learning with it can sometimes be very slow. For example, let's say that you're trying to optimize a cost function which has contours like this image. The red dot denotes the position of the minimum. And we can see this up and own oscillations slows down the gradient descent. The basic idea of SGD with momentum, in one sentence, is to compute an exponentially decaying moving average of past gradients and continues to move in that direction. This method can help SGD to accelerate learning process. To understand how to compute the exponentially weighted averages of the last N iterations where N is determined by the hyperparameter alpha the algorithm, for example, when alpha equals .99, which is commonly used in practice, we are averaging over the last 100 iterations. Because after 100 days the weight becomes about a third of the weight of the current day. It is not a very accurage way to calculate the average value over the last N days, but it takes very little memory. The image on the right illustrates the effect of momentum, if you average out these gradients of the black arrows, you can find that the oscillations in the right diagonal direction will tend to average out to something closer to zero. In the mean time, it will keep the direvative on the left diagonal direction. So this allows the optimizer to take a more straight foward path or to damp out the oscillations in the path to the minimum."

The hyper prameters we have here are learning rate, beta, beta2, batch size. The SGD gradient estimator introduces a source of noise which makes the optimizer oscliates when we arrive around a minimum, thus it is common to decay the learning rate.

III. RMSProp

"there is another algorithm called RMSProp, which stands for root mean square prop, that can also speed up gradient descent. The algorithm is very similar to SGD with momentum, but instead of accumulating the gradient, we are taking average of the sqaured gradient and divide the gradient by the sqaure root of the average squared gradient. By dividing the average magnitude of the derivative in each dimenstion, the net effect is that the right diagonal derivative is devided by a relatively larger number and so that helps damp out the oscillations. Comparing to SGD with momentum, we can use a larger learning rate, and get faster learning without diverging the right diagonal direction, where in SGD with momentum, the learning rate is applying to all each dimension with same step size."

IV. Adam

By combining the ideas of momentum and RMSProp, we get Adam optimization algorithm which has been shown to work well across a wider range of deep learning architectures. In Adam, in each iteration, we first update the first moment estimate (the mean of the derivatives), which is exactly what we had when we're implementing momentum. And similarly, we do the rms prop to update the second moment estimate (the exp weighted average of the squared gradient). Then we do bias corrections. Finally, we compute the update by dividing the bias corrected first moment estimate with the sqaure root of the second moment estimate and reversing the sign. "This is commonly used learning algorithm that is proven to be very effective for many different neural networks of a variety

of architectures." Common choices for ro1, ro2, sigma are 0.9, 0.99 and 10 to -8 respectively. The common pratice for Adam hyperparameter tunning is to keep ro1 and ro2, and delta default and try a range of values of the learning rate to see what works best.

V. AdaGrad

AdaGrad can be understood as a special condition of RMSProp.

VI. Discussion about metrics

1 pixel is 1 square km; Original image pixel 21600x10800, and with 128x128 there are 14000, we excluded the ocean by excluding images with 90 percent pixels with max minors min value less than 0.05 (total range is from 0 to 1). And we end up with 4305. We have excluded flat land and waters, so we expect no such images from the output. We propose a metric of max -min / min as a metric.

Compare performence of each optimizer: we need a metric to tell if the generated image is good or bad. Based on our goal, we want noticeable feasures like mountains or lakes, valley, coasts. Compare min-max diff histogram of the trainning data and the output data.

Distribution distances: KL-divergence

VII. Results

Sometime it oscilates and don't converge. That's one of the big challenge of GAN training is that you don't get this clean monotonic improvment that your are used to with training supervised models.

"It is still an open probelm to have good metrics. If you had a really really good metric, you can propabally use it as a optimiztion critieria and optimize against it" "Assuming you don't do optimization directly on the Frechet Inception Distance, it can be a nice independent measure of the amount of variantion and cristness of the image generated ; when you do optimize against it, you will find results not exactly what you want" Adam 0.0002 lr and 16 batch size, we think based on the 3d model generated by this software looks realistic enough to meet our experience. We manually sweeped the lr rate and batch size using ADAM and collected the loss fuction over trainnign iterations. Next step will colecte results using other optimizers and figure out a metric to quantify the results we see. The discriminitor can always

VIII. Disctributed Term Paper

We implemented the Mendes-Herlihy algorithm and the Vaidya-Garg algorithm shown in Algorithm 1 and 3. Both algorithms are for multidimensional byzantine approximate agreement problem. The problem we are trying to analyze in this project is "the $Multidimensional\ Byzantine\ Approximate\ Agreement$ (MBA) for asynchronous systems. The problem requires processes to decide on multiple d-dimensional vectors in R^d, all within a fixed Euclidean distance ϵ of each other and inside the convex hull of d-dimensional vectors that were given by the non-faulty processes" [1] In an asynchronous system. It is well known according to FLP [2], byzantine binary consensus is impossible under a single crash failure. In the approximate agreement problem, not all processes are required to agree on the same output, but they all need to be within a fixed Euclidean distance ϵ. The following result was obtained by this paper [1] for the MBA problem, in a fully connected distributed system with up to f Byzantine failures:

– In asynchronous systems, $n > (d + 2)f$ is necessary and sufficient to solve the multidimensional approximate agreement problem. The assumptions for this algorithm includes FIFO, reliable communication channels, and fully connected distributed network. For solving MBA, a protocol must satisfy the following conditions given by this paper [1]:

–"$Validity$. The output vector at all non-faulty processes must be inside the convex hull of the input inputs. Formally, $O_i \in Poly(I_G)$ for any $pi \in G$.

–"$Termination$. Each non-faulty process must terminate within a finite amount of time.

Algorithm 1 The Mendes-Herlihy Algorithm

```
(R, v) ← CalculateRounds(I)
for m → 1,....,d do
    H← ∅
    r← 1
    while |H| ⩽ f do
        RBSend((p, m.r, v))
        upon V ← RBReceiveWitness(m.r) do
            S ← Safe(V)
            v ← v ∈ S such that v[m] = Midpoint(S(m))
            if r = R then
                RBSend((p, m.r, halt))
            r ← r + 1
        upon RBRecv((p', m.r', halt)), with r' ⩾ r do
            H ← H ∪ ((p', m.r', halt))
    end while
end for
return v
```

Algorithm 2 p.CalculateRounds(I)

```
RBSend((p, 0, I))
(V, W) ← (Val, Content(Wit)) fromRBReceiveWitness(0)
U ← barycenterofSafe_f(W') : W' ∈ W
v ← barycenterofSafe_f(U)
R ← ⌈log_2(√d/ε) max(σ_U(m) : 1 ≤ m ≤ d))⌉
return (R, v)
```

IX. Implementation Setup

We choose to implement a robot convergence problem to examine the message and time complexity of each algorithm.We setup the same environment used to test both algorithms with the following properties:

(1) There are total of 6 servers with 1 byzantine server.

Algorithm 3 The Vaidya-Garg Algorithm

$R \leftarrow 1 + \log_{1/(1-\gamma)} \frac{\sqrt{d}(U-v)}{\epsilon}$
for r $\rightarrow$ 1,.... R **do**
  RBSend$((p, r, v))$
  upon M $\leftarrow$ RBReceiveWitness(r) **do**
    for $M' \subseteq M, |M'| = n - f$ **do**
      $S_{M'} \leftarrow Safe_f(M')$
      Z $\leftarrow$ Z $\cup$ DeterministicallyChoosePoint$(S_{M'})$
    end for
    v $\leftarrow (\Sigma_{z \in Z} z)\ /|Z|$
end for
return v

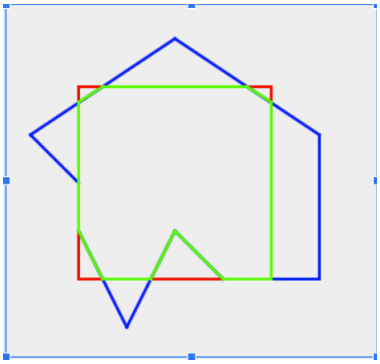

Fig. 1. Example results of take intersection of two convex hull.

(2) Each server is fully connected to other servers with TCP connections.
(3) Each server has an input of 2 dimensional vector that initially form a convex shape.
(4) The byzantine server will initially send random input vector values to all other servers.
(5) When implementing $if\ never\ sent$ in function $RBEcho_f$, we saved the sender's pid as key in a hash table and save the number of time we sent the corresponding message as value in the hash table.
(6) Since we are working on the 2 dimensional case of the MBA problem, we wrote our $Safe_f$ function based on the Safe Area definition, where we take intersection (see fig 1 of an exemple of how we took intersections [6])of any two $Restrict_f$ of the input message set and returns the vertices of the resulting convex hull.
(7) In line 9 of algorithm 6, the process computes, $S(m)$ the projection of the safe area on coordinate m, then choose a point in the safe area such that its m-th coordinate is in the midpoint of $S(m)$. Here we choose the other coordinate such that it is in the midpoint of the interval of the safe area as well.

X. Message Complexity Comparison

In MH algorithm, each non-faulty process spends $O(n^2)$ messages for reliable broadcast and the witness algorithm for each round. The overall message complexity for all the rounds is

$$O(n^2 d \log(d/\epsilon\ max(\sigma_{I_G}(m) : 1 \leq m \leq d)). \quad (1)$$

In VG algorithm the overall message complexity for all the rounds is

$$O(n^2 d \log_{1/(1-\gamma)}(d/\epsilon\ max(\sigma_{I_G}(m) : 1 \leq m \leq d)). \quad (2)$$

XI. Experimental Time Comparison

For live demo time result comparison, We setup a millisecond time counter that began at the initiation phase of running the algorithms, and logs the time elapsed. For the MH algorithm, initial calculate round function requires around 3000 milliseconds to complete. Both algorithms depend on reliable broadcast and receive witness, thus their time for each round is similar at around 3000 to 4000 milliseconds. The number of rounds needed for The VG algorithm is significantly higher than the MH algorithm, therefore the completion of MH algorithm is faster than the VG Algorithm. Since Our implementation of each process is thread based, accuracy of the logged elapsed time may not be true reflection of a real world implementation of these algorithms.

XII. Algorithm Comparison Observations

According to the original paper, "MH [**?**] algorithm has an asymptotically constant convergence factor in relation to n, but requires certain geometrical operations too restrictive for applications such as robot navigation. In such application, instead of exchanging messages, processes obtain information via visual sensors, projecting them over private coordinate systems. The VG [4] algorithm is better suited for the application above, at the cost of a slower convergence,now asymptotically dependent on n,the number of processes."

XIII. Future Works

While implementing the two algorithms, we found it useful to create more comparative methods to show the convergence factor differences. More visualization tools would be useful in further displaying convergence comparisons. During the implementation of this project, we focused on the 2-dimensional scenario, in the next step, we can further expand on the dimensions based on the linear programming algorithm provided in the appendix of the paper.
Robot convergence is one of the more physically tangible application of byzantine vector consensus, other problems such as distributed voting can equally utilize the agreement.

References

[1] Mendes H, Herlihy M, Vaidya N, Garg VK. Multidimensional agreement in Byzantine systems. Distributed Computing. 2015 Dec 1;28(6):423-41.
[2] Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Trans. Program. Lang. Syst. 4(3), 382–401 (1982).
[3] Mendes, H., Herlihy, M.: Multidimensional approximate agreementin-byzantineasynchronoussystems. In:Proceedings of the 45th annual ACM symposium on Symposium on theory of computing, STOC '13, pp. 391–400. ACM, New York, NY, USA (2013).

[4] Vaidya,N.,Garg,V.K.:Byzantine vector consensus incomplete graphs. In: Proceedings of the 2013 ACM symposium on Principles of distributed computing, PODC '13. ACM, New York, NY, USA (2013)
[5] Potop-Butucaru, M., Raynal, M., Tixeuil, S.: Distributed computing with mobile robots: An introductory survey. In: 14th International Conference on Network-Based Information Systems (NBiS), pp. 318 –324 (2011).
[6] Intersection of Convex Polygons Algorithm, by By Sinan Oz, https://www.swtestacademy.com/intersection-convex-polygons-algorithm/
[7] The Centroid of Convex Polygons by Gilles Bellot, https://bell0bytes.eu/centroid-convex/