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

I. Introduction

Generative Adversarial Networks (GAN) [1] have been wildy used in generative applications such as anime generation, human-face generation, and video generation. In this study, we focus on evaluating the performence of three gradient-based optimization algorithms in training GANs for procedural terrain generation [2]. ”Procedural terrain generation for video games has been traditionally done with smartly designed but handcrafted algorithms that generate hightmaps” [2] $\{DESCRIBE\ dataset\}$ The first goal of this project is for us as a term project to extend what we have learned in class about gradient-based optimization algorithms and to dive deeper in this topic by experimenting with three state-of-the-art gradient based algorithm for training neural networks. The second goal is to explore the practical differences that each optimizer can make for TerrianGAN. The third goal is to understands the results of fine-tuning parameters on the different optimizers and their effects on TerrianGAN.

II. Gradient-based optimization algorithms review

Before we dive into the evaluation of three gradient-based optimizers we compared in this study, SGD with momentum, RMSProp, and ADAM, we want to provide a review of these three algorithms. We will start from the classic gradient descent algorithm. In one sentence, the general idea of the classic gradient descent algorithm is that at every iteration, the desired parameters are moving in the direction of the negtive gradient of the objective function based on the entire training dataset to decrease the loss function. The vanilla gradient descent may be accelerated considerably by using stochastic gradient descent (SGD) which follows the gradient of randomly selected minibatches. Even though SGD ingtroduced this very important idea of training with minibatches, learning with it can sometimes be very slow. For example, let’s say that you’re trying to optimize a cost function which has a contour in image 1. The red dot denotes the position of the minimum. And we can see this up and own oscillations in black lines slows down the gradient descent. Therefore, it’s rear to see large scale neural network training with classic SGD. One improvement made to SGD is by adding momentum.The basic idea of SGD with momentum is to compute an exponentially

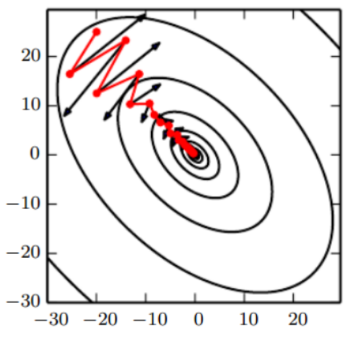


Fig. 1. Visualization of the effect of momentum [3]

```
**Require:** Learning rate $\epsilon$, momentum parameter $\alpha$.
**Require:** Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  **while** stopping criterion not met **do**
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)};\boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha\boldsymbol{v} - \epsilon\boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  **end while**
```

Fig. 2. SGD with momentum [3]

decaying moving average of the past gradients and continues to move in that direction [3]. The velocity update step in algortihm 2 shows how to compute the exponentially weighted averages of the last N iterations, where N is determined by the hyperparameter alpha. For exambple, when alpha equals 0.99, which is commonly used in practice, we are approximately averaging over the last 100 iterations. This method can only approximate the average value over the last N days, but it takes very little memory. Image 1 illustrates the effect of momentum. If you average out these gradients of the black arrows, you can find that the oscillations in the forward diagonal direction will tend to average out to something closer to zero. In the mean time, it will keep the direvative on the backward 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.

Root mean squared prop (RMSProp) is another algorithm

```
Require: Global learning rate $\epsilon$, decay rate $\rho$.
Require: Initial parameter $\boldsymbol{\theta}$
Require: Small constant $\delta$, usually $10^{-6}$, used to stabilize division by small
  numbers.
  Initialize accumulation variables $\boldsymbol{r} = 0$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Accumulate squared gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1-\rho)\boldsymbol{g} \odot \boldsymbol{g}$
    Compute parameter update: $\Delta\boldsymbol{\theta} = -\frac{\epsilon}{\sqrt{\delta + \boldsymbol{r}}} \odot \boldsymbol{g}$.   ($\frac{1}{\sqrt{\delta + \boldsymbol{r}}}$ applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while
```

Fig. 3. RMSProp [3]

that can speed up the classic gradient descent. This algorithm (see figure in 3) is very similar to SGD with momentum, but instead of accumulating the gradient, we are taking the average of the sqaured gradient and divide the gradient by the sqaure root of the average squared gradient [3]. By dividing the average magnitude of the derivative in each dimenstion, the net effect is that the forward diagonal derivative in image 1 is devided by a relatively larger number, and it therefore helps damp out the oscillations. Comparing to SGD with momentum, we can use a larger learning rate, and get faster learning without diverging in the forward diagonal direction, where in SGD with momentum, the learning rate is applying to all each dimension with same step size.

By combining both 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, in the first highlighted equation in figure 4 we first update the first moment estimate (the mean of the derivatives) , which is exactly what we had when we're implementing SGD with momentum. And similarly, we do the rms prop to update the second moment estimate in the second equation. 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. Common choices for ρ_1, ρ_2, σ are 0.9, 0.99 and 10^{-8} respectively. The common pratice for Adam hyperparameter tunning is to keep ρ_1, ρ_2, and σ default and try a range of values of the learning rate to see which works the best.

III. Terrain GAN review

Traditionally, video game terrains have been either manually generated or procedually gendreatd by algorithms designed to mimic real-life terrains such as mountains, lakes, and coasts. These methods are capable of generating high quality terrains but also come with drawbacks such as high expense of human labour and the lack of flexibility and complexity of the nature. With the ability to recover the training data distribution [1], GAN becomes the perfect algorithm for this task. A first step towards conquering this problem using GAN has been proposed in [2], where DCGAN and LSGAN have been applied to generate high quality heightmaps (see image 5) from high-resolution earth heightmap data provided by NASA.

```
Require: Step size $\epsilon$ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, $\rho_1$ and $\rho_2$ in $[0, 1)$.
  (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant $\delta$ used for numerical stabilization. (Suggested default:
  $10^{-8}$)
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \boldsymbol{0}$, $\boldsymbol{r} = \boldsymbol{0}$
  Initialize time step $t = 0$
  while stopping criterion not met do
    Sample a minibatch of $m$ examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with
    corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m}\nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1-\rho_1)\boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1-\rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1-\rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1-\rho_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$   (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while
```

Fig. 4. ADAM [3]

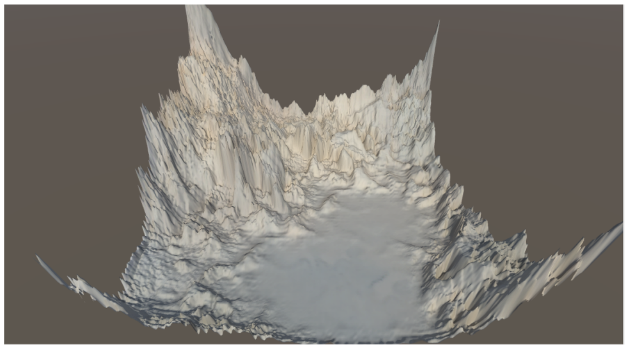


Fig. 5. A redering of a generated heightmap in [2]

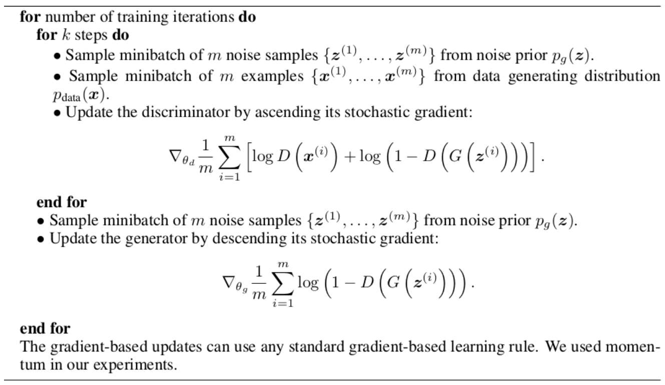
In our project, we focus on evaluating the effects of different optimizers on LSGAN since it is suggested in the paper [2] that LSGAN provide better training stability than DCGAN. One novelty of GAN is learning the training data distribution via an adversaria process. The training process consists of two steps as shown in the psudocode in figure 7, where we first train the discriminator$G$ for k steps, and then train the generator D for one step. The number of steps to apply to the discriminator, k, is a hyperparameter. As in the GAN paper, we used $k = 1$, the least expensive option, in all of our experiments in the next section.

DCGAN and LSGAN are two variants of GAN, where DCGAN improves the orginal GAN with fine-tuned architecture details and showed us that GAN is capable of generating perceptually good samples (see figure 6 and interpolations [4]. And LSGAN adopted the least squares loss instead of the cross entropy as the objective function since the original loss function may lead to the vanishing gradients problem.

IV. Experiments and results

In this project, the training data are 128px height maps from the original NASA image As we can see in the graph on the right, the loss function of both G and D oscilate and don't converge. That's one of the big challenge of GAN training

Fig. 6. DCGAN samples on faces [3]



```
for number of training iterations do
  for $k$ steps do
    • Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Sample minibatch of $m$ examples $\{x^{(1)}, \ldots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
    • Update the discriminator by ascending its stochastic gradient:
        $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right) \right].$
  end for
  • Sample minibatch of $m$ noise samples $\{z^{(1)}, \ldots, z^{(m)}\}$ from noise prior $p_g(z)$.
  • Update the generator by descending its stochastic gradient:
        $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$
end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
```

Fig. 7. Minibatch stochastic gradient descent training of GAN [1]

is that you don't get this clean monotonic improvment that your are used to with training supervised models.In GANs, the objective function for the generator and the discriminator usually measures how well they are doing relative to the opponent. For example, we measure how well the generator is fooling the discriminator.

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

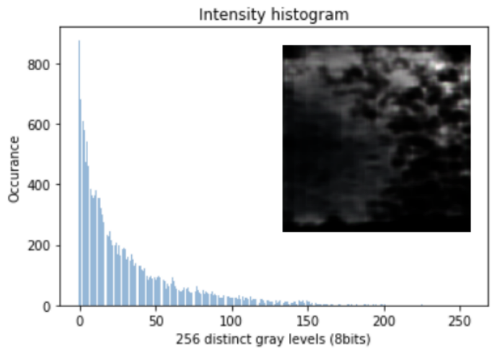


Fig. 8. Image intensity histogram of a generated height map.

V. Discussion about metrics

As we have seen in the loss graphs, the oscilation of the loss makes it harder to measrue the training progress. Currently, it is still an open probelm to find good metrics for GAN training. On the other hand, If you had a really really good metric, you can propabally use it as a optimiztion critieria and optimize against it. Inception score [?] and Frechet Inception Distance score (FID) [?] are two state of the art metrics people use to measure the GAN performence. However, both metrics don't apply to the training data we have. Inception score requires categorical labeled data. FID requires a pre-trained classification neural network. Inspired by the goal of gan which is to learn the probability distribution from the training data, and implicitly represent it using neurual network, we propose a new metric to measure the Terrain GAN performance, the KL-divergence between normalized intensity histogram of training images and generated images.

Before we introduce how we calculate this metric, we need first introduce the idea of image intensity histogram, which is a widly used technique in image processing. The definition is that the histograms plots how many times each intensity value in an image occurs. Here we have a example of intensity histogram of a generated hight map in figure 8.

However, we want the distribuion over a large set of images, the original intensity historgram doesn't work here, so we calculate the average occurrences of each intensity value and divide it by the total number of pixels over all images, and define it the normalized histogram. Image 9 shows the normalized itensity histogram of the training images, and image 10 shows the normalized itensity histogram of 50 generated image using the neural net trianed by Adam with lr .0002 and batchsize 16 for 150 epchos. We can see that they both have mean around 30 gray levels.

Finally, Here is the KL divergence equation. In our case the P(x) is the real dist and QX is the generated hight map. From the normalized histogram we can tell that the ADAM has more overlap than the sgd. This also matches what we see from the generated images. And the KL divergence value confirmed our observations. The SGD with this set of hyper prameter has no overlap with the training data thus the KL

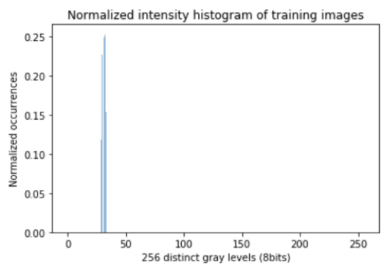


Fig. 9. Normalized image intensity histogram of the training dataset.

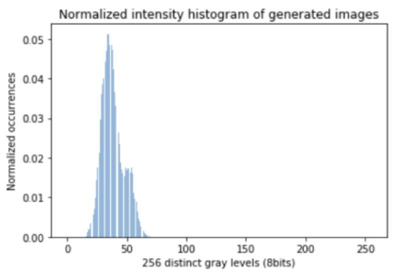


Fig. 10. Normalized image intensity histogram of generated images.

div is +inf.

$$D_{KL}(P||Q) = \sum_{x \in \chi} P(x) log(P(x)/Q(x)$$

Here is a bar plot of the KL divergence of different hyperprameter settings of adam after 150 ephch. Visually, we can see that the higher the metric, the harder you can provide some geographic meaning to the generated images.

Again, we have all the results of the lowest KL divergence model.

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

VI. Conclusion

VII. Future Works

References

[1] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial networks,” 2014.

[2] C. Beckham and C. Pal, “A step towards procedural terrain generation with gans,” 2017.

[3] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[4] A. Radford, L. Metz, and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” 2015.