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 optimizer 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 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 provides a very important idea, 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. Thus, it's rear to see large scale neural network training with the classic SGD. That's why SGD with momentum was introduced. The basic idea of SGD with momentum is to compute an exponentially decaying moving average of past gradients and continues to move in that direction [**?**]. The exponentially weighted averages of the last N iterations where N is determined by the hyperparameter alpha the algorithm, for exambple, when alpha equals 0.99, which is commonly used in practice, we are approximately 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.

Root mean squared prop (RMSProp) is another algorithm that can speed up classic gradient descent. This algorithm 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. 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."

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

III. GAN, DCGAN, and LSGAN review

GAN was originally proposed by Ian Goodfellow in [1]

IV. Experiments and results

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 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

V. Discussion about metrics

As we have seen in loss graph, the oscilation of the loss makes it harder to measrue the training progress. Currently, It is still an open probelm for gan training to have good metrics. 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 FID are two state of the art metric people use to measure the performence. However, both metrics don’t apply to the data we. Inception score requires categorical labeled data. The Frechet Inception Distance score (FID) requires a pre-trained classification neural network. Since the goal of gan is to learn the probability distribution from the training data, and implicitly represent it using neurual network. Thus we have proposed a new metric to measure the Terrain GAN performance, the KL-divergence between normalized intensity histogram of training images and generated images.

This is how way calculate this metric, first: Image intensity histogram, which is a widly used technique in image processing. The definition is that Histograms plots how many times (frequency) each intensity value in an image occurs. Here we have a example of intensity histogram of a generated hight map.

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. The top image shows the normalized itensity histogram of the training images. The bottom image 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 div is +inf.

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.