Paper Summary Top-k Training of GANs : Improving GAN Performance by Throwing Away Bad Samples

Abstract

We will summarize the paper : Top-k Training of GANs : Improving GAN Performance by Throwing Away Bad Samples.

1. The Proposed Method

This paper presents a simple modification (one line of code) to the standard GAN learning algorithm that improves the results without increasing the computational cost. The researchers perform what is called a "**top k update**": they update the generator parameters only on the samples of the mini-batch that the critic considers the most realistic.

Thus at each update they simply eliminate the gradient contributions of the batch elements that the critic considers as "less realistic".

They mainly do two studies:

- Why the method works
- Analysis of the results on the CIFAR and ImageNet datasets

1.1. Background

Generative Adversarial Networks: A Generative Adversarial Network (GAN) is composed of a generator, G, and a critic, D, where in practice both G and D are neural networks. The backpropagation formulas for the generator G and the critic D used in the paper are given here :

$$\theta_D \leftarrow \theta_D + \alpha_D \sum_X \nabla_D V(D,G)$$

$$\theta_G \leftarrow \theta_G - \alpha_G \sum_Z \nabla_G V(D,G)$$

with G the generator, D the critic and $V(D,G)$ is a loss function, $X = \{x_i \sim p(x), i = 1, ..., B\}$, $p(x)$ the target distribution, $Z = \{G(Z_i), z_i \sim p(z), i = 1, \ldots, B\}$, B mini batches of samples, $G(z), z \sim p(z)$ the generator output distribution and $\alpha_D(\theta_D)$, $\alpha_G(\theta_G)$ are the learning rates for the critic and generator respectively. Intuitively, the generator is trained to "trick" the critic into being unable to correctly classify the samples by their true output distributions.

1.2. Top-k Training of GANs

The researchers propose a simple modification to the GAN training procedure. When they update the generator parameters on a mini-batch of generated samples, they simply zero out the gradients of the mini-batch elements corresponding to the least critical outputs.

And this is done through the **top k operation**: given a collection of scalar values, it retains only the k elements of that collection that have the highest value.

Mathematically, they only change the formula for updating the generator:

$$\theta_G \leftarrow \theta_G - \alpha_G \sum_Z \nabla_G V(D,G)$$

to :

$$\theta_G \leftarrow \theta_G - \alpha_G \sum_{\max_k\{D(Z)\}} \nabla_G V(D,G)$$

with $D(Z)$ is shorthand for the critic's output for all entries in the mini-batch Z.

1.2.1. Annealing k

They define $k = B$, where B is the full batch size, at the beginning of the learning (early on in training, the critic may not be a reliable scoring function for samples from the generator), and they gradually reduce it over the course of learning. In practice, we decrease k by a constant factor, γ, in each training epoch until a minimum of $k = \nu$ is reached. They use the minimum value ν so that training does not progress to the point of having only one element in the mini-batch.

2. Mixture of Gaussians

To better understand the behavior of GAN with their modification, they train the GAN **top- k training** on a "toy

dataset": they set the target distribution to be a mixture of 2D isotropic Gaussians with a constant standard deviation of 0.05 and means evenly spaced on a 2D grid.

This allows them to identify two important things:

1. **top-k training of GANs can reduce dropping mode** (i.e., learning to generate only a subset of the individual components of the mixture) and improve the sample quality in this context. Indeed, when increasing the number of modes in the target distribution, **top-k learning** is able to improve both the fraction of recovered modes and the fraction of high-quality samples: The fact that the number of modes recovered by performing top-k training is larger than the number of modes recovered without top-k training shows that top-k training can help mitigate mode dropping.

2. **When gradient updates are performed on the bottom-k elements rather than the top-k elements, samples tend to move away from their closest mode**

The strategy used by the researchers is defined here:

- For each of these samples, they compute the direction of the nearest mode, which we call the oracle direction2.
- Then, with these oracle directions as a reference point, we compare the top-k update and the 'bottom-k' update, which respectively update the generator using only the critic's predictions.
- After performing a gradient descent step, we then measure the motion of the samples after the update steps. By isolating the effect of a gradient step, we can understand what happens when the generator is updated using the "bad" samples versus what happens when it is updated using the "good" samples.

In order to understand why updating on the worst samples is detrimental, they evaluate the cosine similarity between the oracle direction and the displacement calculated above, for each sample. By evaluating the cosine similarity, they roughly measure the quality of the gradient update: Roughly speaking, the closer the cosine similarity is to 1, the better the update is, since a value of 1 means that the points are pushed in the exact direction of the mode. The closer the cosine similarity is to -1, the worse the gradients are, since a value of -1 means that the points are pushed in the opposite direction of the closest mode.

This experiment gives a somewhat surprising result when only the bottom-k update is performed: The cosine similarity between the update direction and the oracle direction is negative in this case, even for samples that we consider to be high-quality samples. This suggests that the points are actively pushed away from the mode they are already close to. For samples that are very close to the nearest mode, the cosine similarity is less important since these samples are already "good".

This experiment further shows how the bottom-k samples actively result in a worse generator after an update. Finding the mean gradient signal from the full batch will result in the added negative influence from the bottom-k samples; their method reduces the negative effects by simply discarding the bottom-k samples, which is a computationally efficient, effective and easy-to-implement solution.

This phenomenon suggests a mechanism by which the **top-k training** improves the performance of GAN: it does not use these "useless" gradients in its mini-batch stochastic estimation of the full gradient.

3. Experiments on Image Datasets

The proposed modification for GANs significantly improves the results on the CIFAR and ImageNet datasets.

By leaving the original hyperparameters fixed, the **top-k training** is an "immediate" improvement for each of the GANs tested. An important remark is that they have better performances for the simple GANs (DCGAN and WGAN) than for the advanced architectures (self attention GAN or spectral normalization GAN): maybe because there is less room for improvement on the more sophisticated models.

For the case of **Anomaly Detection**: they observed a clear improvement of the results when testing on different GANs, the **top-k training** improved the performances in all the contexts where we tested it.

Similarly for **batch size**, they obtain the same conclusion: this demonstrates once again the usefulness of **top-k training**, as it is able to improve GAN learning for variable batch sizes, without modifying any hyperparameter.

With all these tests, the researchers realize two important things:

- Using too small a value of γ hurts performance by discarding too many samples at the beginning of training.
- Using too large a value for ν degrades performance, because if ν is too large, then too few samples are discarded, and top-k training becomes similar to normal training.

4. Conclusion

This paper, using a very simple method (by modifying one line of code), achieves improvements on a wide variety of **GAN architectures**.