Top-k Training of GANs: Improving Generators by Making Critics Less Critical

Samarth Sinha [1] **Anirudh Goyal** [2] **Colin Raffel** [3] **Augustus Odena** [3]

Abstract

We introduce a simple (one line of code) modification to the Generative Adversarial Network (GAN) training algorithm that materially improves results with no increase in computational cost: When updating the generator parameters, we simply zero out the gradient contributions from the elements of the batch that the critic scores as 'least realistic'. Through experiments on many different GAN variants, we show that this 'top-k update' procedure is a generally applicable improvement. In order to understand the nature of the improvement, we conduct extensive analysis on a simple mixture-of-Gaussians dataset and discover several interesting phenomena. Among these is that, when gradient updates are computed using the worst-scoring batch elements, samples can actually be pushed further away from the their nearest mode.

1. Introduction

Generative Adversarial Networks (GANs) (Goodfellow et al., 2014) have been successfully used for image synthesis (Miyato et al., 2018; Zhang et al., 2018; Brock et al., 2018), audio synthesis (Donahue et al., 2018b;a), domain adaptation (Zhu et al., 2017; Zhang et al., 2017), and other applications (Xian et al., 2018; Ledig et al., 2017). It is well known that GANs are difficult to train, and much research focuses on ways to modify the training procedure to reduce this difficulty. Since the generator parameters are updated by performing gradient descent through the critic, much of this work focuses on modifying the critic in some way (Arjovsky et al., 2017; Mao et al., 2017; Nowozin et al., 2016; Gulrajani et al., 2017) so that the gradients the generator gets will be more 'useful'. What 'usefulness' means is generally somewhat ill-defined, but we can define it implicitly and say that useful gradients are those which result in the generator learning a better model of the target distribution.

Recent work by Wu et al. (2019) suggests that gradients can be more useful when computed on samples closer to the data-manifold – that is, if we tend to update the generator and critic weights using samples that are more realistic, the generator will tend to output more realistic samples. Wu et al. (2019) achieves state-of-the-art results on the ImageNet conditional image synthesis task by generating samples from the generator, computing the gradient of the critic with respect to the sampled prior that generated those samples, updating that sampled prior in the direction of that gradient, and then finally updating the generator parameters using this new draw from the prior. In short, they update the generator and critic parameters using a z' such that the critic thinks $G(z')$ is 'more realistic' than $G(z)$. However, this procedure is complicated and computationally expensive: it requires twice as many operations per gradient update. In this work, we demonstrate that similar improvements can be achieved with a much simpler technique: we propose to simply zero out the gradient contributions from the elements of the batch that the critic scores as 'least realistic'.

Why should this help? In an idealized GAN, the trained critic would slowly lose its ability to tell which inputs were samples from the generator and which inputs were elements of the target distribution, but in practice this doesn't happen. Azadi et al. (2018) show that a trained critic can actually be used to perform rejection sampling on a trained generator and significantly improve the performance of the trained generator. Thus, as training progresses, the critic can serve as a useful arbiter of which samples are 'good'. Then, if we accept the premise that updating on 'good' samples improves GAN training, we should be able to use the critic during training to make decisions about which samples to update on. But why should we accept this premise? Why would updating on the 'bad' samples hurt instead of helping? In this work, we provide a partial answer by showing that in practice, gradient updates derived from samples the critic deems 'bad' can actually point *away* from the true data manifold.

Since the critic's ability to tell us which samples are bad improves during training, we anneal the fraction of the batch

[1]University of Toronto [2]Mila, Université de Montréal [3]Google Brain. Correspondence to: Samarth Sinha <samarth.sinha@mail.utoronto.ca>, Augustus Odena <augustusodena@google.com>.

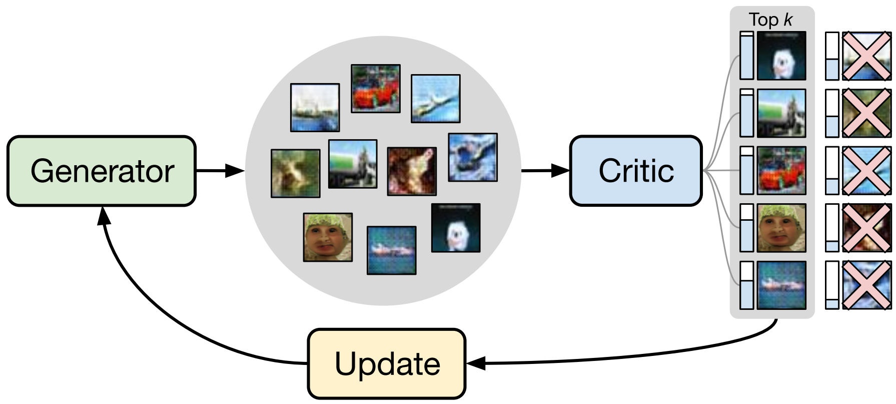

Figure 1. Diagram of top-k training of a GAN. The generator generates a batch of samples, which are scored by the critic. Only the k samples with the highest scores are used to update the generator.

that is used for updates as training progresses. In the beginning of training, we use samples from the entire batch, and gradually reduce k after each training epoch.

Our contributions can be summarized as follows:

- We propose a simple 'one-line' modification to the standard GAN training algorithm that only updates the generator parameters on the samples from the mini-batch that the critic scores as most realistic.
- We thoroughly study (on a 'toy' dataset) the mechanism by which our proposed method improves performance and discover that gradients computed on discarded samples would point in the 'wrong' direction.
- We conduct further experiments on the CIFAR (Krizhevsky et al., 2009) and ImageNet (Russakovsky et al., 2015) datasets and show that our proposed modification can significantly improve FID (Heusel et al., 2017) numbers for several popular GAN variants.

2. 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 generator takes as input a sample z from a simple prior distribution $p(z)$ and is trained so that its output appears indistinguishable from a sample from the target distribution $p(x)$. The critic is trained to be able to *discriminate* whether a sample is from the target distribution, $p(x)$ or from the generator's output distribution $G(z), z \sim p(z)$. Both networks are trained via a min-max game $\min_G \max_D V(D,G)$ where $V(D,G)$ is a loss function. For example, as originally proposed in (Goodfellow et al., 2014), $V(D,G) = \mathbb{E}_{x\sim p(x)}[\log D(x)] + \mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z)))]$. Many alternate formulations of $V(D,G)$ have been proposed; for a survey see (Kurach et al., 2019). In practice, mini-batches of B samples $\mathcal{X} = \{x_i \sim p(x), i = 1,\ldots,B\}$ and $\mathcal{Z} = \{G(z_i), z_i \sim p(z), i = 1,\ldots,B\}$ are used in alternating stochastic gradient descent to relax the minimax game:

$$\theta_D \leftarrow \theta_D + \alpha_D \sum_{\mathcal{X}} \nabla_{\theta_D} V(D,G) \qquad (1)$$

$$\theta_G \leftarrow \theta_G - \alpha_G \sum_{\mathcal{Z}} \nabla_{\theta_G} V(D,G) \qquad (2)$$

where α_D (θ_D) and α_G (θ_G) are the learning rates (and parameters) 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.

Fréchet Inception Distance: Heusel et al. (2017) proposed Fréchet Inception Distance (FID) as a metric to measure how well a generative model has fit an target distribution. The metric utilizes an internal representation from a pre-trained Inception classifier (Szegedy et al., 2017) and measures the Fréchet distance from the target distribution $p(x)$ to the generated distribution $G(z)$ (Dowson & Landau, 1982). The FID score is calculated by:

$$||m - m_w||_2^2 + Tr(C + C_w - 2(CC_w)^{1/2})$$

where m and C are the mean and co-variances of the Inception embeddings for real-data, and m_w and C_w are the mean and covariance matrix of the Inception embeddings for the generated samples. In practice, 50,000 generated samples are used to measure the FID of a GAN.

Top-k Operation: The top-k operation does what its name suggests: given a collection of scalar values, it retains only the k elements of that collection that have the highest value. We use $\max_k\{Q\}$ to denote the largest k elements from a set Q of scalars.

3. Top-k Training of GANs

3.1. The Proposed Method

We propose a simple modification to the GAN training procedure. When we update the generator parameters on a mini-batch of generated samples, we simply zero out the gradients from the elements of the mini-batch corresponding to the lowest critic outputs. More formally, we modify the generator update step from Equation 2 to

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

where $D(\mathcal{Z})$ is shorthand for the critic's output for all entries in the mini-batch $\mathcal{Z}$. Intuitively, as training progresses, the critic, D, can be seen as a scoring function for the generated samples: a generated sample that is close to the target distribution will receive a higher score, and a sample that is far from the target distribution will receive a lower score. By performing the top-k operation on the critic predictions, we are only updating the generator on the 'best' generated samples in a given batch, as scored by the critic. A diagram of our approach is shown in Fig. 1.

3.2. Annealing k

Early on in training, the critic may not be a reliable scoring function for samples from the generator. Thus, it won't be helpful to throw out gradients from samples scored poorly by the critic at the beginning of training – it would just amount to throwing out random samples, which be roughly equivalent to simply using a smaller batch size.

Thus, we set $k = B$, where B is the full batch size, at the start of training and gradually reduce it over the course of training. In practice, we decay k by a constant factor, γ, every epoch of training to a minimum of $k = \nu$. We use the minimum value ν so that training doesn't progress to the point of only having one element in the mini-batch. Refer to Section 4 and 5 for more details on the values of γ and ν we used in practice.

3.3. Top-k Training of GANs in PyTorch

A sample PyTorch-like code snippet is available below for any general generator-critic GAN model to show the ease of implementation (Paszke et al., 2017a). In the code snippet, `generator_loss` represents any standard generator loss function. Compared to standard GAN training, using our top-k formulation amounts simply to the addition of line 8 of the example code. This highlights its ease of implementation and generality.

```
1  # Generate samples from the generator
2  fake_samples = Generator(prior_samples)
3
4  # Get critic predictions
5  predictions = Critic(fake_samples)
6
7  #Get topk predictions
8  topk_predictions = torch.topk(predictions, k)
9
10 # Compute loss for generator on top-k predictions
11 loss = generator_loss(topk_predictions)
```

4. Mixture of Gaussians

In this section we investigate the performance of top-k GAN training on a toy task in order to better understand its behaviour. Following Azadi et al. (2018) our toy task has a target distribution that is a mixture of Gaussians with a varying number of modes. We will first demonstrate and discuss how top-k training of GANs can reduce mode dropping (i.e. learning to generate only a subset of the individual mixture components) and boost sample quality in this setting. We then move on to discuss an interesting phenomenon: when gradient updates are performed on the bottom-k instead of the top-k batch elements, samples actually tend to be pushed away from their nearest mode. This phenomenon suggests a mechanism by which top-k training improves GAN performance: it doesn't use these "unhelpful" gradients in its stochastic mini-batch estimate of the full gradient.

4.1. Experimental Setup

We follow the same experimental setup as in Azadi et al. (2018) and Sinha et al. (2019). We 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. The generator and critic are 4-layer MLPs with 256 hidden units in each layer, which are trained using the 'non-saturating' loss from Goodfellow et al. (2014). We train each network with a constant batch-size of 256 for all experiments. All networks are trained with Adam optimizer with a learning rate of 10^{-4} (Kingma & Ba, 2014).

For all experiments we measure (as in Azadi et al. (2018)): i) High quality samples: percent of samples that lie at most 4 standard deviations away from the nearest mode and ii) Modes recovered: percent of modes which have at least one high quality sample. The more modes the generator is able to recover, the less we say it suffers from mode-dropping. For this evaluation, we randomly sample $10{,}000$ samples from the generator. We train the networks for $100{,}000$ iterations and decay k every $2{,}000$ iterations.

For top-k training, we initialize k to be the full mini-batch size. We use a decay factor, γ, of 0.99 to decay k until it reaches its minimum value, ν, of 75% of the initial mini-batch size, or 192. Formally, we do:

$$k \leftarrow \max(\gamma k, \nu)$$

Number of Modes	% High Quality Samples (GAN)	% High Quality Samples (Top-k GAN)	% Modes Recovered (GAN)	% Modes Recovered (Top-k GAN)
25	85.6	**95.5**	**100**	**100**
64	73.8	**81.8**	96.2	**100**
100	40.3	**54.7**	94.6	**100**

Table 1. Top-k Training of GANs vs. a standard baseline GAN on a Mixture of Gaussians. 'High Quality Samples' measures the fraction of samples that lie at most 4 standard deviations away from the nearest mode. 'Modes Recovered' measures the fraction of modes which have at least one high quality sample. For 25, 64, and 100 modes, top-k training substantially increases the number of High Quality Samples and (somewhat less substantially) reduces mode-dropping.

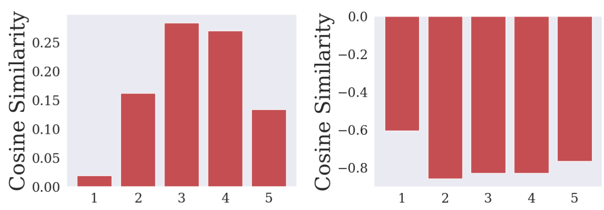

Figure 2. Cosine similarity between the direction moved by a generator sample after an update to the direction to the nearest mode for top-k (left) and bottom-k (right) samples. Each bin in the histogram represents samples which are within a given standard deviation away from the nearest mode.

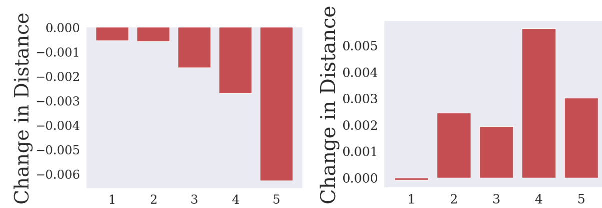

Figure 3. Change in distance to the nearest mode for generator samples after an update for top-k (left) and bottom-k (right). Each bin in the histogram represents samples which are within a given standard deviation away from the nearest mode.

4.2. Quantitative Results

The quantitative results for all Mixture-of-Gaussians experiments are summarized in Table 1. We see that as we increase the number of modes in the target distribution, top-k training is able to improve both the fraction of modes recovered and the fraction of high quality samples: As the number of modes is increased from 25 to 100, the number of high-quality samples decreases dramatically for normal GANs; top-k training performs significantly better. The fact that the number of modes recovered by performing top-k training is larger than the number recovered without shows that top-k training may help mitigate mode-dropping.

As GAN training progresses, the critic implicitly learns to classify whether or not a sample is drawn from the true distribution. Thus, generated samples that are in-between the modes of the target distribution tend to yield lower outputs from the critic (Azadi et al., 2018). By discarding these samples, we focus updates to the generator parameters on the best-scoring samples in the mini-batch, which results in better GAN training results. But why does this happen? In the next section, we will show that gradient updates computed on samples which are 'in-between modes' (samples that top-k sampling will discard) often cause samples to move in the *wrong* direction (i.e. away from the nearest mode) after each gradient update.

4.3. Why Does Throwing out Bad Samples Help?

In this section, we examine what happens when the GAN generator is updated on either the best-scoring elements or *worst*-scoring elements in a mini-batch. This sheds some light on a possible reason that top-k training is helpful: gradient updates computed on the worst-scoring samples tend to move samples away from the nearest mode.

For this experiment, we train a normal GAN for 50,000 iterations (half the number of iterations as in the experiments from Table 1) on a mixture of 25 Gaussians. Besides halving the number of iterations, we keep the settings otherwise the same as in Table 1. We then draw 1,000 samples from the generator's prior distribution $z \sim p(z)$. We use this batch z of 1,000 samples to generate samples from the generator. For each of those samples, we compute the direction to the nearest mode, which we refer to as the *oracle direction*[1].

Then, with these oracle directions as a reference point, we compare top-k and 'bottom-k' updating, which respectively update the generator using only the top-7,500 or bottom-2,500 critic predictions. After performing a gradient descent step, we re-compute the generator samples using the same z that were used to produce the oracle directions. We then measure the movement of the samples after the update steps. By isolating the effect of one gradient step, we can understand what happens when the generator is updated using the

[1] Note that, if some modes are very over-represented, the oracle direction won't be quite right, but in practice this is not a big issue.

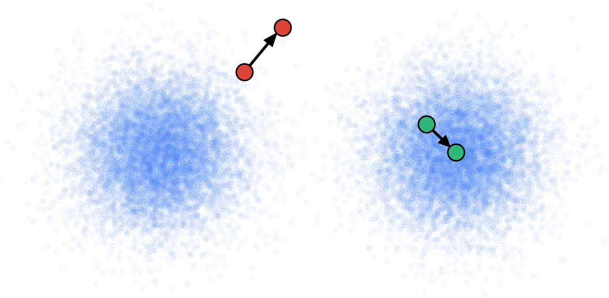

Figure 4. Diagram of what appears to happen in our toy Mixture of Gaussians experiment. On the left, we show the result of bottom-k updating (that is, only updating the generator parameters using the worst scoring elements of the mini-batch). The blue points represent samples from the target distribution, and the red points represent a sample before and after the bottom-k update: note that the sample moves *away* from the nearest mode. On the right, we show the result of top-k updating. The green points show a sample before and after the top-k update: note that the sample moves toward the nearest mode. One subtlety that bears mentioning - if samples are not evenly distributed about the modes, it's not strictly true that a sample should be pushed toward its nearest mode. However, in our experimental settings, samples are evenly distributed enough that it is a close approximation of the truth.

'bad' samples compared to what happens when it is updated using the 'good' samples. This comparison is unbiased because we use the same generator and critic and the same batch of z for both the top-k and the bottom-k update.

In order to understand why updating on the worst-scoring samples is harmful, we evaluate the cosine similarity between the oracle direction and the displacement computed above, for each sample. By evaluating the cosine similarity, we are loosely measuring 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 being 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 points are being pushed in the opposite direction of the nearest mode. The results of this experiment are summarized in Fig. 2: We bin points by how many standard deviations away they are from the nearest mode and compute a histogram of mean cosine distance for each bin. The '5' bar in each plot are all the points that greater than 4 standard deviations away (not high-quality samples).

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 are we consider to be high-quality samples (those within 4 standard deviations from the closest mode). This suggests that points are being actively pushed away from the mode that they are already close to. For samples which are very close to the nearest mode, the cosine similarity is less important since these samples are already "good". That is what we see when we update the generator using the proposed top-k method. The points that are more than 4 standard deviations away move in the correct direction, even when the generator was not directly updated on those exact points because of the masking operation from top-k. Figure 4 further visualizes this behavior.

We also compute the change in distance to the nearest mode after the gradient update is done. We want the distance to the nearest mode to decrease after the gradient update which means that the generated distribution is getting closer to the target distribution. We notice similar effects to cosine similarity, where when updating only on the bottom-k samples, we see that the distance to the nearest mode increases after the gradient update, while updating only on the top-k samples, the distance tends to decrease. The top-k plot shows that the further the point is from the nearest mode, the more it moves closer to it, and the points that are already very close to the mode remain relatively unaffected by the gradient update. 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; our method reduces the negative effects by simply discarding the bottom-k samples, which is a computationally efficient, effective and easy-to-implement solution.

5. Experiments on Image Datasets

In order to investigate whether top-k training scales beyond toy tasks, we apply it to several common GAN benchmarks. We first investigate its utility as a general tool for GAN-based architectures by testing it on anomaly detection (Chandola et al., 2009), finding that it improves results substantially. We then conduct experiments on the CIFAR-10 dataset (Krizhevsky et al., 2009) using six different popular GAN variants and on the ImageNet dataset (Russakovsky et al., 2015) using the SAGAN (Zhang et al., 2018) architecture. We test on a variety of GAN variants to ensure that our technique is generally applicable and the quantitative results indicate that it is: top-k training improved performance in all of the contexts where we tested it.

5.1. Anomaly Detection

We conduct experiments on the anomaly detection task and model from Kumar et al. (2019) (as also used in Sinha et al. (2019)). We augment Maximum Entropy Generators (MEG) with top-k sampling on the generator. MEG performs anomaly detection by learning the manifold of the true distribution; by learning a better generator function,

Held-out Digit	Bi-GAN	MEG	Top-k+MEG
1	0.287	0.281	**0.320**
4	0.443	0.401	**0.478**
5	0.514	0.402	**0.561**
7	0.347	0.29	**0.358**
9	0.307	0.342	**0.367**

Table 2. Experiments with Anomaly Detection on MNIST dataset. The 'Held-out Digit' is the digit that was held out of the training set during training and treated as the 'anomaly' class. The numbers reported is the area under the precision-recall curve.

DC-GAN	Top-k + DC-GAN	WGAN+GC	Top-k + WGAN+GC	WGAN+GP	Top-k + WGAN+GP
38.09 $\pm$ 0.3	**35.62 $\pm$ 0.4**	37.33 $\pm$ 0.3	**34.41 $\pm$ 0.3**	31.80 $\pm$ 0.2	**29.83 $\pm$ 0.2**
MS-GAN	Top-k + MS-GAN	SN-GAN	Top-k + SN-GAN	SA-GAN	Top-k + SA-GAN
27.33 $\pm$ 0.2	**26.54 $\pm$ 0.3**	21.36 $\pm$ 0.2	**19.80 $\pm$ 0.2**	19.02 $\pm$ 0.2	**17.93 $\pm$ 0.2**

Table 3. Reporting the FID-50k metric on the CIFAR dataset for various GAN architectures. The GAN architectures considered are DC-GAN, WGAN with Gradient Clipping, WGAN with Gradient Penalty, Mode-Seeking GAN, Spectral-Normalization GAN and Self-Attention GAN.

MEG aims to be able to learn a better model for anomaly detection.

As in Kumar et al. (2019), we train a generative model on 9 out of the 10 digits on the MNIST dataset (LeCun et al., 1998), where the images of the held-out digit are meant to simulate the anomalous examples that the method is intended to find. Since the results from Kumar et al. (2019) using MEGs are comparable to those from Zenati et al. (2018) (which uses BiGANs (Donahue et al., 2016)) we report both MEG and BiGAN-based solutions as our baseline methods. The results, which are shown in 2, are reported in terms of area under the precision-recall curve, as in Kumar et al. (2019). Broadly speaking, they show that applying top-k training to the MEG-based method yields results better than both the MEG-based and BiGAN-based methods for all 5 of the held-out digits we tried. Though we only apply top-k training to the MEG method in this instance, we suspect it can be fruitfully applied to BiGAN-based methods as well. By applying top-k training to a task other than image synthesis, we aim to show that it is a generally useful technique, rather than a task-specific hack.

5.2. Experiments on CIFAR-10

Since the most common application of GANs is to image synthesis, we exhaustively evaluate our method on different GAN variants using the CIFAR-10 dataset (Krizhevsky et al., 2009). The CIFAR dataset is a natural image dataset consisting of 50,000 training samples and 10,000 test samples from 10 possible classes and is probably the most widely-studied GAN benchmark. For all of our experiments, we compute the FID (Heusel et al., 2017) of the generator using 50,000 training images and 50,000 generator samples. It is important to note that we use a PyTorch Inception (Paszke et al., 2017b) network to compute the FID, instead of the TensorFlow implementation (Paszke et al., 2017b). This means that the overall values will be lower, but it does not affect relative ranking of models, so it still enables unbiased comparisons. Since we use the same implementation to compute each FID value in this paper, the results are comparable.

We apply top-k training to all of the following GAN variants:

- DC-GAN (Radford et al., 2015): A simple, widely used architecture that uses convolutions and deconvolutions for the critic and the generator.
- WGAN with Gradient Clipping (Arjovsky et al., 2017): Attempts to use an approximate Wasserstein distance as the critic loss by clipping the weights of the critic to bound the gradient of the critic with respect to its inputs.
- WGAN with Gradient Penalty (Gulrajani et al., 2017): Improves on WGAN (Arjovsky et al., 2017) by adding an gradient norm penalty to the critic instead of clipping weights.
- Mode-Seeking GAN (Mao et al., 2019): Attempts to generate more diverse images by selecting more samples from under-represented modes of the target distribution.
- Spectral Normalization GAN (Miyato et al., 2018):

SAGAN	Top-k + SAGAN
19.98	**18.44**

Table 4. Reporting the FID metric for Self Attention GAN on the ImageNet Dataset

$\gamma = 0.999$	$\gamma = 0.99$	$\gamma = 0.9$	$\gamma = 0.75$	$\gamma = 0.5$	$\nu = 0.9$	$\nu = 0.75$	$\nu = 0.5$	D	G & D
18.68	**17.93**	18.14	21.60	25.30	18.47	18.08	**17.93**	27.44	27.57

Table 5. FID scores for SAGAN on CIFAR over a range of ablation studies. For each experiment, all other hyperparameters are as proposed. Note: ν is listed as a percent of full mini-batch size (128). The **bold** values represent the proposed values of the given hyperparameters. Experiment labeled "D" represents applying top-k on just the critic. Experiment labeled "G & D"represents applying top-k on both the generator and critic.

Replaces the gradient penalty with a (loose) bound on the spectral norm of the weight matrices of the critic.

- Self-Attention GAN (Zhang et al., 2018): Applies self-attention on both the generator and critic.

For all experiments, we use a mini-batch size of 128 and initialize the value of k to be the full mini-batch size. Unless otherwise noted, we set γ to 0.99, where we decay k after every epoch until it reaches the value of half the original batch size, $\nu = 64$. We considered values of γ in the range of $[0.75, 0.999]$ and values of ν in the range of $[32, 100]$. **For each model, all other hyper-parameters used were same as those used in the paper proposing that model.** By leaving the original hyperparameters fixed, we can demonstrate how top-k training is a "drop-in" improvement for each of these GANs.

The results of these experiments are summarized in Table 3. We see that using top-k sampling significantly helps the performance across all GAN variants. For the simpler GAN variants, such as DCGAN and WGAN with gradient clipping, we see that the performance is significantly better when using top-k. Even for the state-of-the-art GAN architectures, such as Self-Attention GAN and Spectral Normalization GAN, our method is able to outperform the baseline by a good margin. We speculate that we achieve larger improvements on less sophisticated GAN models for the simple reason that there is less room for improvement on the more sophisticated models (though top-k training yields substantial improvements in all cases).

The fact that using top-k sampling yields improvements in so many different contexts is evidence that it addresses a fundamental problem with GAN training, and suggests that it may not simply be a 'hack' that happens to mesh well with certain training procedures or GAN objectives.

5.3. Experiments On ImageNet

ImageNet (Russakovsky et al., 2015) is a large-scale image dataset consisting of over 1.2 million images from 1,000 different classes. Training a GAN to perform Conditional Image Synthesis on the ImageNet dataset is now a standard GAN benchmark to show how a given GAN scales. Since this benchmark is considered more difficult than training an image synthesizer on the CIFAR-10 dataset, we include these experiments as evidence that top-k training can scale up to more difficult problem settings.

We run our experiments with the Self Attention GAN (SAGAN) (Zhang et al., 2018), since it is relatively standard, has open-source code available, and is easy to modify. As in our CIFAR-10 experiments, we train the baseline model using the same hyper-parameters as suggested in the original paper. We set the top-k decay rate – γ — to 0.98 due to the large size of the dataset. We report the FID score on 50,000 generated samples from the generator. The results are summarized in Table 4.

Broadly speaking, the results show that top-k training can substantially improve FID scores, despite being an extremely simple intervention. The fact that we were able to achieve this results with minimal hyper-parameter modifications is a testament to the broad applicability of the top-k training technique.

6. Examining the Main Hyper-parameters

In this section we study the effects of the various hyper-parameters involved in performing top-k GAN training. In particular, we focus on the effect of the decay rate, γ; the minimum value of k, ν; and the effect of applying top-k updates to the critic as well as just to the generator.

For all of these experiments we train a Self Attention GAN (Zhang et al., 2018) to model the CIFAR-10 dataset (Krizhevsky et al., 2009). Except for the hyper-parameters under consideration, we keep all settings the same as in Section 5.2 unless otherwise noted. The results are presented in Table 5.

The first thing to notice is that using too small of a value of γ hurts performance by discarding too many samples early on in training. Secondly, 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.

6.1. Applying Top-k Updates to The Critic Hurts Performance

Perhaps most interesting of all is that applying top-k updates to the critic (instead of just to the generator, as we do in all other experiments) completely destabilizes training. Further study of this phenomenon is best deferred to future work, but we can briefly speculate that modifying the critic in this way is harmful because it causes the critic only to update for modes that the generator has learned early on, ignoring other parts of the target distribution and thus preventing the generator from learning those parts.

7. Related Work

7.1. Generative Adversarial Networks

Recent GAN reserach has focused on generating increasingly realistic images. Both Odena (2019) and Kurach et al. (2019) serve as good overviews on the state of current GAN research.

A wide vareity of techniques have been proposed to improve GAN training, including mimicking or using large batches (Sinha et al., 2019; Brock et al., 2018), different GAN architectures (Chavdarova & Fleuret, 2018; Zhang et al., 2018; Radford et al., 2015; Brock et al., 2018; Li et al., 2017), and different GAN training objectives (Mao et al., 2017; Arjovsky et al., 2017; Nowozin et al., 2016; Zhang et al., 2020; Bellemare et al., 2017; Metz et al., 2016). Alternatively, variance reduction has been explored as a way to stabilize the GAN training procedure: Gidel et al. (2018) proposes solve a variational-inequality problem instead of the solving the min-max two player GAN objective, and Chavdarova & Fleuret (2018); Chavdarova et al. (2019) propose using an extra-gradient method while training. Some recently proposed methods have tried to improve GANs from a computer graphics lens (Karras et al., 2019b;a; 2017). Other work focuses on conditional image synthesis on large-scale datasets such as ImageNet Russakovsky et al. (2015), (Odena et al., 2017; Miyato et al., 2018; Brock et al., 2018; Daras et al., 2019; Wu et al., 2019; Zhang et al., 2020). Some work even focuses on totally different ways to evaluate generative models (and GANs in particular) (Olsson et al., 2018; Gulrajani et al., 2020).

7.2. Effectively Using critic Outputs

Our work is more closely related to the line of research which aims to use the critic output to further augment the GAN training procedure. The goal is to distill more information from the critic than is possible using only standard GAN optimization techniques. Azadi et al. (2018) proposed a post-training procedure to use rejection sampling on the critic outputs for the generated samples. Their method shows that by rejecting samples from a trained GAN in proportion to the critic's outputs on those samples, the distribution modeled by the generator can be pushed much closer to the target distribution.

Wu et al. (2019) show that a similar trick can work *during training*, by only updating the generator using draws from the prior that have themselves been modified in response to the critic output using a gradient correction. This technique is expensive, however, and relatively complicated to implement as it requires two forward and backward passes for each GAN gradient update.

7.3. Top-k operation in Machine Learning

The top-k operation has been extensively explored for relational databases (Soliman et al., 2007; Ilyas et al., 2004). However, recent work has showed how the top-k operation can also be useful in supervised learning while performing SGD to learn robust classifiers (Berrada et al., 2018; Shah et al., 2020), for encouraging specialization between different modules in a modular architecture (Goyal et al., 2019), for training sparsely-gated neural networks (Shazeer et al., 2017) and for learning long term dependencies in recurrent neural networks (Ke et al., 2018).

8. Conclusion

We have described a technique that is very simple – it requires changing only one line of code – that yields non-trivial improvements across a wide variety of GAN architectures. In fact, it yielded substantial improvements in every context in which we evaluated it. A more sophisticated technique could probably yield slightly better results after substantial tweaking, but there are serious barriers to using such a technique in practice – simplicity tends to win out. We hope that this technique will become standard.

We have also discovered and studied an interesting phenomenon in the Mixture-of-Gaussians setting: generators updated using top-k updating push samples toward their nearest mode, while generators updated using bottom-k updating tend to push samples *away from their nearest mode*. This partially explains why top-k sampling is successful (it removes from the mini-batch incorrect contributions to the estimate of the gradient), but it is also interesting in its own right. We hope that further study of this phenomenon can spur advances in our understanding of the GAN training procedure: perhaps it can connected with other interesting experimental observations about GANs, or used to explain performance improvements from other heuristically motivated techniques.

References

Arjovsky, M., Chintala, S., and Bottou, L. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

Azadi, S., Olsson, C., Darrell, T., Goodfellow, I., and Odena, A. Discriminator rejection sampling. *arXiv preprint arXiv:1810.06758*, 2018.

Bellemare, M. G., Danihelka, I., Dabney, W., Mohamed, S., Lakshminarayanan, B., Hoyer, S., and Munos, R. The cramer distance as a solution to biased wasserstein gradients. *arXiv preprint arXiv:1705.10743*, 2017.

Berrada, L., Zisserman, A., and Kumar, M. P. Smooth loss functions for deep top-k classification. *arXiv preprint arXiv:1802.07595*, 2018.

Brock, A., Donahue, J., and Simonyan, K. Large scale gan training for high fidelity natural image synthesis. *arXiv preprint arXiv:1809.11096*, 2018.

Chandola, V., Banerjee, A., and Kumar, V. Anomaly detection: A survey. *ACM computing surveys (CSUR)*, 41(3): 15, 2009.

Chavdarova, T. and Fleuret, F. Sgan: An alternative training of generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 9407–9415, 2018.

Chavdarova, T., Gidel, G., Fleuret, F., and Lacoste-Julien, S. Reducing noise in gan training with variance reduced extragradient. *arXiv preprint arXiv:1904.08598*, 2019.

Daras, G., Odena, A., Zhang, H., and Dimakis, A. G. Your local gan: Designing two dimensional local attention mechanisms for generative models. *arXiv preprint arXiv:1911.12287*, 2019.

Donahue, C., McAuley, J., and Puckette, M. Synthesizing audio with generative adversarial networks. *arXiv preprint arXiv:1802.04208*, 2018a.

Donahue, C., McAuley, J., and Puckette, M. Adversarial audio synthesis. *arXiv preprint arXiv:1802.04208*, 2018b.

Donahue, J., Krähenbühl, P., and Darrell, T. Adversarial feature learning. *arXiv preprint arXiv:1605.09782*, 2016.

Dowson, D. and Landau, B. The fréchet distance between multivariate normal distributions. *Journal of multivariate analysis*, 12(3):450–455, 1982.

Gidel, G., Berard, H., Vignoud, G., Vincent, P., and Lacoste-Julien, S. A variational inequality perspective on generative adversarial networks. *arXiv preprint arXiv:1802.10551*, 2018.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Goyal, A., Lamb, A., Hoffmann, J., Sodhani, S., Levine, S., Bengio, Y., and Schölkopf, B. Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*, 2019.

Gulrajani, I., Ahmed, F., Arjovsky, M., Dumoulin, V., and Courville, A. C. Improved training of wasserstein gans. In *Advances in neural information processing systems*, pp. 5767–5777, 2017.

Gulrajani, I., Raffel, C., and Metz, L. Towards gan benchmarks which require generalization. *arXiv preprint arXiv:2001.03653*, 2020.

Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., and Hochreiter, S. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pp. 6626–6637, 2017.

Ilyas, I. F., Aref, W. G., and Elmagarmid, A. K. Supporting top-k join queries in relational databases. *The VLDB Journal—The International Journal on Very Large Data Bases*, 13(3):207–221, 2004.

Karras, T., Aila, T., Laine, S., and Lehtinen, J. Progressive growing of gans for improved quality, stability, and variation. *arXiv preprint arXiv:1710.10196*, 2017.

Karras, T., Laine, S., and Aila, T. A style-based generator architecture for generative adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4401–4410, 2019a.

Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., and Aila, T. Analyzing and improving the image quality of stylegan. *arXiv preprint arXiv:1912.04958*, 2019b.

Ke, N. R., GOYAL, A. G. A. P., Bilaniuk, O., Binas, J., Mozer, M. C., Pal, C., and Bengio, Y. Sparse attentive backtracking: Temporal credit assignment through reminding. In *Advances in neural information processing systems*, pp. 7640–7651, 2018.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

Kumar, R., Goyal, A., Courville, A., and Bengio, Y. Maximum entropy generators for energy-based models. *arXiv preprint arXiv:1901.08508*, 2019.

Kurach, K., Lucic, M., Zhai, X., Michalski, M., and Gelly, S. A large-scale study on regularization and normalization in gans. In *ICML*, 2019.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z., et al. Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4681–4690, 2017.

Li, C.-L., Chang, W.-C., Cheng, Y., Yang, Y., and Póczos, B. Mmd gan: Towards deeper understanding of moment matching network. In *Advances in Neural Information Processing Systems*, pp. 2203–2213, 2017.

Mao, Q., Lee, H.-Y., Tseng, H.-Y., Ma, S., and Yang, M.-H. Mode seeking generative adversarial networks for diverse image synthesis, 2019.

Mao, X., Li, Q., Xie, H., Lau, R. Y., Wang, Z., and Paul Smolley, S. Least squares generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2794–2802, 2017.

Metz, L., Poole, B., Pfau, D., and Sohl-Dickstein, J. Unrolled generative adversarial networks. *arXiv preprint arXiv:1611.02163*, 2016.

Miyato, T., Kataoka, T., Koyama, M., and Yoshida, Y. Spectral normalization for generative adversarial networks. *arXiv preprint arXiv:1802.05957*, 2018.

Nowozin, S., Cseke, B., and Tomioka, R. f-gan: Training generative neural samplers using variational divergence minimization. In *Advances in neural information processing systems*, pp. 271–279, 2016.

Odena, A. Open questions about generative adversarial networks. *Distill*, 4(4):e18, 2019.

Odena, A., Olah, C., and Shlens, J. Conditional image synthesis with auxiliary classifier gans. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 2642–2651. JMLR. org, 2017.

Olsson, C., Bhupatiraju, S., Brown, T., Odena, A., and Goodfellow, I. Skill rating for generative models. 2018.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. 2017a.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in pytorch. 2017b.

Radford, A., Metz, L., and Chintala, S. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3): 211–252, 2015.

Shah, V., Wu, X., and Sanghavi, S. Choosing the sample with lowest loss makes sgd robust. *arXiv preprint arXiv:2001.03316*, 2020.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Sinha, S., Zhang, H., Goyal, A., Bengio, Y., Larochelle, H., and Odena, A. Small-gan: Speeding up gan training using core-sets. *arXiv preprint arXiv:1910.13540*, 2019.

Soliman, M. A., Ilyas, I. F., and Chang, K. C.-C. Top-k query processing in uncertain databases. In *2007 IEEE 23rd International Conference on Data Engineering*, pp. 896–905. IEEE, 2007.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

Wu, Y., Donahue, J., Balduzzi, D., Simonyan, K., and Lillicrap, T. Logan: Latent optimisation for generative adversarial networks. *arXiv preprint arXiv:1912.00953*, 2019.

Xian, Y., Lorenz, T., Schiele, B., and Akata, Z. Feature generating networks for zero-shot learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5542–5551, 2018.

Zenati, H., Foo, C. S., Lecouat, B., Manek, G., and Chandrasekhar, V. R. Efficient gan-based anomaly detection. *arXiv preprint arXiv:1802.06222*, 2018.

Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., and Metaxas, D. N. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 5907–5915, 2017.

Zhang, H., Goodfellow, I., Metaxas, D., and Odena, A. Self-attention generative adversarial networks. *arXiv preprint arXiv:1805.08318*, 2018.

Zhang, H., Zhang, Z., Odena, A., and Lee, H. Consistency regularization for generative adversarial networks. In *ICLR*, 2020.

Zhu, J.-Y., Park, T., Isola, P., and Efros, A. A. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pp. 2223–2232, 2017.