Boosting Generative Adversarial Networks

Haiyan Wang
Duke University
haiyan.wang916@duke.edu

1 Introduction

1.1 Generative Adversarial Networks

Generative adversarial networks (GANs) were formulated by Goodfellow et al. (2014) as a novel framework for creating generative artificial intelligence models. GANs imagine the training process of generative models as a game that is, as the name implies, built upon an adversarial interaction between two "players" – a generative model that produces artificial samples and a discriminative classifier that seeks to distinguish these artificial samples from true ones drawn from training data. These models "compete" against one another, with the game being defined in such a way that improvements in one player induce improvements in the other, at least in theory. At a high level, it is obvious how such a game leads its players to converge to strong approximations: as the generator improves, it tricks the disciminator more frequently, increasing its loss and forcing it to improve its classification capabilities. On the next iteration however, the generator's existing strategy can no longer beat the improved discriminator, so generator loss increases and forces it to improve the quality of its generated samples. The adversarial players compel each other to make incremental improvements, converging gradually toward optimal model generative and discriminative models, respectively, that induce a Nash equilibrium in the game. This approach strongly suggests an underlying minimax algorithm, and indeed, GANs represent a two-player minimax game in which the generator is the minimizer and the discriminator is the maximizer. In particular, the generator is trained to maximize the probability that the discriminator misclassifies artificial samples as real data.

To develop a mathematical formulation for the objective function underlying this game, consider a GAN in which the generator and discriminator are both multilayer perceptrons. The generator learns a function G that maps a latent space p_Z to a distribution of artificial samples p_G. More specifically, given some noise vector $z \sim p_Z$, the generator returns an artificial sample $G(z, \theta_G) \sim p_G$, where θ_G are the learned parameters of the generator. Similarly, the discriminator learns a function D that, given some unlabeled sample x (artificial or true), returns a scalar $D(x, \theta_D) \in [0, 1]$, where θ_D are the learned parameters of the discriminator. The output of D is the probability that x is a true sample, or $x \sim p_X$ where p_X is the underlying distribution of the training data. D is trained to maximize the probability of correctly classifying artificial samples from p_G and true samples from p_X, while G is trained to minimize the probability of D correctly classifying artificial samples. Expressing this objective in terms of log probabilities, we arrive at the value function $V(G,D)$ that describes the minimax game between the generator and discriminator:

$$\min_G \max_D V(G,D) = \mathbb{E}_{x\sim p_X}[\log D(x)] + \mathbb{E}_{z\sim p_Z}[\log(1 - D(G(Z)))] \tag{1}$$

Submitted to:
MATH465: High Dimensional Data Analysis (Fall 2023), Duke University

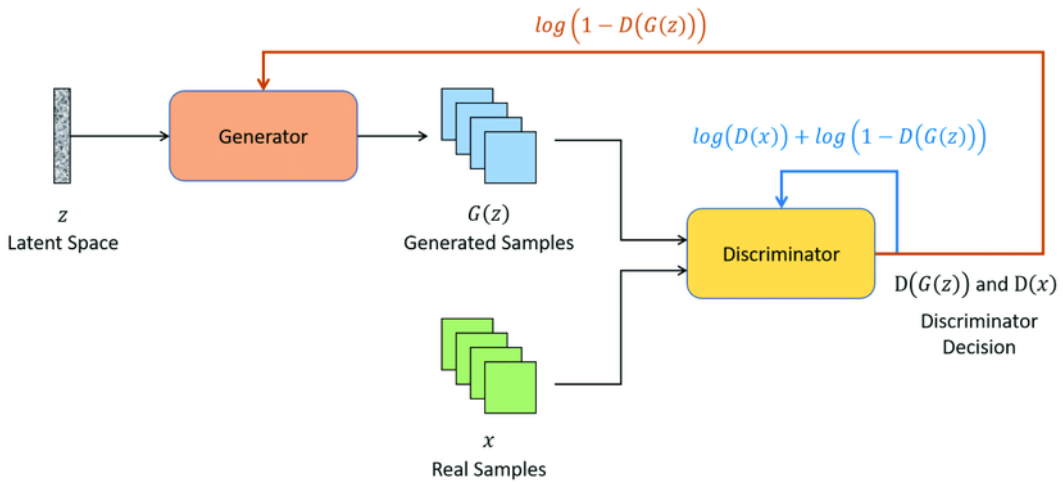

Figure 1: Basic GAN architecture (Vint et al. (2021))

Goodfellow et al. (2014) demonstrate several theoretical results from the above criterion in the context of models with infinite capacity.

Theorem 1. *For any fixed generator G, the optimal discriminator D is given by*

$$D_G^*(x) = \frac{f_X(x)}{f_X(x) + f_G(x)} \tag{2}$$

where $f_X(x)$ and $f_G(x)$ are the probability distribution functions of p_X and p_G, respectively.

Proof. Let $f_Z(z)$ be the probability distribution function of p_Z and recall the training criterion for the discriminator D

$$\begin{aligned}
D_G^*(x) &= \max_D V(G,D) \\
&= \max_D \mathbb{E}_{x\sim p_X}[\log D(x)] + \mathbb{E}_{z\sim p_Z}[\log(1-D(G(Z)))] \\
&= \max_D \left[\int_{p_X} f_X(x)\log[D(x)]dx + \int_{p_Z} f_Z(z)\log[1-D(G(z))]dz \right] \\
&= \max_D \left[\int_{p_X} f_X(x)\log[D(x)] + f_G(x)\log[1-D(x)]dx \right]
\end{aligned}$$

For any $a,b \in \mathbb{R}$ with $a,b \neq 0$, the function $l(\lambda) = a\log\lambda + b\log(1-\lambda)$ has a maximum on $\lambda \in [0,1]$ at $\lambda = \frac{a}{a+b}$. We need only consider this interval because $D(x) \in [0,1]$. Then,

$$D_G^*(x) = \frac{f_X(x)}{f_X(x) + f_G(x)}$$

□

Rewriting the value function of the minimax game, the cost of the generator for the optimal discriminator $D_G^*(x)$ is

$$\begin{aligned} C(G) &= \max_D V(G,D) = V(G,D_G^*(x)) \\ &= \mathbb{E}_{x\sim p_X}[\log D_G^*(x)] + \mathbb{E}_{z\sim p_Z}[\log(1-D_G^*(G(z)))] \\ &= \mathbb{E}_{x\sim p_X}\left[\log \frac{f_X(x)}{f_X(x)+f_G(x)}\right] + \mathbb{E}_{x\sim p_G}\left[\log \frac{f_G(x)}{f_X(x)+f_G(x)}\right] \\ &= KL(p_X \| p_X + p_G) = KL(p_G \| p_X + p_G) \end{aligned}$$

where KL denotes Kullback-Leibler divergence.

We want to find the optimal generator $G*$ where

$$G* = \min_G C(G) = \min_G \left[\mathbb{E}_{x\sim p_X}\left[\log \frac{f_X(x)}{f_X(x)+f_G(x)}\right] + \mathbb{E}_{x\sim p_G}\left[\log \frac{f_G(x)}{f_X(x)+f_G(x)}\right]\right] \tag{3}$$

Theorem 2. *$C(G)$ reaches its global minimum if and only if $p_G = p_X$, at which point $C(G) = -\log 4$*

Proof. For $p_G = p_X$, $f_G(x) = f_X(x)$ and $D_G^*(x) = \frac{1}{2}$. Then $C(G) = \log\frac{1}{2} + \log\frac{1}{2} = -\log 4$.

Note that $-\log 4 = \mathbb{E}_{x\sim p_X}[-\log 2] + \mathbb{E}_{x\sim p_G}[-\log 2]$. We can then rewrite $C(G)$ as

$$C(G) = -\log 4 + KL\left(p_X \middle\| \frac{p_X+p_G}{2}\right) + KL\left(p_G \middle\| \frac{p_X+p_G}{2}\right) \tag{4}$$

where we've simply added $-\log 4$ and subtracted $\mathbb{E}_{x\sim p_X}[-\log 2] + \mathbb{E}_{x\sim p_G}[-\log 2]$, factoring the latter term into the KL divergences through linearity of expectation. We then recognize the form of Jensen-Shannon divergence, finally writing $C(G)$ as

$$C(G) = -\log 4 + 2\, JSD(p_X \| p_G) \tag{5}$$

For any two distributions p and q, $JSD(p\|q) \geq 0$ with $JSD(p\|q) = 0$ if and only if $p = q$. Therefore, the cost of the optimal generator is $C^* = -\log 4$ which is only achieved when $p_G = p_X$ and the generative model perfectly recreates the underlying distribution. □

Goodfellow et al. proves that under certain assumptions, nested gradient descent on the generator-discriminator pair produces convergence of p_G to p_X. That is, for each iteration of the generator, the discriminator is optimized to that generator, and the generator is updated based on the performance of the discriminator. In particular, if the generator and discriminator have sufficient capacity, the discriminator is trained to its optimal configuration D_G^* for each generator version G, and p_G is updated to improve $min_G\, V(G, D_G^*)$, then nested gradient descent leads to the convergence of p_G to p_X. In practice however, several additional restrictions are imposed on training the GAN. First, in early training iterations, the discriminator can often distinguish between true and artificial samples with high confidence, so modifications to the GAN objective may become necessary to avoid vanishing gradients. Second, while the game is still implemented through gradient descent with nested iterative optimization of the generator and discriminator, the discriminator is only optimized some fixed k number of times for each iteration

of generator optimization (where k is a hyperparameter). Fully optimizing the discriminator for every iteration of the generator, as in the theoretical case, is both computationally infeasible and would likely result in overfitting on finite datasets.

Algorithm 1 Minibatch stochastic gradient descent on GANs

for number of training iterations **do**

for k steps **do**

- Sample a minibatch of m noise vectors $\{z_1, ..., z_m\}$ from p_z
- Sample a minibatch of m real samples $\{x_1, ..., x_m\}$ from p_x
- Update the discriminator:

$$\nabla_{\theta_D} \frac{1}{m} \sum_{i=1}^{m} [\log D(x_i) + \log(1 - D(G(z_i)))] \tag{6}$$

end for

- Sample a minibatch of m noise vectors $\{z_1, ..., z_m\}$ from p_z
- Update the generator:

$$\nabla_{\theta_G} \frac{1}{m} \sum_{i=1}^{m} \log(1 - D(G(z_i))) \tag{7}$$

end for

1.2 Mode Collapse

Though they are powerful in theory and have achieved numerous cutting edge results in practice, GANs are notoriously difficult to train and fail in many cases to converge to Nash equilibrium if such a state even exists (Farnia, Ozdaglar (2020)). A particularly troublesome instance of this problem is mode collapse, in which a GAN will fail to produce samples in all but very specific regions of space despite an expansive training set. This phenomenon is termed the missing modes problem, and naively, it occurs in games where the objective of the discriminator reaches a local minimum that further optimization cannot escape. More specifically, we say intuitively that the generator is tasked with producing "believable" outputs that trick the discriminator. If it finds such an output, it has no incentive to continue exploring other outputs independently and will continue producing this output. Conversely however, if the discriminator is tricked repeatedly by the same output, it should learn the optimal strategy of consistently rejecting that output. This forces the generator to explore new outputs from training data, and over time, it learns how to reproduce the entire space underlying that data. However, if the discriminator fails to find this strategy and instead becomes trapped at some local minimum in its objective function, the generator will never be induced to change. It instead overfits to a single iteration of the disciminator, and its resulting outputs are limited to a subset of training classes.

A number of solutions have been proposed to address mode collapse during GAN training. In this paper, we review the application of methods that strongly resemble boosting in classification tasks to the training of generative models, with the hopes of mitigating mode collapse.

1.3 Boosting

Boosting has its roots in a question posed by Kearns (1988), termed the Hypothesis Boosting Problem. Informally, the question asks whether the existence of a "weak" classifier, or a model that performs slightly better than random guessing with high probability, implies the existence of a "strong" classifier, or a model whose accuracy can be arbitrarily high. Schapire (1990) gave the answer in a proof of the Strength of Weak Learnability, which states that any concept class is weakly learnable, i.e. there exists a hypothesis with slightly better than random performance, if and only if it is strongly learnable, i.e. there exists a hypothesis with arbitrarily low error. Here, we briefly explore Adaptive Boosting, or AdaBoost, a meta-algorithmic technique introduced by Schapire and Freund (1995) that leverages the theoretical implications of boosting to build arbitrarily strong classifiers from a collection of weak models.

Algorithm 2 AdaBoost

Input: Training sample X

Initialize the weights on the training samples as a uniform distribution $d_{1,i} = \frac{1}{n}$

for classifier t of T **do**

- Generate weak classifier $h_t = A(X, d_t)$ on samples weighted by d_t
- Calculate the weighted misclassification error of h_t

$$\varepsilon_t = \frac{1}{n} \sum_{i=1}^{n} d_{t,i} \mathbf{1}_{h_t(x_i) \neq y_i}$$

- Calculate the weight of h_t in the final strong classifier

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$$

- Define a normalization factor Z_t such that updated weights are a discrete probability distribution

$$Z_t = \sum_{i=1}^{n} e^{\alpha_t y_i h_t(x_i)}$$

- Update sample weights where $y_i h_t(x_i) = 1$ if the prediction is correct and -1 if not

$$d_{t+1,i} = \frac{d_{t,i}}{Z_t} e^{\alpha_t y_i h_t(x_i)}$$

end for

Aggregate the weak classifiers

$$H = \sum_{t=1}^{T} \alpha_t \cdot h_t$$

At first glance, many of steps and calculations in Algorithm 2, such as the selection of weights for each weak classifier in the final ensemble or the reweighting of samples based on a probability distribution, seem arbitrary. However, these steps are a result of the optimization technique underlying AdaBoost. Specifically, AdaBoost is proven to be optimal through coordinate descent, and its objective function is defined over exponential loss. After working through the coordinate descent procedure (which is excluded here), the above reweighting and ensemble scheme emerges.

AdaBoost illustrates many important characteristics of boosting. Broadly speaking, the intuition underlying boosted classifiers is the idea that building an ensemble model by iteratively and greedily training new classifiers specifically optimized towards classifying samples that previous iterations have failed at leads to a powerful overall model once all these classifiers are combined. Boosting accomplishes this reweighting the dataset such that frequently misclassified samples in past iterations are weighted more heavily in the training of future classifiers, and vice versa. Logically, a similar approach should be applicable to generators as well. If a GAN suffers from mode collapse, why not train another GAN that specifically aims to generate samples from regions of space that weren't represented in the first? In the next section, we explore existing methods that aim to apply this methodology.

2 AdaGAN

It turns out that the methodology suggested previously, though valid in a very general sense, is much easier said than done. Given some dataset, we could train a generator and realize that we've fallen victim to mode collapse with only a few classes from the original data being represented in the generated samples. Rather than bothering with any sort of reweighting scheme, we simply remove all examples of those represented classes from the training set and run the same training algorithm again. Barring practical complications like limited dataset size, we would end up with a collection of generators that, when aggregated as a mixture model, would produce generated samples that are fairly representative of training data. Unfortunately, this approach assumes the dataset is labeled, which is not always the case (and in any case, one of the biggest selling points of many generative models is the fact that they can learn to replicate some distribution without requiring any information about that distribution, including class labels). Instead, a workaround to requiring labels is proposed by Tolstikhin et al. (2017) in a procedure affectionately named AdaGAN for its similarities to AdaBoost, though much like AdaBoost, this approach is meta-algorithmic and could be applied to any generative model. In particular, they propose introducing another classifier into the training process that separates samples drawn from training data form samples created by a mixture of multiple generative models. This classifier is very similar to the discriminator used to train GANs, but it is distinguished by its independence from the individual generative models. That is to say, a new discriminator is created everytime a new GAN is trained and added to the mixture, but this classifier attempts to distinguish between training samples and samples generated by *all* created GANs thus far. We expect such a classifier to perform confidently for samples that are represented only in training data and not in generated data and less confidently on samples that are represented in both training and generated data. Using this "confidence," the training dataset can be reweighted such that underrepresented images for which the classifier is confident are more highly weighted in future training iterations. Tolstikhin et al. show that this approach – repeatedly training new generators on a weighted dataset, aggregating these generators as a mixture model, and reweighting the dataset according to the representative power of the mixture model – results in a mixture that approaches the true distribution.

Algorithm 3 AdaGAN

Input: Training sample X

Initialize the weights on the training samples as a uniform distribution $W_1 = \frac{1}{n}$

Train first generator: $G_1 = \text{GAN}(X, W_1)$

for GAN t of T **do**

- Choose a mixture weight β_t for the next component
- Update training weights W_t based performance of previous mixture G_{t-1}
- Train a new component generator G_t^c: $G_t^c = \text{GAN}(S_n, W_t)$
- Add the new generator to the mixture: $G_t = (1-\beta_t)G_{t-1} + \beta_t G_t^c$

end for

Return the resulting mixture model G_T

Algorithm 3 returns an additive mixture of the form

$$G_T = \sum_{i=1}^{T} \alpha_i G_i^c \tag{8}$$

where weights α_i form a discrete probability distribution and each G_i^c is a generator. A key distinction from boosting for classification lies in the fact that while boosted classifiers may return classifications based on the aggregate output of an ensemble model, we sample from this mixture G_T by first choosing a component from the mixture (under a multinomial distribution with parameters α_i) before sampling from that component by passing some noise vector z to it and noting the outputted generated sample $G_i^c(z)$.

Tolstikhin et al. show several results that we will explore here. First, it is possible to build an optimal mixture under any general f-divergence by incrementally adding new generators to the mixture that were trained on reweighted distributions. Second, under certain conditions that define "weak generators," mixtures of such generators converge to optimality.

2.1 Previous Work

Prior to the Tolstikhin et al., several authors have explored the application of boosting techniques to generative tasks. For example, Grover and Ermon (2017) provide a framework for multiplicative boosting (as opposed to additive boosting, demonstrated in AdaBoost). They establish a framework under which incorporating a new model into an ensemble is guaranteed to improve the ensemble's similarity to an unknown target distribution, and they find that such multiplicative boosting techniques for generative models are generally superior to additive boosting. However, Tolstikhin et al. points out that the tradeoff is that sampling from multiplicative ensembles is nontrivial, at least compared to the procedure outlined above. Barron and Li (1997) suggest an approach more similar to the one analyzed here, wherein mixtures are greedily constructed to minimize KL divergence. AdaGAN and the following summary can be regarded as an amalgamation of these previous results. We elect to trade the increased effectiveness of multiplicative ensembles for the ease of sampling provided by additive models, and in particular we

outline a greedy approach to boosting GANs that is theoretically effective and is more generally optimal (i.e. to f-divergences in general, not just KL divergence specifically).

2.2 Optimality of Additive Mixtures Under f-divergences

In the context of GANs and other generative models, it is impossible to directly compute the distribution of the training data p_X or the generated data p_G. However, it is possible to sample from both distributions (from training by simply drawing training samples and from testing by drawing $z \sim p_Z$ and passing it to the generator to produce $G(z) \sim p_G$). Therefore, we can estimate the similarity between the training and generated distributions $D(p_X \| p_G)$ by sampling from both distributions, and the goal of generator training is to find some function G over a class of functions $\mathscr{G}$ that approximately maximizes this similarity based on sampling results. This is the process that underlies GAN training.

From above, it is clear how f-divergences tie into this discussion. The similarity metric $D(p\|q)$ between distributions p and q with densities dp and dq, respectively, can be measured through f-divergences of the form

$$D_f(p\|q) = \int f\left(\frac{dp}{dq}(x)\right) dq(x) \tag{9}$$

with $D_f(p\|q) \geq 0$ and $D_f(p\|q) = 0$ if and only if $p = q$. This implies some limitations on f, namely that it is convex, defined on $(0,\infty)$, and satisfies $f(1) = 0$. f-divergences like Kullback-Leibler and Jensen-Shannon divergence have already appeared in this discussion, and indeed, they go hand-in-hand with generative models like GANs that seek to maximize the similarity between generated and underlying distributions. In fact, while the given objective function of GANs gives rise to optimums that involve Kullback-Leibler and Jensen-Shannon divergence, any f-divergence between distributions p and q can be applied to optimize objective functions of the form $\mathbb{E}_{x\sim p}[f_1(D(x))] + \mathbb{E}_{x\sim q}[f_2(D(G(x)))]$ through adversarial training. Thus, Tolstikhin et al. contextualizes their discussion of optimization with respect to f-divergences generally and base their work on the fact that we can estimate $D_f(p_X\|p_G)$ by taking two samples $x \sim p_X$ and $G(z) \sim p_G$ with $z \sim p_Z$ and training a classifier to label them with the distribution from which they were drawn. More specifically, given some sample drawn independently from p_X (which is unknown), we can build a simple model $G \in \mathscr{G}$ with distribution p_G that approximately minimizes $D_f(p_G\|p_X)$.

2.3 Greedy Construction

Building on Section 2.2, we want to approximate p_X with multiple distributions aggregated as a mixture where each distribution p_G^c was obtained by approximately minimizing $D_f(p_G^c\|p_X)$. We can obviously do this iteratively by training an initial model G_1^c with distribution p_1^c and creating a corresponding mixture G_1 where G_1^c has weight $\beta_1 = 1$. We then add subsequent models as follows. If the mixture G_t has t models in it already, we add model G_{t+1}^c with weight $\beta_t \in [0,1]$ by forming the new mixture G_{t+1}:

$$G_{t+1} = (1-\beta_t)G_t + \beta_t G_{t+1}^c \tag{10}$$

This process is greedy in that each component G_{t+1}^c added to the model should slightly improve the mixtures approxmation of p_X:

$$D_f((1-\beta_t)G_t + \beta_t G_{t+1}^c\|p_X) < D_f(G_t\|p_X) \tag{11}$$

While this approach works well in theory, we are constrained in practice by finite datasets. Specifically, when we construct the mixture and t grows larger, β decreases. Then when we sample from the mixture to train subsequent models, later components don't contribute enough samples to extract meaningful information about p_G from, and it becomes harder to opimize the generator's distribution.

3 Conclusions

The above explores the potential relationship between generative tasks and boosting methods and surveys existing results about how these boosting methods that were originally designed for classifiers can be applied to generative models as well, augmenting the output of such models. The greedy approach outlined is a theoretical start, but as stated, has significant pitfalls with regard to finite datasets in practice. In response to these shortcomings, Tolstikhin et al. proposes to instead optimize an upper bound that is more practical as an optimization objective for constructing the mixture. An exploration of this methodology and how we can bridge the gap between theoretically effective boosted classifiers to practically effective ones is suggested for future review.