# AdaGAN: Boosting Generative Models

Ilya Tolstikhin[1], Sylvain Gelly[2], Olivier Bousquet[2], Carl-Johann Simon-Gabriel[1], and Bernhard Schölkopf[1]

[1]Max Planck Institute for Intelligent Systems
[2]Google Brain

**Abstract**

Generative Adversarial Networks (GAN) [1] are an effective method for training generative models of complex data such as natural images. However, they are notoriously hard to train and can suffer from the problem of *missing modes* where the model is not able to produce examples in certain regions of the space. We propose an iterative procedure, called *AdaGAN*, where at every step we add a new component into a mixture model by running a GAN algorithm on a reweighted sample. This is inspired by *boosting* algorithms, where many potentially weak individual predictors are greedily aggregated to form a strong composite predictor. We prove that such an incremental procedure leads to convergence to the true distribution in a finite number of steps if each step is optimal, and convergence at an exponential rate otherwise. We also illustrate experimentally that this procedure addresses the problem of missing modes.

## 1 Introduction

Imagine we have a large corpus, containing unlabeled pictures of animals, and our task is to build a generative probabilistic model of the data. We run a recently proposed algorithm and end up with a model which produces impressive pictures of cats and dogs, but not a single giraffe. A natural way to fix this would be to manually remove all cats and dogs from the training set and run the algorithm on the updated corpus. The algorithm would then have no choice but to produce new animals and, by iterating this process until there's only giraffes left in the training set, we would arrive at a model generating giraffes (assuming sufficient sample size). At the end, we aggregate the models obtained by building a mixture model. Unfortunately, the described meta-algorithm requires manual work for removing certain pictures from the *unlabeled* training set at every iteration.

Let us turn this into an automatic approach, and rather than including or excluding a picture, put continuous weights on them. To this end, we train a binary classifier to separate "true" pictures of the original corpus from the set of "synthetic" pictures generated by the mixture of *all the models* trained so far. We would expect the classifier to make *confident* predictions for the true pictures of animals missed by the model (giraffes), because there are no synthetic pictures nearby to be confused with them. By a similar argument, the classifier should make less confident predictions for the true pictures containing animals already generated by one of the trained models (cats and dogs). For each picture in the corpus, we can thus use the classifier's confidence to compute a weight which we use for that picture in the next iteration, to be performed on the re-weighted dataset.

The present work provides a principled way to perform this re-weighting, with theoretical guarantees showing that the resulting mixture models indeed approach the true data distribution.[1]

Before discussing how to build the mixture, let us consider the question of building a single generative model. A recent trend in modelling high dimensional data such as natural images is to use neural networks [2, 1]. One popular approach are *Generative Adversarial Networks* (GAN) [1], where the generator

[1]Note that the term "mixture" should not be interpreted to imply that each component models only one mode: the models to be combined into a mixture can themselves cover multiple modes.

is trained adversarially against a classifier, which tries to differentiate the true from the generated data. While the original GAN algorithm often produces realistically looking data, several issues were reported in the literature, among which the *missing modes problem*, where the generator converges to only one or a few modes of the data distribution, thus not providing enough variability in the generated data. This seems to match the situation described earlier, which is why we will most often illustrate our algorithm with a GAN as the underlying base generator. We call it *AdaGAN*, for Adaptive GAN, but we could actually use any other generator: a Gaussian mixture model, a VAE [2], a WGAN [3], or even an unrolled [4] or mode-regularized GAN [5], which were both already specifically developed to tackle the missing mode problem. Thus, we do not aim at improving the original GAN or any other generative algorithm. We rather propose and analyse a meta-algorithm that can be used on top of any of them. This meta-algorithm is similar in spirit to AdaBoost [6] in the sense that each iteration corresponds to learning a "weak" generative model (e.g., GAN) with respect to a re-weighted data distribution. The weights change over time to focus on the "hard" examples, i.e. those that the mixture has not been able to properly generate so far.

## 1.1 Boosting via Additive Mixtures

Motivated by the problem of missing modes, in this work we propose to use multiple generative models combined into a mixture. These generative models are trained iteratively by adding, at each step, another model to the mixture that should hopefully cover the areas of the space not covered by the previous mixture components.[2] We show analytically that the optimal next mixture component can be obtained by reweighting the true data, and thus propose to use the reweighted data distribution as the target for the optimization of the next mixture components. This leads us naturally to a *meta-algorithm*, which is similar in spirit to AdaBoost in the sense that each iteration corresponds to learning a "weak" generative model (e.g., GAN) with respect to a reweighted data distribution. The latter adapts over time to focus on the "hard" examples, i.e. those that the mixture has not been able to properly generate thus far.

Before diving into the technical details we provide an informal intuitive discussion of our new meta-algorithm, which we call *AdaGAN* (a shorthand for Adaptive GAN, similar to AdaBoost). The pseudocode is presented in Algorithm 1.

On the first step we run the GAN algorithm (or some other generative model) in the usual way and initialize our generative model with the resulting generator $G_1$. On every $t$-th step we (a) pick the mixture weight $\beta_t$ for the next component, (b) update weights $W_t$ of examples from the training set in such a way to bias the next component towards "hard" ones, not covered by the current mixture of generators $G_{t-1}$, (c) run the GAN algorithm, this time importance sampling mini-batches according to the updated weights $W_t$, resulting in a new generator $G_t^c$, and finally (d) update our mixture of generators $G_t = (1-\beta_t)G_{t-1} + \beta_t G_t^c$ (notation expressing the mixture of $G_{t-1}$ and $G_t^c$ with probabilities $1-\beta_t$ and $\beta_t$). This procedure outputs $T$ generator functions $G_1^c, \ldots, G_T^c$ and $T$ corresponding non-negative weights $\alpha_1, \ldots, \alpha_T$, which sum to one. For sampling from the resulting model we first define a generator $G_i^c$, by sampling the index $i$ from a multinomial distribution with parameters $\alpha_1, \ldots, \alpha_T$, and then we return $G_i^c(Z)$, where $Z \sim P_Z$ is a standard latent noise variable used in the GAN literature.

The effect of the described procedure is illustrated in a toy example in Figure 1. On the left images, the red dots are the training (true data) points, the blue dots are points sampled from the model mixture of generators $G_t$. The background colour gives the density of the distribution corresponding to $G_t$, non zero around the generated points, (almost) zero everywhere else. On the right images, the color corresponds to the weights of training points, following the reweighting scheme proposed in this work. The top row corresponds to the first iteration of AdaGAN, and the bottom row to the second iteration. After the first iteration (the result of the vanilla GAN), we see that only the top left mode is covered, while the three other modes are not covered at all. The new weights (top right) show that the examples from covered mode are aggressively downweighted. After the second iteration (bottom left), the combined generator can then generate two modes.

[2]Note that the term "mixture" should not be interpreted to imply that each component models only one mode: the models to be combined into a mixture can themselves cover multiple modes already.

**Algorithm 1:** AdaGAN, a meta-algorithm to construct a "strong" mixture of $T$ individual GANs, trained sequentially. The mixture weight schedule ChooseMixtureWeight and the training set reweighting schedule UpdateTrainingWeights should be provided by the user. Section 3 gives a complete instance of this family.

**Input:** Training sample $S_N := \{X_1, \ldots, X_N\}$.

**Output:** Mixture generative model $G = G_T$.

Train vanilla GAN:

$W_1 = (1/N, \ldots, 1/N)$

$G_1 = \text{GAN}(S_N, W_t)$

**for** $t = 2, \ldots, T$ **do**

*#Choose a mixture weight for the next component*

$\beta_t = \text{ChooseMixtureWeight}(t)$

*#Update weights of training examples*

$W_t = \text{UpdateTrainingWeights}(G_{t-1}, S_N, \beta_t)$

*#Train t-th "weak" component generator* $G_t^c$

$G_t^c = \text{GAN}(S_N, W_t)$

*#Update the overall generative model*

*#Notation below means forming a mixture of* $G_{t-1}$ *and* $G_t^c$.

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

**end for**

Although motivated by GANs, we cast our results in the general framework of the minimization of an $f$-divergence (cf. [7]) with respect to an additive mixture of distributions. We also note that our approach may be combined with different "weak" generative models, including but not limited to GAN.

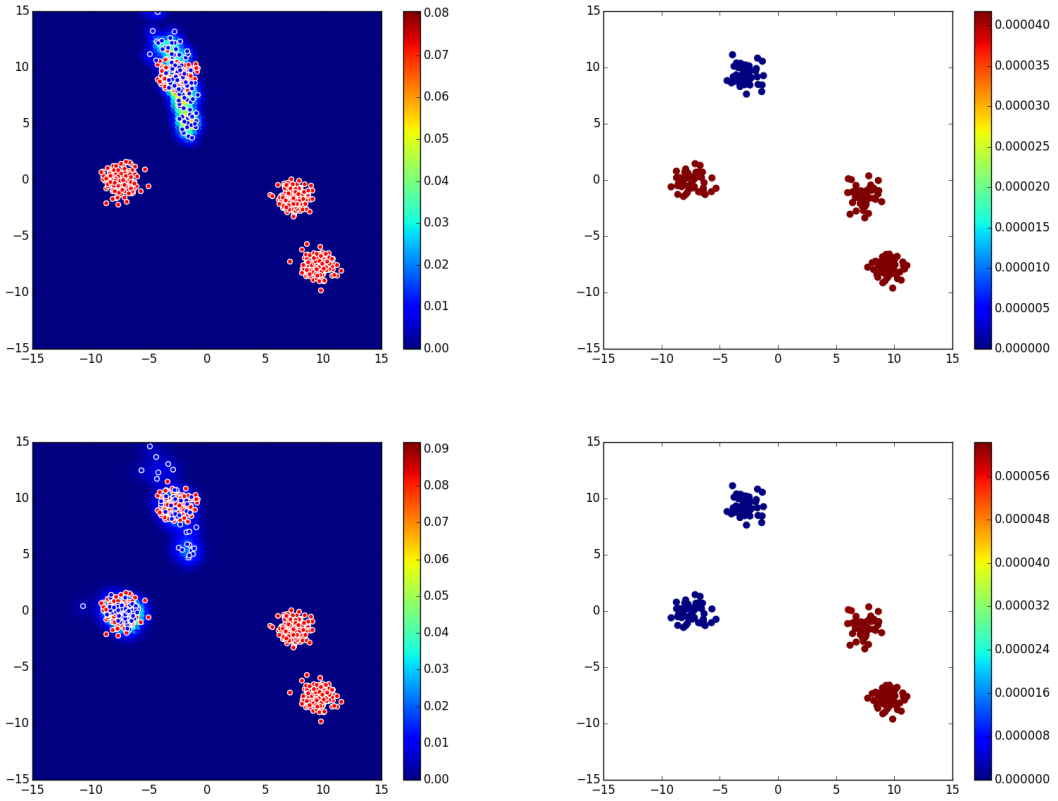

Figure 1: A toy illustration of the missing mode problem and the effect of sample reweighting, following the discussion in Section 1.1. On the left images, the red dots are the training (true data) points, the blue dots are points sampled from the model mixture of generators $G_t$. On the right images, the color corresponds to the weights of training points, following the reweighting scheme proposed in this work. The top row corresponds to the first iteration of AdaGAN, and the bottom row to the second iteration.

## 1.2 Related Work

Several authors [8, 9, 10] have proposed to use boosting techniques in the context of density estimation by incrementally adding components in the log domain. In particular, the work of Grover and Ermon [10], done in parallel to and independent of ours, is applying this idea to GANs. A major downside of these approaches is that the resulting mixture is a product of components and sampling from such a model is nontrivial (at least when applied to GANs where the model density is not expressed analytically) and requires to use techniques such as Annealed Importance Sampling [11] for the normalization.

Rosset and Segal [12] proposed to use an additive mixture model in the case where the log likelihood can be computed. They derived the update rule via computing the steepest descent direction when adding a component with infinitesimal weight. This leads to an update rule which is degenerate if the generative model can produce arbitrarily concentrated distributions (indeed the optimal component is just a Dirac distribution) which is thus not suitable for the GAN setting. Moreover, their results do not apply once the

weight $\beta$ becomes non-infinitesimal. In contrast, for any fixed weight of the new component our approach gives the overall optimal update (rather than just the best direction), and applies to any $f$-divergence. Remarkably, in both theories, improvements of the mixture are guaranteed only if the new "weak" learner is still good enough (see Conditions 14&15)

Similarly, Barron and Li [13] studied the construction of mixtures minimizing the Kullback divergence and proposed a greedy procedure for doing so. They also proved that under certain conditions, finite mixtures can approximate arbitrary mixtures at a rate $1/k$ where $k$ is the number of components in the mixture when the weight of each newly added component is $1/k$. These results are specific to the Kullback divergence but are consistent with our more general results.

Wang et al. [14] propose an additive procedure similar to ours but with a different reweighting scheme, which is not motivated by a theoretical analysis of optimality conditions. On every new iteration the authors propose to run GAN on the top $k$ training examples with maximum value of the discriminator from the last iteration. Empirical results of Section 4 show that this heuristic often fails to address the missing modes problem.

Finally, many papers investigate completely different approaches for addressing the same issue by directly modifying the training objective of an individual GAN. For instance, Che et al. [5] add an autoencoding cost to the training objective of GAN, while Metz et al. [4] allow the generator to "look few steps ahead" when making a gradient step.

The paper is organized as follows. In Section 2 we present our main theoretical results regarding optimization of mixture models under general $f$-divergences. In particular we show that it is possible to build an optimal mixture in an incremental fashion, where each additional component is obtained by applying a GAN-style procedure with a reweighted distribution. In Section 2.5 we show that if the GAN optimization at each step is perfect, the process converges to the true data distribution at exponential rate (or even in a *finite number of steps*, for which we provide a necessary and sufficient condition). Then we show in Section 2.6 that imperfect GAN solutions still lead to the exponential rate of convergence under certain "weak learnability" conditions. These results naturally lead us to a new boosting-style iterative procedure for constructing generative models, which is combined with GAN in Section 3, resulting in a new algorithm called *AdaGAN*. Finally, we report initial empirical results in Section 4, where we compare AdaGAN with several benchmarks, including original GAN, uniform mixture of multiple independently trained GANs, and iterative procedure of Wang et al. [14].

# 2 Minimizing $f$-divergence with Additive Mixtures

In this section we derive a general result on the minimization of $f$-divergences over mixture models.

## 2.1 Preliminaries and notations

In this work we will write $P_d$ and $P_{model}$ to denote a real data distribution and our approximate model distribution, respectively, both defined over the data space $\mathcal{X}$.

**Generative Density Estimation** In the generative approach to density estimation, instead of building a probabilistic model of the data directly, one builds a function $G : \mathcal{Z} \to \mathcal{X}$ that transforms a fixed probability distribution $P_Z$ (often called the *noise* distribution) over a latent space $\mathcal{Z}$ into a distribution over $\mathcal{X}$. Hence $P_{model}$ is the pushforward of $P_Z$, i.e. $P_{model}(A) = P_Z(G^{-1}(A))$. Because of this definition, it is generally impossible to compute the density $dP_{model}(x)$, hence it is not possible to compute the log-likelihood of the training data under the model. However, if $P_Z$ is a distribution from which one can sample, it is easy to also sample from $P_{model}$ (simply sampling from $P_Z$ and applying $G$ to each example gives a sample from $P_{model}$).

So the problem of generative density estimation becomes a problem of finding a function $G$ such that $P_{model}$ looks like $P_d$ in the sense that samples from $P_{model}$ and from $P_d$ look similar. Another way to state this problem is to say that we are given a measure of similarity between distributions $D(P_{model} \| P_d)$ which

can be estimated from samples of those distributions, and thus approximately minimized over a class $\mathcal{G}$ of functions.

**$f$-Divergences** In order to measure the agreement between the model distribution and the true distribution of the data we will use an $f$-divergence defined in the following way:

$$D_f(Q\|P) := \int f\left(\frac{dQ}{dP}(x)\right) dP(x) \tag{1}$$

for any pair of distributions $P, Q$ with densities $dP$, $dQ$ with respect to some dominating reference measure $\mu$. In this work we assume that the function $f$ is convex, defined on $(0, \infty)$, and satisfies $f(1) = 0$. The definition of $D_f$ holds for both continuous and discrete probability measures and does not depend on specific choice of $\mu$.[3] It is easy to verify that $D_f \geq 0$ and it is equal to 0 when $P = Q$. Note that $D_f$ is not symmetric, but $D_f(P\|Q) = D_{f^\circ}(Q\|P)$ for $f^\circ(x) := xf(1/x)$ and any $P$ and $Q$. The $f$-divergence is *symmetric* when $f(x) = f^\circ(x)$ for all $x \in (0, \infty)$, as in this case $D_f(P, Q) = D_f(Q, P)$.

We also note that the divergences corresponding to $f(x)$ and $f(x) + C \cdot (x - 1)$ are identical for any constant $C$. In some cases, it is thus convenient to work with $f_0(x) := f(x) - (x - 1)f'(1)$, (where $f'(1)$ is any subderivative of $f$ at 1) as $D_f(Q\|P) = D_{f_0}(Q\|P)$ for all $Q$ and $P$, while $f_0$ is nonnegative, nonincreasing on $(0, 1]$, and nondecreasing on $(1, \infty)$. In the remainder, we will denote by $\mathcal{F}$ the set of functions that are suitable for $f$-divergences, i.e. the set of functions of the form $f_0$ for any convex $f$ with $f(1) = 0$.

Classical examples of $f$-divergences include the Kullback-Leibler divergence (obtained for $f(x) = -\log x$, $f_0(x) = -\log x + x - 1$), the reverse Kullback-Leibler divergence (obtained for $f(x) = x \log x$, $f_0(x) = x \log x - x + 1$), the Total Variation distance ($f(x) = f_0(x) = |x - 1|$), and the Jensen-Shannon divergence ($f(x) = f_0(x) = -(x + 1) \log \frac{x+1}{2} + x \log x$). More details can be found in Appendix D. Other examples can be found in [7]. For further details on $f$-divergences we refer to Section 1.3 of [15] and [16].

**GAN and $f$-divergences** We now explain the connection between the GAN algorithm and $f$-divergences. The original GAN algorithm [1] consists in optimizing the following criterion:

$$\min_G \max_D \mathbb{E}_{P_d}\left[\log D(X)\right] + \mathbb{E}_{P_Z}\left[\log(1 - D(G(Z)))\right] , \tag{2}$$

where $D$ and $G$ are two functions represented by neural networks, and this optimization is actually performed on a pair of samples (one being the training sample, the other one being created from the chosen distribution $P_Z$), which corresponds to approximating the above criterion by using the empirical distributions. For a fixed $G$, it has been shown in [1] that the optimal $D$ for (2) is given by $D^*(x) = \frac{dP_d(x)}{dP_d(x)+dP_g(x)}$ and plugging this optimal value into (2) gives the following:

$$\min_G -\log(4) + 2JS(P_d \,\|\, P_g) , \tag{3}$$

where $JS$ is the Jensen-Shannon divergence. Of course, the actual GAN algorithm uses an approximation to $D^*$ which is computed by training a neural network on a sample, which means that the GAN algorithm can be considered to minimize an approximation of (3)[4]. This point of view can be generalized by plugging another $f$-divergence into (3), and it turns out that other $f$-divergences can be written as the solution to a maximization of a criterion similar to (2). Indeed, as demonstrated in [7], any $f$-divergence between $P_d$ and $P_g$ can be seen as the optimal value of a quantity of the form $\mathbb{E}_{P_d}\left[f_1(D(X))\right] + \mathbb{E}_{P_g}\left[f_2(D(G(Z)))\right]$ for appropriate $f_1$ and $f_2$, and thus can be optimized by the same adversarial training technique.

There is thus a strong connection between adversarial training of generative models and minimization of $f$-divergences, and this is why we cast the results of this section in the context of general $f$-divergences.

[3] The integral in (1) is well defined (but may take infinite values) even if $P(dQ = 0) > 0$ or $Q(dP = 0) > 0$. In this case the integral is understood as $D_f(Q\|P) = \int f(dQ/dP)\mathbb{1}_{[dP(x)>0, dQ(x)>0]}dP(x) + f(0)P(dQ = 0) + f^\circ(0)Q(dP = 0)$, where both $f(0)$ and $f^\circ(0)$ may take value $\infty$ [15]. This is especially important in case of GAN, where it is impossible to constrain $P_{model}$ to be absolutely continuous with respect to $P_d$ or vice versa.

[4] Actually the criterion that is minimized is an empirical version of a lower bound of the Jensen-Shannon divergence.

**Hilbertian Metrics** As demonstrated in [17, 18], several commonly used symmetric $f$-divergences are Hilbertian metrics, which in particular means that their square root satisfies the triangle inequality. This is true for the Jensen-Shannon divergence[5] as well as for the Hellinger distance and the Total Variation among others. We will denote by $\mathcal{F}_H$ the set of $f$ functions such that $D_f$ is a Hilbertian metric. For those divergences, we have $D_f(P\|Q) \leq (\sqrt{D_f(P\|R)} + \sqrt{D_f(R\|Q)})^2$.

**Generative Mixture Models** In order to model complex data distributions, it can be convenient to use a mixture model of the following form:

$$P_{model}^T := \sum_{i=1}^{T} \alpha_i P_i, \tag{4}$$

where $\alpha_i \geq 0$, $\sum_i \alpha_i = 1$, and each of the $T$ components is a generative density model. This is very natural in the generative context, since sampling from a mixture corresponds to a two-step sampling, where one first picks the mixture component (according to the multinomial distribution whose parameters are the $\alpha_i$) and then samples from it. Also, this allows to construct complex models from simpler ones.

## 2.2 Incremental Mixture Building

As discussed earlier, in the context of generative modeling, we are given a measure of similarity between distributions. We will restrict ourselves to the case of $f$-divergences. Indeed, for any $f$-divergence, it is possible (as explained for example in [7]) to estimate $D_f(Q\,\|\,P)$ from two samples (one from $Q$, one from $P$) by training a "discriminator" function, i.e. by solving an optimization problem (which is a binary classification problem in the case where the divergence is symmetric[6]). It turns out that the empirical estimate $\hat{D}$ of $D_f(Q\,\|\,P)$ thus obtained provides a criterion for optimizing $Q$ itself. Indeed, $\hat{D}$ is a function of $Y_1, \ldots, Y_n \sim Q$ and $X_1, \ldots, X_n \sim P$, where $Y_i = G(Z_i)$ for some mapping function $G$. Hence it is possible to optimize $\hat{D}$ with respect to $G$ (and in particular compute gradients with respect to the parameters of $G$ if $G$ comes from a smoothly parametrized model such as a neural network).

In this work we thus assume that, given an i.i.d. sample from any unknown distribution $P$ we can construct a simple model $Q \in \mathcal{G}$ which approximately minimizes

$$\min_{Q \in \mathcal{G}} D_f(Q\,\|\,P). \tag{5}$$

Instead of just modelling the data with a single distribution, we now want to model it with a mixture of the form (4) where each $P_i$ is obtained by a training procedure of the form (5) with (possibly) different target distributions $P$ for each $i$.

A natural way to build a mixture is to do it incrementally: we train the first model $P_1$ to minimize $D_f(P_1\,\|\,P_d)$ and set the corresponding weight to $\alpha_1 = 1$, leading to $P_{model}^1 = P_1$. Then after having trained $t$ components $P_1, \ldots, P_t \in \mathcal{G}$ we can form the $(t+1)$-st mixture model by adding a new component $Q$ with weight $\beta$ as follows:

$$P_{model}^{t+1} := \sum_{i=1}^{t} (1-\beta)\alpha_i P_i + \beta Q. \tag{6}$$

We are going to choose $\beta \in [0, 1]$ and $Q \in \mathcal{G}$ greedily, while keeping all the other parameters of the generative model fixed, so as to minimize

$$D_f((1-\beta)P_g + \beta Q\,\|\,P_d), \tag{7}$$

where we denoted $P_g := P_{model}^t$ the current generative mixture model before adding the new component.

We do not necessarily need to find the optimal $Q$ that minimizes (7) at each step. Indeed, it would be sufficient to find some $Q$ which allows to build a slightly better approximation of $P_d$. This means that a

[5] which means such a property can be used in the context of the original GAN algorithm.

[6] One example of such a setting is running GANs, which are known to approximately minimize the Jensen-Shannon divergence.

more modest goal could be to find $Q$ such that, for some $c < 1$,

$$D_f((1-\beta)P_g + \beta Q \,\|\, P_d) \leq c \cdot D_f(P_g \,\|\, P_d)\,. \tag{8}$$

However, we observe that this greedy approach has a significant drawback in practice. Indeed, as we build up the mixture, we need to make $\beta$ decrease (as $P^t_{model}$ approximates $P_d$ better and better, one should make the correction at each step smaller and smaller). Since we are approximating (7) using samples from both distributions, this means that the sample from the mixture will only contain a fraction $\beta$ of examples from $Q$. So, as $t$ increases, getting meaningful information from a sample so as to tune $Q$ becomes harder and harder (the information is "diluted").

To address this issue, we propose to optimize an upper bound on (7) which involves a term of the form $D_f(Q \,\|\, Q_0)$ for some distribution $Q_0$, which can be computed as a reweighting of the original data distribution $P_d$.

In the following sections we will analyze the properties of (7) (Section 2.4) and derive upper bounds that provide practical optimization criteria for building the mixture (Section 2.3). We will also show that under certain assumptions, the minimization of the upper bound will lead to the optimum of the original criterion.

This procedure is reminiscent of the AdaBoost algorithm [6], which combines multiple *weak* predictors into one very accurate *strong* composition. On each step AdaBoost adds one new predictor to the current composition, which is trained to minimize the binary loss on the reweighted training set. The weights are constantly updated in order to bias the next weak learner towards "hard" examples, which were incorrectly classified during previous stages.

## 2.3 Upper Bounds

Next lemma provides two upper bounds on the divergence of the mixture in terms of the divergence of the additive component $Q$ with respect to some reference distribution $R$.

**Lemma 1** *Let $f \in \mathcal{F}$. Given two distributions $P_d, P_g$ and some $\beta \in [0,1]$, for any distribution $Q$ and any distribution $R$ such that $\beta dR \leq dP_d$, we have*

$$D_f((1-\beta)P_g + \beta Q \,\|\, P_d) \leq \beta D(Q \,\|\, R) + (1-\beta) D_f\left(P_g \,\|\, \frac{P_d - \beta R}{1-\beta}\right). \tag{9}$$

*If furthermore $f \in \mathcal{F}_H$, then, for any $R$, we have*

$$D_f((1-\beta)P_g + \beta Q \,\|\, P_d) \leq \left(\sqrt{\beta D_f(Q \,\|\, R)} + \sqrt{D_f((1-\beta)P_g + \beta R \,\|\, P_d)}\right)^2. \tag{10}$$

**Proof** For the first inequality, we use the fact that $D_f$ is jointly convex. We write $P_d = (1-\beta)\frac{P_d - \beta R}{1-\beta} + \beta R$ which is a convex combination of two distributions when the assumptions are satisfied.

The second inequality follows from using the triangle inequality for the square root of the Hilbertian metric $D_f$ and using convexity of $D_f$ in its first argument. ∎

We can exploit the upper bounds of Lemma 1 by introducing some well-chosen distribution $R$ and minimizing with respect to $Q$. A natural choice for $R$ is a distribution that minimizes the last term of the upper bound (which does not depend on $Q$).

## 2.4 Optimal Upper Bounds

In this section we provide general theorems about the optimization of the right-most terms in the upper bounds of Lemma 1.

For the upper bound (10), this means we need to find $R$ minimizing $D_f((1-\beta)P_g + \beta R \,\|\, P_d)$. The solution for this problem is given in the following theorem.

**Theorem 1** *For any $f$-divergence $D_f$, with $f \in \mathcal{F}$ and $f$ differentiable, any fixed distributions $P_d, P_g$, and any $\beta \in (0, 1]$, the solution to the following minimization problem:*

$$\min_{Q \in \mathbb{P}} D_f((1-\beta)P_g + \beta Q \,\|\, P_d),$$

*where $\mathbb{P}$ is a class of all probability distributions, has the density*

$$dQ^*_\beta(x) = \frac{1}{\beta}\left(\lambda^* dP_d(x) - (1-\beta)dP_g(x)\right)_+$$

*for some unique $\lambda^*$ satisfying $\int dQ^*_\beta = 1$. Furthermore, $\beta \le \lambda^* \le \min(1, \beta/\delta)$, where $\delta := P_d(dP_g = 0)$. Also, $\lambda^* = 1$ if and only if $P_d((1-\beta)dP_g > dP_d) = 0$, which is equivalent to $\beta dQ^*_\beta = dP_d - (1-\beta)dP_g$.*

**Proof** See Appendix C.1. ∎

**Remark 1** *The form of $Q^*_\beta$ may look unexpected at first glance: why not setting $dQ := \big(dP_d - (1-\beta)dP_g\big)/\beta$, which would make arguments of the $f$-divergence identical? Unfortunately, it may be the case that $dP_d(X) < (1-\beta)dP_g(X)$ for some of $X \in \mathcal{X}$, leading to the negative values of $dQ$.*

For the upper bound (9), we need to minimize $D_f\Big(P_g \,\|\, \frac{P_d - \beta R}{1-\beta}\Big)$. The solution is given in the next theorem.

**Theorem 2** *Given two distributions $P_d, P_g$ and some $\beta \in (0, 1]$, assume*

$$P_d\left(dP_g = 0\right) < \beta.$$

*Let $f \in \mathcal{F}$. The solution to the minimization problem*

$$\min_{Q : \beta dQ \le dP_d} D_f\left(P_g \,\|\, \frac{P_d - \beta Q}{1-\beta}\right)$$

*is given by the distribution*

$$dQ^\dagger_\beta(x) = \frac{1}{\beta}\big(dP_d(x) - \lambda^\dagger(1-\beta)dP_g(x)\big)_+$$

*for a unique $\lambda^\dagger \ge 1$ satisfying $\int dQ^\dagger_\beta = 1$.*

**Proof** See Appendix C.2. ∎

**Remark 2** *Notice that the term that we optimized in upper bound* (10) *is exactly the initial objective* (7)*. So that Theorem 1 also tells us what the form of the optimal distribution is for the initial objective.*

**Remark 3** *Surprisingly, in both Theorem 1 and 2, the solution does not depend on the choice of the function $f$, which means that the solution is the same for* any *$f$-divergence. This also means that by replacing $f$ by $f^\circ$, we get similar results for the criterion written in the other direction, with again the same solution. Hence the order in which we write the divergence does not matter and the optimal solution is optimal for both orders.*

**Remark 4** *Note that $\lambda^*$ is implicitly defined by a fixed-point equation. In Section 3.1 we will show how it can be computed efficiently in the case of empirical distributions.*

**Remark 5** *Obviously, $\lambda^\dagger \ge \lambda^*$, where $\lambda^*$ was defined in Theorem 1. Moreover, we have $\lambda^* \le 1/\lambda^\dagger$. Indeed, it is enough to insert $\lambda^\dagger = 1/\lambda^*$ into definition of $Q^\dagger_\beta$ and check that in this case $Q^\dagger_\beta \ge 1$.*

## 2.5 Convergence Analysis for Optimal Updates

In previous section we derived analytical expressions for the distributions $R$ minimizing last terms in upper bounds (9) and (10). Assuming $Q$ can perfectly match $R$, i.e. $D_f(Q \,\|\, R) = 0$, we are now interested in the convergence of the mixture (6) to the true data distribution $P_d$ for $Q = Q^*_\beta$ or $Q = Q^\dagger_\beta$.

We start with simple results showing that adding $Q^*_\beta$ or $Q^\dagger_\beta$ to the current mixture would yield a strict improvement of the divergence.

**Lemma 2** *Under the conditions of Theorem 1, we have*

$$D_f\big((1-\beta)P_g + \beta Q^*_\beta \,\big\|\, P_d\big) \leq D_f\big((1-\beta)P_g + \beta P_d \,\big\|\, P_d\big) \leq (1-\beta) D_f(P_g \,\|\, P_d). \tag{11}$$

*Under the conditions of Theorem 2, we have*

$$D_f\left(P_g \,\Big\|\, \frac{P_d - \beta Q^\dagger_\beta}{1-\beta}\right) \leq D_f(P_g \,\|\, P_d)\,,$$

*and*

$$D_f\big((1-\beta)P_g + \beta Q^\dagger_\beta \,\big\|\, P_d\big) \leq (1-\beta) D_f(P_g \,\|\, P_d).$$

**Proof** The first inequality follows immediately from the optimality of $Q^*_\beta$ (hence the value of the objective at $Q^*_\beta$ is smaller than at $P_d$), and the fact that $D_f$ is convex in its first argument and $D_f(P_d \| P_d) = 0$. The second inequality follows from the optimality of $Q^\dagger_\beta$ (hence the value of the objective at $Q^\dagger_\beta$ is smaller than its value at $P_d$ which itself satisfies the condition $\beta dP_d \leq dP_d$). For the third inequality, we combine the second inequality with the first inequality of Lemma 1 (with $Q = R = Q^\dagger_\beta$). ∎

The upper bound (11) of Lemma 2 can be refined if the ratio $dP_g/dP_d$ is almost surely bounded:

**Lemma 3** *Under the conditions of Theorem 1, if there exists $M > 1$ such that*

$$P_d((1-\beta)dP_g > M dP_d) = 0$$

*then*

$$D_f\big((1-\beta)P_g + \beta Q^*_\beta \,\big\|\, P_d\big) \leq f(\lambda^*) + \frac{f(M)(1-\lambda^*)}{M-1}.$$

**Proof** We use Inequality (20) of Lemma 6 with $X = \beta$, $Y = (1-\beta)dP_g/dP_d$, and $c = \lambda^*$. We easily verify that $X + Y = ((1-\beta)dP_g + \beta dP_d)/dP_d$ and $\max(c, Y) = ((1-\beta)dP_g + \beta dQ^*_\beta)/dP_d$ and both have expectation 1 with respect to $P_d$. We thus obtain:

$$D_f((1-\beta)P_g + \beta Q^*_\beta \,\|\, P_d) \leq f(\lambda^*) + \frac{f(M) - f(\lambda^*)}{M - \lambda^*}(1-\lambda^*)\,.$$

Since $\lambda^* \leq 1$ and $f$ is non-increasing on $(0, 1)$ we get

$$D_f((1-\beta)P_g + \beta Q^*_\beta \,\|\, P_d) \leq f(\lambda^*) + \frac{f(M)(1-\lambda^*)}{M-1}.$$

∎

**Remark 6** *This upper bound can be tighter than that of Lemma 2 when $\lambda^*$ gets close to 1. Indeed, for $\lambda^* = 1$ the upper bound is exactly 0 and is thus tight, while the upper bound of Lemma 2 will not be zero in this case.*

Imagine repeatedly adding $T$ new components to the current mixture $P_g$, where on every step we use the same weight $\beta$ and choose the components described in Theorem 1. In this case Lemma 2 guarantees that the original objective value $D_f(P_g \,\|\, P_d)$ would be reduced at least to $(1-\beta)^T D_f(P_g \,\|\, P_d)$. This exponential rate of convergence, which at first may look surprisingly good, is simply explained by the fact that $Q^*_\beta$ depends on the true distribution $P_d$, which is of course unknown.

Lemma 2 also suggests setting $\beta$ as large as possible. This is intuitively clear: the smaller the $\beta$, the less we alter our current model $P_g$. As a consequence, choosing small $\beta$ when $P_g$ is far away from $P_d$ would lead to only minor improvements in objective (7). In fact, the global minimum of (7) can be reached by setting $\beta = 1$ and $Q = P_d$. Nevertheless, in practice we may prefer to keep $\beta$ relatively small, preserving what we learned so far through $P_g$: for instance, when $P_g$ already covered part of the modes of $P_d$ and we want $Q$ to cover the remaining ones. We provide further discussions on choosing $\beta$ in Section 3.2.

In the reminder of this section we study the convergence of (7) to 0 in the case where we use the upper bound (10) and the weight $\beta$ is fixed (i.e. the same value at each iteration). This analysis can easily be extended to a variable $\beta$.

**Lemma 4** *For any $f \in \mathcal{F}$ such that $f(x) \neq 0$ for $x \neq 1$, the following conditions are equivalent:*

*(i)* $P_d((1-\beta)dP_g > dP_d) = 0$;

*(ii)* $D_f((1-\beta)P_g + \beta Q^*_\beta \,\|\, P_d) = 0$.

**Proof** The first condition is equivalent to $\lambda^* = 1$ according to Theorem 1. In this case, $(1-\beta)P_g + \beta Q^*_\beta = P_d$, hence the divergence is 0. In the other direction, when the divergence is 0, since $f$ is strictly positive for $x \neq 1$ (keep in mind that we can always replace $f$ by $f_0$ to get a non-negative function which will be strictly positive if $f(x) \neq 0$ for $x \neq 1$), this means that with $P_d$ probability 1 we have the equality $dP_d = (1-\beta)dP_g + \beta dQ^*_\beta$, which implies that $(1-\beta)dP_g > dP_d$ with $P_d$ probability 1 and also $\lambda^* = 1$. ∎

This result tells that we can not perfectly match $P_d$ by adding a new mixture component to $P_g$ as long as there are points in the space where our current model $P_g$ severely over-samples. As an example, consider an extreme case where $P_g$ puts a positive mass in a region outside of the support of $P_d$. Clearly, unless $\beta = 1$, we will not be able to match $P_d$.

Finally, we provide a necessary and sufficient condition for the iterative process to converge to the data distribution $P_d$ in finite number of steps. The criterion is based on the ratio $dP_1/dP_d$, where $P_1$ is the first component of our mixture model.

**Corollary 1** *Take any $f \in \mathcal{F}$ such that $f(x) \neq 0$ for $x \neq 1$. Starting from $P^1_{model} = P_1$, update the model iteratively according to $P^{t+1}_{model} = (1-\beta)P^t_{model} + \beta Q^*_\beta$, where on every step $Q^*_\beta$ is as defined in Theorem 1 with $P_g := P^t_{model}$. In this case $D_f(P^t_{model} \,\|\, P_d)$ will reach 0 in a finite number of steps if and only if there exists $M > 0$ such that*

$$P_d((1-\beta)dP_1 > M dP_d) = 0\,. \tag{12}$$

*When the finite convergence happens, it takes at most $-\ln \max(M, 1)/\ln(1-\beta)$ steps.*

**Proof** From Lemma 4, it is clear that if $M \leq 1$ the convergence happens after the first update. So let us assume $M > 1$. Notice that $dP^{t+1}_{model} = (1-\beta)dP^t_{model} + \beta dQ^*_\beta = \max(\lambda^* dP_d, (1-\beta)dP^t_{model})$ so that if $P_d((1-\beta)dP^t_{model} > M dP_d) = 0$, then $P_d((1-\beta)dP^{t+1}_{model} > M(1-\beta)dP_d) = 0$. This proves that (12) is a sufficient condition.

Now assume the process converged in a finite number of steps. Let $P^t_{model}$ be a mixture right before the final step. Note that $P^t_{model}$ is represented by $(1-\beta)^{t-1}P_1 + (1-(1-\beta)^{t-1})P$ for certain probability distribution $P$. According to Lemma 4 we have $P_d((1-\beta)dP^t_{model} > dP_d) = 0$. Together these two facts immediately imply (12). ∎

It is also important to keep in mind that even if (12) is not satisfied the process still converges to the true distribution at exponential rate (see Lemma 2 as well as Corollaries 2 and 3 below)

## 2.6 Weak to Strong Learnability

In practice the component $Q$ that we add to the mixture is not exactly $Q^*_\beta$ or $Q^\dagger_\beta$, but rather an approximation to them. We need to show that if this approximation is good enough, then we retain the property that (8) is reached. In this section we will show that this is indeed the case.

Looking again at Lemma 1 we notice that the first upper bound is less tight than the second one. Indeed, take the optimal distributions provided by Theorems 1 and 2 and plug them back as $R$ into the upper bounds of Lemma 1. Also assume that $Q$ can match $R$ exactly, i.e. we can achieve $D_f(Q \,\|\, R) = 0$. In this case both sides of (10) are equal to $D_f((1-\beta)P_g + \beta Q^*_\beta \,\|\, P_d)$, which is the optimal value for the original objective (7). On the other hand, (9) does not become an equality and the r.h.s. is not the optimal one for (7).

This means that using (10) allows to reach the optimal value of the original objective (7), whereas using (9) does not. However, this is not such a big issue since, as we mentioned earlier, we only need to improve the mixture by adding the next component (we do not need to add the optimal next component). So despite the solution of (7) not being reachable with the first upper bound, we will still show that (8) can be reached.

The first result provides sufficient conditions for strict improvements when we use the upper bound (9).

**Corollary 2** *Given two distributions $P_d, P_g$, and some $\beta \in (0, 1]$, assume*

$$P_d\left(\frac{dP_g}{dP_d} = 0\right) < \beta. \tag{13}$$

*Let $Q^\dagger_\beta$ be as defined in Theorem 2. If $Q$ is a distribution satisfying*

$$D_f(Q \,\|\, Q^\dagger_\beta) \leq \gamma D_f(P_g \,\|\, P_d) \tag{14}$$

*for $\gamma \in [0, 1]$ then*

$$D_f((1-\beta)P_g + \beta Q \,\|\, P_d) \leq (1 - \beta(1-\gamma)) D_f(P_g \,\|\, P_d).$$

**Proof** Immediately follows from combining Lemma 1, Theorem 1, and Lemma 2. ∎

Next one holds for Hilbertian metrics and corresponds to the upper bound (10).

**Corollary 3** *Assume $f \in \mathcal{F}_H$, i.e. $D_f$ is a Hilbertian metric. Take any $\beta \in (0, 1]$, $P_d$, $P_g$, and let $Q^*_\beta$ be as defined in Theorem 1. If $Q$ is a distribution satisfying*

$$D_f(Q \,\|\, Q^*_\beta) \leq \gamma D_f(P_g \,\|\, P_d) \tag{15}$$

*for some $\gamma \in [0, 1]$, then*

$$D_f((1-\beta)P_g + \beta Q \,\|\, P_d) \leq \left(\sqrt{\gamma\beta} + \sqrt{1-\beta}\right)^2 D_f(P_g \,\|\, P_d)\,.$$

*In particular, the right-hand side is strictly smaller than $D_f(P_g \,\|\, P_d)$ as soon as $\gamma < \beta/4$ (and $\beta > 0$).*

**Proof** Immediately follows from combining Lemma 1, Theorem 2, and Lemma 2. It is easy to verify that for $\gamma < \beta/4$, the coefficient is less than $(\beta/2 + \sqrt{1-\beta})^2$ which is $< 1$ (for $\beta > 0$). ∎

**Remark 7** *We emphasize once again that the upper bound* (10) *and Corollary 3 both hold for Jensen-Shannon, Hellinger, and total variation divergences among others. In particular they can be applied to the original GAN algorithm.*

Conditions 14 and 15 may be compared to the "weak learnability" condition of AdaBoost. As long as our weak learner is able to solve the surrogate problem (5) of matching respectively $Q^\dagger_\beta$ or $Q^*_\beta$ accurately enough, the original objective (7) is guaranteed to decrease as well. It should be however noted that Condition 15

with $\gamma < \beta/4$ is perhaps too strong to call it "weak learnability". Indeed, as already mentioned before, the weight $\beta$ is expected to decrease to zero as the number of components in the mixture distribution $P_g$ increases. This leads to $\gamma \to 0$, making it harder to meet Condition 15. This obstacle may be partially resolved by the fact that we will use a GAN to fit $Q$, which corresponds to a relatively rich[7] class of models $\mathcal{G}$ in (5). In other words, our weak learner is not so weak.

On the other hand, Condition (14) of Corollary 2 is much milder. No matter what $\gamma \in [0, 1]$ and $\beta \in (0, 1]$ we choose, the new component $Q$ is guaranteed to strictly improve the objective functional. This comes at the price of the additional Condition (13), which asserts that $\beta$ should be larger than the mass of true data $P_d$ missed by the current model $P_g$. We argue that this is a rather reasonable condition: if $P_g$ misses many modes of $P_d$ we would prefer assigning a relatively large weight $\beta$ to the new component $Q$.

# 3 AdaGAN

In this section we provide a more detailed description of Algorithm 1 from Section 1.1, in particular how to reweight the training examples for the next iteration and how to choose the mixture weights.

In a nutshell, at each iteration we want to add a new component $Q$ to the current mixture $P_g$ with weight $\beta$, to create a mixture with distribution $(1-\beta)P_g + \beta Q$. This component $Q$ should approach an "optimal target" $Q^*_\beta$ and we know from Theorem 1 that:

$$dQ^*_\beta = \frac{dP_d}{\beta}\left(\lambda^* - (1-\beta)\frac{dP_g}{dP_d}\right)_+ .$$

Computing this distribution requires to know the density ratio $dP_g/dP_d$, which is not directly accessible, but it can be estimated using the idea of adversarial training. Indeed, we can train a discriminator $D$ to distinguish between samples from $P_d$ and $P_g$. It is known that for an arbitrary $f$-divergence, there exists a corresponding function $h$ (see [7]) such that the values of the optimal discriminator $D$ are related to the density ratio in the following way:

$$\frac{dP_g}{dP_d}(X) = h\big(D(X)\big). \tag{16}$$

In particular, for the Jensen-Shannon divergence, used by the original GAN algorithm, it holds that $h\big(D(X)\big) = \frac{1-D(X)}{D(X)}$. So in this case for the optimal discriminator we have

$$dQ^*_\beta = \frac{dP_d}{\beta}\big(\lambda^* - (1-\beta)h(D)\big)_+ ,$$

which can be viewed as a reweighted version of the original data distribution $P_d$.

In particular, when we compute $dQ^*_\beta$ on the training sample $S_N = (X_1, \ldots, X_N)$, each example $X_i$ has the following weight:

$$w_i = \frac{p_i}{\beta}\big(\lambda^* - (1-\beta)h(d_i)\big)_+ \tag{17}$$

with $p_i = dP_d(X_i)$ and $d_i = D(X_i)$. In practice, we use the empirical distribution over the training sample which means we set $p_i = 1/N$.

## 3.1 How to compute $\lambda^*$ of Theorem 1

Next we derive an algorithm to determine $\lambda^*$. We need to find a value of $\lambda^*$ such that the weights $w_i$ in (17) are normalized, i.e.:

$$\sum_i w_i = \sum_{i \in \mathcal{I}(\lambda^*)} \frac{p_i}{\beta}\big(\lambda^* - (1-\beta)h(d_i)\big) = 1 ,$$

[7]The hardness of meeting Condition 15 of course largely depends on the class of models $\mathcal{G}$ used to fit $Q$ in (5). For now we ignore this question and leave it for future research.

where $\mathcal{I}(\lambda) := \{i : \lambda > (1-\beta)h(d_i)\}$. This in turn yields:

$$\lambda^* = \frac{\beta}{\sum_{i\in\mathcal{I}(\lambda^*)} p_i} \left(1 + \frac{(1-\beta)}{\beta} \sum_{i\in\mathcal{I}(\lambda^*)} p_i h(d_i)\right) . \tag{18}$$

Now, to compute the r.h.s., we need to know $\mathcal{I}(\lambda^*)$. To do so, we sort the values $h(d_i)$ in increasing order: $h(d_1) \leq h(d_2) \leq \ldots \leq h(d_N)$. Then $\mathcal{I}(\lambda^*)$ is simply a set consisting of the first $k$ values, where we have to determine $k$. Thus, it suffices to test successively all positive integers $k$ until the $\lambda$ given by Equation (18) verifies:

$$(1-\beta)h(d_k) < \lambda \leq (1-\beta)h(d_{k+1}) .$$

This procedure is guaranteed to converge, because by Theorem 1, we know that $\lambda^*$ exists, and it satisfies (18). In summary, $\lambda^*$ can be determined by Algorithm 2.

**Algorithm 2:** Determining $\lambda^*$

Sort the values $h(d_i)$ in increasing order ;
Initialize $\lambda \leftarrow \frac{\beta}{p_1}\left(1 + \frac{1-\beta}{\beta} p_1 h(d_1)\right)$ and $k \leftarrow 1$ ;
**while** $(1-\beta)h(d_k) \geq \lambda$ **do**
    $k \leftarrow k+1$;
    $\lambda \leftarrow \frac{\beta}{\sum_{i=1}^{k} p_i}\left(1 + \frac{(1-\beta)}{\beta} \sum_{i=1}^{k} p_i h(d_i)\right)$

### 3.2 How to choose a mixture weight $\beta$

While for every $\beta$ there is an optimal reweighting scheme, the weights from (17) depend on $\beta$. In particular, if $\beta$ is large enough to verify $dP_d(x)\lambda^* - (1-\beta)dP_g(x) \geq 0$ for all $x$, the optimal component $Q^*_\beta$ satisfies $(1-\beta)P_g + \beta Q^*_\beta = P_d$, as proved in Lemma 4. In other words, in this case we exactly match the data distribution $P_d$, assuming the GAN can approximate the target $Q^*_\beta$ perfectly. This criterion alone would lead to choosing $\beta = 1$. However in practice we know we can't get a generator that produces exactly the target distribution $Q^*_\beta$. We thus propose a few heuristics one can follow to choose $\beta$:

- Any fixed constant value $\beta$ for all iterations.
- All generators to be combined with equal weights in the final mixture model. This corresponds to setting $\beta_t = \frac{1}{t}$, where $t$ is the iteration.
- Instead of choosing directly a value for $\beta$ one could pick a ratio $0 < r < 1$ of examples which should have a weight $w_i > 0$. Given such an $r$, there is a unique value of $\beta$ ($\beta_r$) resulting in $w_i > 0$ for exactly $N \cdot r$ training examples. Such a value $\beta_r$ can be determined by binary search over $\beta$ in Algorithm 2. Possible choices for $r$ include:
    - $r$ constant, chosen experimentally.
    - $r$ decreasing with the number of iterations, e.g., $r = c_1 e^{-c_2 t}$ for any positive constants $c_1, c_2$.
- Alternatively, one can set a particular threshold for the density ratio estimate $h(D)$, compute the fraction $r$ of training examples that have a value above that threshold and derive $\beta$ from this ratio $r$ (as above). Indeed, when $h(D)$ is large, that means that the generator does not generate enough examples in that region, and the next iteration should be encouraged to generate more there.

**Algorithm 3:** AdaGAN, a meta-algorithm to construct a "strong" mixture of $T$ individual GANs, trained sequentially. The mixture weight schedule ChooseMixtureWeight should be provided by the user (see 3.2). This is an instance of the high level Algorithm 1, instantiating UpdateTrainingWeights.

**Input:** Training sample $S_N := \{X_1, \dots, X_N\}$.

**Output:** Mixture generative model $G = G_T$.

Train vanilla GAN: $G_1 = \text{GAN}(S_N)$

**for** $t = 2, \dots, T$ **do**

*#Choose a mixture weight for the next component*

$\beta_t = \text{ChooseMixtureWeight}(t)$

*#Compute the new weights of the training examples (UpdateTrainingWeights)*

*#Compute the discriminator between the original (unweighted) data and the current mixture* $G_{t-1}$

$D \leftarrow DGAN(S_N, G_{t-1})$;

*#Compute* $\lambda^*$ *using Algorithm 2*

$\lambda^* \leftarrow \lambda(\beta_t, D)$

*#Compute the new weight for each example*

**for** $i = 1, \dots, N$ **do**

$W_t^i = \frac{1}{N\beta_t} \left(\lambda^* - (1-\beta_t) h(D(X_i))\right)_+$

**end for**

*#Train t-th "weak" component generator* $G_t^c$

$G_t^c = \text{GAN}(S_N, W_t)$

*#Update the overall generative model*

*#Notation below means forming a mixture of* $G_{t-1}$ *and* $G_t^c$.

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

**end for**

### 3.3 Complete algorithm

Now we have all the necessary components to introduce the complete AdaGAN meta-algorithm. The algorithm uses any given GAN implementation (which can be the original one of Goodfellow et al. [1] or any later modifications) as a building block. Accordingly, $G^c \leftarrow GAN(S_N, W)$ returns a generator $G^c$ for a given set of examples $S_N = (X_1, \dots, X_N)$ and corresponding weights $W = (w_1, \dots, w_N)$. Additionally, we write $D \leftarrow DGAN(S_N, G)$ to denote a procedure that returns a discriminator from the GAN algorithm trained on a given set of true data examples $S_N$ and examples sampled from the mixture of generators $G$. We also write $\lambda^*(\beta, D)$ to denote the optimal $\lambda^*$ given by Algorithm 2. The complete algorithm is presented in Algorithm 3.

## 4 Experiments

We tested AdaGAN[8] on toy datasets, for which we can interpret the missing modes in a clear and reproducible way, and on MNIST, which is a high-dimensional dataset. The goal of these experiments *was not* to evaluate

[8]Code available online at `https://github.com/tolstikhin/adagan`

the visual quality of individual sample points, but to demonstrate that the re-weighting scheme of AdaGAN promotes diversity and effectively covers the missing modes.

## 4.1 Toy datasets

The target distribution is defined as a mixture of normal distributions, with different variances. The distances between the means are relatively large compared to the variances, so that each Gaussian of the mixture is "isolated". We vary the number of modes to test how well each algorithm performs when there are fewer or more expected modes.

More precisely, we set $\mathcal{X} = \mathcal{R}^2$, each Gaussian component is isotropic, and their centers are sampled uniformly in a square. That particular random seed is fixed for all experiments, which means that for a given number of modes, the target distribution is always the same. The variance parameter is the same for each component, and is decreasing with the number of modes, so that the modes stay apart from each other.

This target density is very easy to learn, using a mixture of Gaussians model, and for example the EM algorithm [19]. If applied to the situation where the generator is producing single Gaussians (i.e. $P_Z$ is a standard Gaussian and $G$ is a linear function), then AdaGAN produces a mixture of Gaussians, however it does so incrementally unlike EM, which keeps a fixed number of components. In any way AdaGAN was not tailored for this particular case and we use the Gaussian mixture model simply as a toy example to illustrate the missing modes problem.

### 4.1.1 Algorithms

We compare different meta-algorithms based on GAN, and the baseline GAN algorithm. All the meta-algorithms use the same implementation of the underlying GAN procedure. In all cases, the generator uses latent space $\mathcal{Z} = \mathcal{R}^5$, and two ReLU hidden layers, of size 10 and 5 respectively. The corresponding discriminator has two ReLU hidden layers of size 20 and 10 respectively. We use 64k training examples, and 15 epochs, which is enough compared to the small scale of the problem, and all networks converge properly and overfitting is never an issue. Despite the simplicity of the problem, there are already differences between the different approaches.

We compare the following algorithms:

- The baseline GAN algorithm, called **Vanilla GAN** in the results.
- The best model out of $T$ runs of GAN, that is: run $T$ GAN instances independently, then take the run that performs best on a validation set. This gives an additional baseline with similar computational complexity as the ensemble approaches. Note that the selection of the best run is done on the reported target metric (see below), rather than on the internal metric. As a result this baseline is slightly overestimated. This procedure is called **Best of T** in the results.
- A mixture of $T$ GAN generators, trained independently, and combined with equal weights (the "bagging" approach). This procedure is called **Ensemble** in the results.
- A mixture of GAN generators, trained sequentially with different choices of data reweighting:
  - The AdaGAN algorithm (Algorithm 1), for $\beta = 1/t$, i.e. each component will have the same weight in the resulting mixture (see § 3.2). This procedure is called **Boosted** in the results.
  - The AdaGAN algorithm (Algorithm 1), for a constant $\beta$, exploring several values. This procedure is called for example **Beta0.3** for $\beta = 0.3$ in the results.
  - Reweighting similar to "Cascade GAN" from [14], i.e. keeping the top $r$ fraction of examples, based on the discriminator corresponding to the *previous* generator. This procedure is called for example **TopKLast0.3** for $r = 0.3$.
  - Keep the top $r$ fraction of examples, based on the discriminator corresponding to *the mixture of all previous* generators. This procedure is called for example **TopK0.3** for $r = 0.3$.

#### 4.1.2 Metrics

To evaluate how well the generated distribution matches the target distribution, we use a *coverage* metric $C$. We compute the probability mass of the true data "covered" by the model distribution $P_{model}$. More precisely, we compute $C := P_d(dP_{model} > t)$ with $t$ such that $P_{model}(dP_{model} > t) = 0.95$. This metric is more interpretable than the likelihood, making it easier to assess the difference in performance of the algorithms. To approximate the density of $P_{model}$ we use a kernel density estimation method, where the bandwidth is chosen by cross validation. Note that we could also use the discriminator $D$ to approximate the coverage as well, using the relation from (16).

Another metric is the likelihood of the true data under the generated distribution. More precisely, we compute $L := \frac{1}{N}\sum_i \log P_{model}(x_i)$, on a sample of $N$ examples from the data. Note that [20] proposes a more general and elegant approach (but less straightforward to implement) to have an objective measure of GAN. On the simple problems we tackle here, we can precisely estimate the likelihood.

In the main results we report the metric $C$ and in Appendix E we report both $L$ and $C$. For a given metric, we repeat the run 35 times with the same parameters (but different random seeds). For each run, the learning rate is optimized using a grid search on a validation set. We report the median over those multiple runs, and the interval corresponding to the 5% and 95% percentiles. Note this is not a *confidence interval* of the median, which would shrink to a singleton with an infinite number of runs. Instead, this gives a measure of the stability of each algorithm. The optimizer is a simple SGD: Adam was also tried but gave slightly less stable results.

#### 4.1.3 Results

With the vanilla GAN algorithm, we observe that not all the modes are covered (see Figure 1 for an illustration). Different modes (and even different number of modes) are possibly covered at each restart of the algorithm, so restarting the algorithm with different random seeds and taking the best ("best of $T$") can improve the results.

Figure 3 summarizes the performance of the main algorithms on the $C$ metric, as a function of the number of iterations $T$. Table 1 gives more detailed results, varying the number of modes for the target distribution. Appendix E contains details on variants for the reweighting heuristics as well as results for the $L$ metric.

As expected, both the ensemble and the boosting approaches significantly outperform the vanilla GAN and the "best of $T$" algorithm. Interestingly, the improvements are significant even after just one or two additional iterations ($T = 2$ or $T = 3$). The boosted approach converges much faster. In addition, the variance is much lower, improving the likelihood that a given run gives good results. On this setup, the vanilla GAN approach has a significant number of catastrophic failures (visible in the lower bound of the interval).

Empirical results on combining AdaGAN meta-algorithm with the unrolled GANs [4] are available in Appendix A.

### 4.2 MNIST and MNIST3

We ran experiments both on the original MNIST and on the 3-digit MNIST (MNIST3) [5, 4] dataset, obtained by concatenating 3 randomly chosen MNIST images to form a 3-digit number between 0 and 999. According to [5, 4], MNIST contains 10 modes, while MNIST3 contains 1000 modes, and these modes can be detected using the pre-trained MNIST classifier. We combined AdaGAN both with simple MLP GANs and DCGANs [21]. We used $T \in \{5, 10\}$, tried models of various sizes and performed a reasonable amount of hyperparameter search. For the details we refer to Appendix B.

Similarly to [4, Sec 3.3.1] we failed to reproduce the missing modes problem for MNIST3 reported in [5] and found that simple GAN architectures are capable of generating all 1000 numbers. The authors of [4] proposed to artificially introduce the missing modes again by limiting the generators' flexibility. In our experiments, GANs trained with the architectures reported in [4] were often generating poorly looking digits. As a result, the pre-trained MNIST classifier was outputting random labels, which again led to full

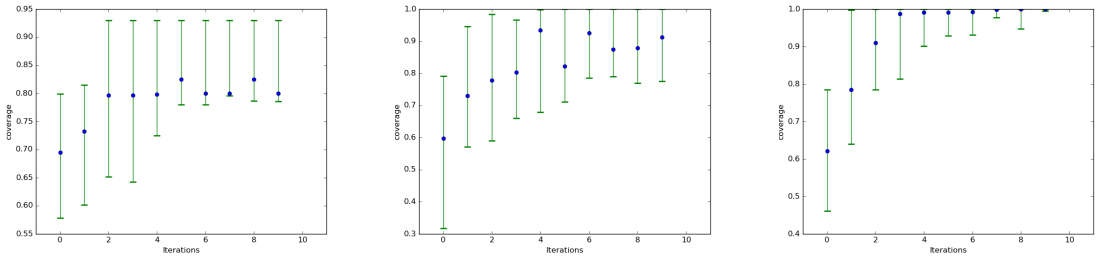


Figure 2: Coverage $C$ of the true data by the model distribution $P^T_{model}$, as a function of iterations $T$. Experiments correspond to the data distribution with 5 modes. Each blue point is the median over 35 runs. Green intervals are defined by the 5% and 95% percentiles (see Section 4.1.2). Iteration 0 is equivalent to one vanilla GAN. The left plot corresponds to taking the best generator out of $T$ runs. The middle plot corresponds to the "ensemble GAN", simply taking a uniform mixture of $T$ independently trained GAN generators. The right plot corresponds to our boosting approach (AdaGAN), carefully reweighting the examples based on the previous generators, with $\beta_t = 1/t$. Both the ensemble and boosting approaches significantly outperform the vanilla approach with few additional iterations. They also outperform taking the best out of $T$ runs. The boosting outperforms all other approaches. For AdaGAN the variance of the performance is also significantly decreased.

| | $Modes: 1$ | $Modes: 2$ | $Modes: 3$ | $Modes: 5$ | $Modes: 7$ | $Modes: 10$ |
|---|---|---|---|---|---|---|
| Vanilla | 0.97 (0.9; 1.0) | 0.88 (0.4; 1.0) | 0.63 (0.5; 1.0) | 0.72 (0.5; 0.8) | 0.58 (0.4; 0.8) | 0.59 (0.2; 0.7) |
| Best of T (T=3) | 0.99 (1.0; 1.0) | 0.96 (0.9; 1.0) | 0.91 (0.7; 1.0) | 0.80 (0.7; 0.9) | 0.84 (0.7; 0.9) | 0.70 (0.6; 0.8) |
| Best of T (T=10) | 0.99 (1.0; 1.0) | 0.99 (1.0; 1.0) | 0.98 (0.8; 1.0) | 0.80 (0.8; 0.9) | 0.87 (0.8; 0.9) | 0.71 (0.7; 0.8) |
| Ensemble (T=3) | 0.99 (1.0; 1.0) | 0.98 (0.9; 1.0) | 0.93 (0.8; 1.0) | 0.78 (0.6; 1.0) | 0.85 (0.6; 1.0) | 0.80 (0.6; 1.0) |
| Ensemble (T=10) | 1.00 (1.0; 1.0) | 0.99 (1.0; 1.0) | 1.00 (1.0; 1.0) | 0.91 (0.8; 1.0) | 0.88 (0.8; 1.0) | 0.89 (0.7; 1.0) |
| TopKLast0.5 (T=3) | 0.98 (0.9; 1.0) | 0.98 (0.9; 1.0) | 0.95 (0.9; 1.0) | 0.95 (0.8; 1.0) | 0.86 (0.7; 1.0) | 0.86 (0.6; 0.9) |
| TopKLast0.5 (T=10) | 0.99 (1.0; 1.0) | 0.98 (0.9; 1.0) | 0.98 (1.0; 1.0) | 0.99 (0.8; 1.0) | 0.99 (0.8; 1.0) | 1.00 (0.8; 1.0) |
| Boosted (T=3) | 0.99 (1.0; 1.0) | 0.99 (0.9; 1.0) | 0.98 (0.9; 1.0) | 0.91 (0.8; 1.0) | 0.91 (0.8; 1.0) | 0.86 (0.7; 1.0) |
| Boosted (T=10) | 1.00 (1.0; 1.0) | 1.00 (1.0; 1.0) | 1.00 (1.0; 1.0) | 1.00 (1.0; 1.0) | 1.00 (1.0; 1.0) | 1.00 (1.0; 1.0) |

Table 1: Performance of the different algorithms on varying number of mixtures of Gaussians. The reported score is the coverage $C$, probability mass of $P_d$ covered by the $5th$ percentile of $P_g$ defined in Section 4.1.2. See Table 2 for more metrics. The reported scores are the median and interval defined by the 5% and 95% percentile (in parenthesis) (see Section 4.1.2), over 35 runs for each setting. Note that the 95% interval is not the usual confidence interval measuring the variance of the experiment itself, but rather measures the stability of the different algorithms (would remain even if each experiment was run an infinite number of times). Both the ensemble and the boosting approaches significantly outperform the vanilla GAN even with just three iterations (i.e. just two additional components). The boosting approach converges faster to the optimal coverage and with smaller variance.

coverage of the 1000 numbers. We tried to threshold the confidence of the pre-trained classifier, but decided that this metric was too ad-hoc.

For MNIST we noticed that the re-weighted distribution was often concentrating its mass on digits having very specific strokes: on different rounds it could highlight thick, thin, vertical, or diagonal digits, indicating that these traits were underrepresented in the generated samples (see Figure 3). This suggests that AdaGAN does a reasonable job at picking up different modes of the dataset, but also that there are more than 10 modes in MNIST (and more than 1000 in MNIST3). It is not clear how to evaluate the quality of generative models in this context.

We also tried to use the "inversion" metric discussed in Section 3.4.1 of [4]. For MNIST3 we noticed that a single GAN was capable of reconstructing most of the training points *very* accurately both visually and in the $\ell_2$-reconstruction sense.

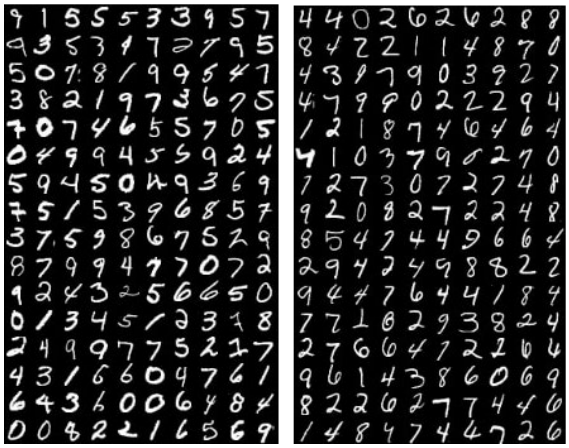

Figure 3: Digits from the MNIST dataset corresponding to the smallest (**left**) and largest (**right**) weights, obtained by the AdaGAN procedure (see Section 3) in one of the runs. Bold digits (left) are already covered and next GAN will concentrate on thin (right) digits.

# 5 Conclusion

We presented an incremental procedure for constructing an additive mixture of generative models by minimizing an $f$-divergence criterion. Based on this, we derived a boosting-style algorithm for GANs, which we call *AdaGAN*. By incrementally adding new generators into a mixture through the optimization of a GAN criterion on a reweighted data, this algorithm is able to progressively cover all the modes of the true data distribution. This addresses one of the main practical issues of training GANs.

We also presented a theoretical analysis of the convergence of this incremental procedure and showed conditions under which the mixture converges to the true distribution either exponentially or in a finite number of steps.

Our preliminary experiments (on toy data) show that this algorithm is effectively addressing the missing modes problem and allows to robustly produce a mixture which covers all modes of the data.

However, since the generative model that we obtain is not a single neural network but a mixture of such networks, the corresponding latent representation no longer has a smooth structure. This can be seen as a disadvantage compared to standard GAN where one can perform smooth interpolation in latent space. On the other hand it also allows to have a partitioned latent representation where one component is discrete. Future work will explore the possibility of leveraging this structure to model discrete aspects of the dataset, such as the class in object recognition datasets in a similar spirit to [22].

# References


[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[2] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. In *ICLR*, 2014.

[3] Martin Arjovsky, Soumith Chintala, and Lon Bottou. Wasserstein GAN. *arXiv:1701.07875*, 2017.

[4] L. Metz, B. Poole, D. Pfau, and J. Sohl-Dickstein. Unrolled generative adversarial networks. *arXiv:1611.02163*, 2017.

[5] Tong Che, Yanran Li, Athul Paul Jacob, Yoshua Bengio, and Wenjie Li. Mode regularized generative adversarial networks. *arXiv:1612.02136*, 2016.

[6] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[7] Sebastian Nowozin, Botond Cseke, and Ryota Tomioka. f-GAN: Training generative neural samplers using variational divergence minimization. In *Advances in Neural Information Processing Systems*, 2016.

[8] Max Welling, Richard S. Zemel, and Geoffrey E. Hinton. Self supervised boosting. In *Advances in neural information processing systems*, pages 665–672, 2002.

[9] Zhuowen Tu. Learning generative models via discriminative approaches. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.

[10] Aditya Grover and Stefano Ermon. Boosted generative models. ICLR 2017 conference submission, 2016.

[11] R. M. Neal. Annealed importance sampling. *Statistics and Computing*, 11(2):125–139, 2001.

[12] Saharon Rosset and Eran Segal. Boosting density estimation. In *Advances in Neural Information Processing Systems*, pages 641–648, 2002.

[13] A Barron and J Li. Mixture density estimation. *Biometrics*, 53:603–618, 1997.

[14] Yaxing Wang, Lichao Zhang, and Joost van de Weijer. Ensembles of generative adversarial networks. *arXiv:1612.00991*, 2016.

[15] F. Liese and K.-J. Miescke. *Statistical Decision Theory*. Springer, 2008.

[16] M. D. Reid and R. C. Williamson. Information, divergence and risk for binary experiments. *Journal of Machine Learning Research*, 12:731–817, 2011.

[17] Bent Fuglede and Flemming Topsoe. Jensen-shannon divergence and hilbert space embedding. In *IEEE International Symposium on Information Theory*, pages 31–31, 2004.

[18] Matthias Hein and Olivier Bousquet. Hilbertian metrics and positive definite kernels on probability measures. In *AISTATS*, pages 136–143, 2005.

[19] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, B*, 39:1–38, 1977.

[20] Yuhuai Wu, Yuri Burda, Ruslan Salakhutdinov, and Roger Grosse. On the quantitative analysis of decoder-based generative models, 2016.

[21] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *ICLR*, 2016.

[22] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2172–2180, 2016.

# A Further details on toy experiments

To illustrate the 'meta-algorithm aspect' of AdaGAN, we also performed experiments with an unrolled GAN [4] instead of a GAN as the base generator. We trained the GANs both with the Jensen-Shannon objective (2), and with its modified version proposed in [1] (and often considered as the baseline GAN), where $\log(1 - D(G(Z)))$ is replaced by $-\log(D(G(Z)))$. We use the same network architecture as in the other toy experiments. Figure 4 illustrates our results. We find that AdaGAN works with all underlying GAN algorithms. Note that, where the usual GAN updates the generator and the discriminator once, an unrolled GAN with 5 unrolling steps updates the generator once and the discriminator $1 + 5$, i.e. 6 times (and then rolls back 5 steps). Thus, in terms of computation time, training 1 single unrolled GAN roughly corresponds to doing 3 steps of AdaGAN with a usual GAN. In that sense, Figure 4 shows that AdaGAN (with a usual GAN) significantly outperforms a single unrolled GAN. Additionally, we note that using the Jensen-Shannon objective (rather than the modified version) seems to have some mode-regularizing effect. Surprisingly, using unrolling steps makes no significant difference.

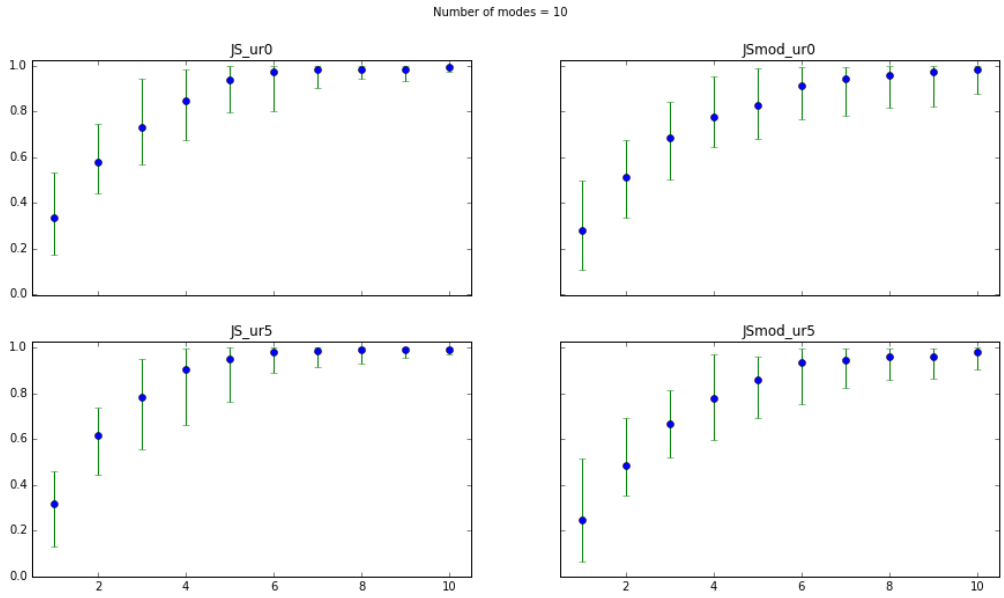


Figure 4: Comparison of AdaGAN ran with a GAN (top row) and with an unrolled GAN [4] (bottom). Coverage $C$ of the true data by the model distribution $P^T_{model}$, as a function of iterations $T$. Experiments are similar to those of Figure 3, but with 10 modes. Top and bottom rows correspond to the usual and the unrolled GAN (with 5 unrolling steps) respectively, trained with the Jensen-Shannon objective (2) on the left, and with the modified objective originally proposed by [1] on the right. In terms of computation time, one step of AdaGAN with unrolled GAN corresponds to roughly 3 steps of AdaGAN with a usual GAN. On all images $T = 1$ corresponds to vanilla unrolled GAN.

# B Further details on MNIST/MNIST3 experiments

**GAN Architecture** We ran AdaGAN on MNIST (28x28 pixel images) using (de)convolutional networks with batch normalizations and leaky ReLu. The latent space has dimension 100. We used the following

architectures:

Generator: 100 x 1 x 1 → fully connected → 7 x 7 x 16 → deconv → 14 x 14 x 8 →
→ deconv → 28 x 28 x 4 → deconv → 28 x 28 x 1

Discriminator: 28 x 28 x 1 → conv → 14 x 14 x 16 → conv → 7 x 7 x 32 →
→ fully connected → 1

where each arrow consists of a leaky ReLu (with 0.3 leak) followed by a batch normalization, conv and deconv are convolutions and transposed convolutions with 5x5 filters, and fully connected are linear layers with bias. The distribution over $\mathcal{Z}$ is uniform over the unit box. We use the Adam optimizer with $\beta_1 = 0.5$, with 2 G steps for 1 D step and learning rates 0.005 for G, 0.001 for D, and 0.0001 for the classifier C that does the reweighting of digits. We optimized D and G over 200 epochs and C over 5 epochs, using the original Jensen-Shannon objective (2), without the log trick, with no unrolling and with minibatches of size 128.

**Empirical observations** Although we could not find any appropriate metric to measure the increase of diversity promoted by AdaGAN, we observed that the re-weighting scheme indeed focuses on digits with very specific strokes. In Figure 5 for example, we see that after one AdaGAN step, the generator produces overly thick digits (top left image). Thus AdaGAN puts small weights on the thick digits of the dataset (bottom left) and high weights on the thin ones (bottom right). After the next step, the new GAN produces both thick and thin digits.

# C Proofs

## C.1 Proof of Theorem 1

Before proving Theorem 1, we introduce two lemmas. The first one is about the determination of the constant $\lambda$, the second one is about comparing the divergences of mixtures.

**Lemma 5** *Let $P$ and $Q$ be two distributions, $\gamma \in [0,1]$ and $\lambda \in \mathcal{R}$. The function*

$$g(\lambda) := \int \left(\lambda - \gamma \frac{dQ}{dP}\right)_+ dP$$

*is nonnegative, convex, nondecreasing, satisfies $g(\lambda) \leq \lambda$, and its right derivative is given by*

$$g'_+(\lambda) = P\left(\lambda \cdot dP \geq \gamma \cdot dQ\right).$$

*The equation*

$$g(\lambda) = 1 - \gamma$$

*has a solution $\lambda^*$ (unique when $\gamma < 1$) with $\lambda^* \in [1-\gamma, 1]$. Finally, if $P(dQ = 0) \geq \delta$ for a strictly positive constant $\delta$ then $\lambda^* \leq (1-\gamma)\delta^{-1}$.*

**Proof** The convexity of $g$ follows immediately from the convexity of $x \mapsto (x)_+$ and the linearity of the integral. Similarly, since $x \mapsto (x)_+$ is non-decreasing, $g$ is non-decreasing.

We define the set $\mathcal{I}(\lambda)$ as follows:

$$\mathcal{I}(\lambda) := \{x \in \mathcal{X} : \lambda \cdot dP(x) \geq \gamma \cdot dQ(x)\}.$$

Now let us consider $g(\lambda + \epsilon) - g(\lambda)$ for some small $\epsilon > 0$. This can also be written:

$$\begin{aligned} g(\lambda+\epsilon) - g(\lambda) &= \int_{\mathcal{I}(\lambda)} \epsilon dP + \int_{\mathcal{I}(\lambda+\epsilon)\setminus\mathcal{I}(\lambda)} (\lambda+\epsilon) dP - \int_{\mathcal{I}(\lambda+\epsilon)\setminus\mathcal{I}(\lambda)} \gamma dQ \\ &= \epsilon P(\mathcal{I}(\lambda)) + \int_{\mathcal{I}(\lambda+\epsilon)\setminus\mathcal{I}(\lambda)} (\lambda+\epsilon) dP - \int_{\mathcal{I}(\lambda+\epsilon)\setminus\mathcal{I}(\lambda)} \gamma dQ. \end{aligned}$$

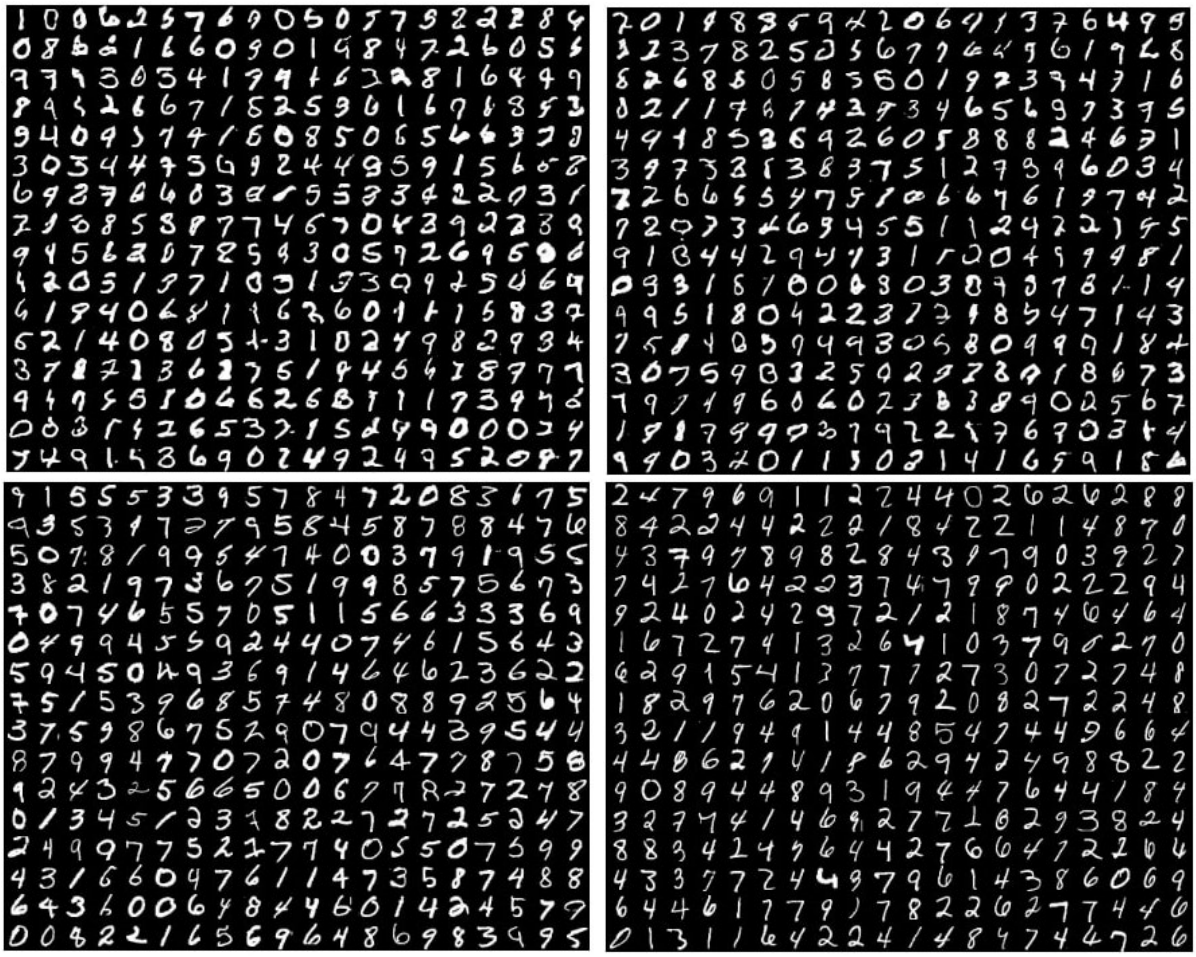

Figure 5: AdaGAN on MNIST. Bottom row are true MNIST digits with smallest (left) and highest (right) weights after re-weighting at the end of the first AdaGAN step. Those with small weight are thick and resemble those generated by the GAN after the first AdaGAN step (top left). After training with the re-weighted dataset during the second iteration of AdaGAN, the new mixture produces more thin digits (top right).

On the set $\mathcal{I}(\lambda+\epsilon)\backslash\mathcal{I}(\lambda)$, we have

$$(\lambda+\epsilon)dP - \gamma dQ \in [0,\epsilon].$$

So that

$$\epsilon P(\mathcal{I}(\gamma)) \le g(\lambda+\epsilon) - g(\lambda) \le \epsilon P(\mathcal{I}(\gamma)) + \epsilon P\big(\mathcal{I}(\lambda+\epsilon)\backslash\mathcal{I}(\lambda)\big) = \epsilon P(\mathcal{I}(\lambda+\epsilon))$$

and thus

$$\lim_{\epsilon\to 0^+} \frac{g(\lambda+\epsilon)-g(\lambda)}{\epsilon} = \lim_{\epsilon\to 0^+} P(\mathcal{I}(\lambda+\epsilon)) = P(\mathcal{I}(\lambda)).$$

This gives the expression of the right derivative of $g$. Moreover, notice that for $\lambda,\gamma>0$

$$g'_+(\lambda) = P\left(\lambda\cdot dP \ge \gamma\cdot dQ\right) = P\left(\frac{dQ}{dP} \le \frac{\lambda}{\gamma}\right) = 1 - P\left(\frac{dQ}{dP} > \frac{\lambda}{\gamma}\right) \ge 1-\gamma/\lambda$$

by Markov's inequality.

It is obvious that $g(0) = 0$. By Jensen's inequality applied to the convex function $x \mapsto (x)_+$, we have $g(\lambda) \geq (\lambda - \gamma)_+$. So $g(1) \geq 1 - \gamma$. Also, $g = 0$ on $\mathcal{R}^-$ and $g \leq \lambda$. This means $g$ is continuous on $\mathcal{R}$ and thus reaches the value $1 - \gamma$ on the interval $(0, 1]$ which shows the existence of $\lambda^* \in (0, 1]$. To show that $\lambda^*$ is unique we notice that since $g(x) = 0$ on $\mathcal{R}^-$, $g$ is convex and non-decreasing, $g$ cannot be constant on an interval not containing 0, and thus $g(x) = 1 - \gamma$ has a unique solution for $\gamma < 1$.

Also by convexity of $g$,

$$g(0) - g(\lambda^*) \geq -\lambda^* g'_+(\lambda^*),$$

which gives $\lambda^* \geq (1-\gamma)/g'_+(\lambda^*) \geq 1-\gamma$ since $g'_+ \leq 1$. If $P(dQ = 0) \geq \delta > 0$ then also $g'_+(0) \geq \delta > 0$. Using the fact that $g'_+$ is increasing we conclude that $\lambda^* \leq (1 - \gamma)\delta^{-1}$. ∎

Next we introduce some simple convenience lemma for comparing convex functions of random variables.

**Lemma 6** *Let $f$ be a convex function, $X, Y$ be real-valued random variables and $c \in \mathcal{R}$ be a constant such that*

$$\mathbb{E}\left[\max(c, Y)\right] = \mathbb{E}\left[X + Y\right].$$

*Then we have the following bound:*

$$\mathbb{E}\left[f(\max(c, Y))\right] \leq \mathbb{E}\left[f(X + Y)\right] - \mathbb{E}\left[X(f'(Y) - f'(c))_+\right] \leq \mathbb{E}\left[f(X + Y)\right]. \tag{19}$$

*If in addition, $Y \leq M$ a.s. for $M \geq c$, then*

$$\mathbb{E}\left[f(\max(c, Y))\right] \leq f(c) + \frac{f(M) - f(c)}{M - c}(\mathbb{E}\left[X + Y\right] - c). \tag{20}$$

**Proof** We decompose the expectation with respect to the value of the max, and use the convexity of $f$:

$$\begin{aligned}
f(X + Y) - f(\max(c, Y)) &= 1_{[Y \leq c]}(f(X + Y) - f(c)) + 1_{[Y > c]}(f(X + Y) - f(Y)) \\
&\geq 1_{[Y \leq c]} f'(c)(X + Y - c) + 1_{[Y > c]} X f'(Y) \\
&= (1 - 1_{[Y > c]}) X f'(c) + f'(c)(Y - \max(c, Y)) + 1_{[Y > c]} X f'(Y) \\
&= f'(c)(X + Y - \max(c, Y)) + 1_{[Y > c]} X (f'(Y) - f'(c)) \\
&= f'(c)(X + Y - \max(c, Y)) + X(f'(Y) - f'(c))_+,
\end{aligned}$$

where we used that $f'$ is non-decreasing in the last step. Taking the expectation gives the first inequality.

For the second inequality, we use the convexity of $f$ on the interval $[c, M]$:

$$f(\max(c, Y)) \leq f(c) + \frac{f(M) - f(c)}{M - c}(\max(c, Y) - c).$$

Taking an expectation on both sides gives the second inequality. ∎

**Proof** [Theorem 1] We first apply Lemma 5 with $\gamma = 1 - \beta$ and this proves the existence of $\lambda^*$ in the interval $(\beta, 1]$, which shows that $Q^*_\beta$ is indeed well-defined as a distribution.

Then we use Inequality (19) of Lemma 6 with $X = \beta dQ/dP_d$, $Y = (1 - \beta)dP_g/dP_d$, and $c = \lambda^*$. We easily verify that $X + Y = ((1 - \beta)dP_g + \beta dQ)/dP_d$ and $\max(c, Y) = ((1 - \beta)dP_g + \beta dQ^*_\beta)/dP_d$ and both have expectation 1 with respect to $P_d$. We thus obtain for any distribution $Q$,

$$D_f((1 - \beta)P_g + \beta Q^*_\beta \,\|\, P_d) \leq D_f((1 - \beta)P_g + \beta Q \,\|\, P_d).$$

This proves the optimality of $Q^*_\beta$. ∎

## C.2 Proof of Theorem 2

**Lemma 7** *Let $P$ and $Q$ be two distributions, $\gamma \in (0,1)$, and $\lambda \geq 0$. The function*

$$h(\lambda) := \int \left(\frac{1}{\gamma} - \lambda \frac{dQ}{dP}\right)_+ dP$$

*is convex, non-increasing, and its right derivative is given by $h'_+(\lambda) = -Q(1/\gamma \geq \lambda dQ(X)/dP(X))$. Denote $\Delta := P(dQ(X)/dP(X) = 0)$. Then the equation*

$$h(\lambda) = \frac{1-\gamma}{\gamma}$$

*has no solutions if $\Delta > 1-\gamma$, has a single solution $\lambda^\dagger \geq 1$ if $\Delta < 1-\gamma$, and has infinitely many or no solutions when $\Delta = 1-\gamma$.*

**Proof** The convexity of $h$ follows immediately from the convexity of $x \mapsto (a-x)_+$ and the linearity of the integral. Similarly, since $x \mapsto (a-x)_+$ is non-increasing, $h$ is non-increasing as well.

We define the set $\mathcal{J}(\lambda)$ as follows:

$$\mathcal{J}(\lambda) := \left\{ x \in \mathcal{X} \colon \frac{1}{\gamma} \geq \lambda \frac{dQ}{dP}(x) \right\}.$$

Now let us consider $h(\lambda) - h(\lambda+\epsilon)$ for any $\epsilon > 0$. Note that $\mathcal{J}(\lambda+\epsilon) \subseteq \mathcal{J}(\lambda)$. We can write:

$$\begin{aligned}
h(\lambda) - h(\lambda+\epsilon) &= \int_{\mathcal{J}(\lambda)} \left(\frac{1}{\gamma} - \lambda \frac{dQ}{dP}\right) dP - \int_{\mathcal{J}(\lambda+\epsilon)} \left(\frac{1}{\gamma} - (\lambda+\epsilon)\frac{dQ}{dP}\right) dP \\
&= \int_{\mathcal{J}(\lambda)\setminus\mathcal{J}(\lambda+\epsilon)} \left(\frac{1}{\gamma} - \lambda \frac{dQ}{dP}\right) dP + \int_{\mathcal{J}(\lambda+\epsilon)} \left(\epsilon \frac{dQ}{dP}\right) dP \\
&= \int_{\mathcal{J}(\lambda)\setminus\mathcal{J}(\lambda+\epsilon)} \left(\frac{1}{\gamma} - \lambda \frac{dQ}{dP}\right) dP + \epsilon \cdot Q(\mathcal{J}(\lambda+\epsilon)).
\end{aligned}$$

Note that for $x \in \mathcal{J}(\lambda) \setminus \mathcal{J}(\lambda+\epsilon)$ we have

$$0 \leq \frac{1}{\gamma} - \lambda \frac{dQ}{dP}(x) < \epsilon \frac{dQ}{dP}(x).$$

This gives the following:

$$\epsilon \cdot Q(\mathcal{J}(\lambda+\epsilon)) \leq h(\lambda) - h(\lambda+\epsilon) \leq \epsilon \cdot Q(\mathcal{J}(\lambda+\epsilon)) + \epsilon \cdot Q(\mathcal{J}(\lambda) \setminus \mathcal{J}(\lambda+\epsilon)) = \epsilon \cdot Q(\mathcal{J}(\lambda)),$$

which shows that $h$ is continuous. Also

$$\lim_{\epsilon \to 0^+} \frac{h(\lambda+\epsilon) - h(\lambda)}{\epsilon} = \lim_{\epsilon \to 0^+} -Q(\mathcal{J}(\lambda+\epsilon)) = -Q(\mathcal{J}(\lambda)).$$

It is obvious that $h(0) = 1/\gamma$ and $h \leq \gamma^{-1}$ for $\lambda \geq 0$. By Jensen's inequality applied to the convex function $x \mapsto (a-x)_+$, we have $h(\lambda) \geq \left(\gamma^{-1} - \lambda\right)_+$. So $h(1) \geq \gamma^{-1} - 1$. We conclude that $h$ may reach the value $(1-\gamma)/\gamma = \gamma^{-1} - 1$ only on $[1, +\infty)$. Note that

$$h(\lambda) \to \frac{1}{\gamma} P\left(\frac{dQ}{dP}(X) = 0\right) = \frac{\Delta}{\gamma} \geq 0 \quad \text{as} \quad \lambda \to \infty.$$

Thus if $\Delta/\gamma > \gamma^{-1} - 1$ the equation $h(\lambda) = \gamma^{-1} - 1$ has no solutions, as $h$ is non-increasing. If $\Delta/\gamma = \gamma^{-1} - 1$ then either $h(\lambda) > \gamma^{-1} - 1$ for all $\lambda \geq 0$ and we have no solutions or there is a finite $\lambda' \geq 1$ such that

$h(\lambda') = \gamma^{-1} - 1$, which means that the equation is also satisfied by all $\lambda \geq \lambda'$, as $h$ is continuous and non-increasing. Finally, if $\Delta/\gamma < \gamma^{-1} - 1$ then there is a unique $\lambda^\dagger$ such that $h(\lambda^\dagger) = \gamma^{-1} - 1$, which follows from the convexity of $h$. ∎

Next we introduce some simple convenience lemma for comparing convex functions of random variables.

**Lemma 8** *Let $f$ be a convex function, $X, Y$ be real-valued random variables such that $X \leq Y$ a.s., and $c \in \mathcal{R}$ be a constant such that*[9]

$$\mathbb{E}\left[\min(c, Y)\right] = \mathbb{E}\left[X\right].$$

*Then we have the following lower bound:*

$$\mathbb{E}\left[f(X) - f(\min(c, Y))\right] \geq 0.$$

**Proof** We decompose the expectation with respect to the value of the min, and use the convexity of $f$:

$$\begin{aligned} f(X) - f(\min(c, Y)) &= \mathsf{1}_{[Y \leq c]}(f(X) - f(Y)) + \mathsf{1}_{[Y > c]}(f(X) - f(c)) \\ &\geq \mathsf{1}_{[Y \leq c]} f'(Y)(X - Y) + \mathsf{1}_{[Y > c]}(X - c) f'(c) \\ &\geq \mathsf{1}_{[Y \leq c]} f'(c)(X - Y) + \mathsf{1}_{[Y > c]}(X - c) f'(c) \\ &= X f'(c) - \min(Y, c) f'(c), \end{aligned}$$

where we used the fact that $f'$ is non-decreasing in the previous to last step. Taking the expectation we get the result. ∎

**Lemma 9** *Let $P_g, P_d$ be two fixed distributions and $\beta \in (0, 1)$. Assume*

$$P_d\left(\frac{dP_g}{dP_d} = 0\right) < \beta.$$

*Let $\mathcal{M}(P_d, \beta)$ be the set of all probability distributions $T$ such that $(1 - \beta)dT \leq dP_d$. Then the following minimization problem:*

$$\min_{T \in \mathcal{M}(P_d, \beta)} D_f(T \,\|\, P_g)$$

*has the solution $T^*$ with density*

$$dT^* := \min(dP_d/(1 - \beta), \lambda^\dagger dP_g),$$

*where $\lambda^\dagger$ is the unique value in $[1, \infty)$ such that $\int dT^* = 1$.*

**Proof** We will use Lemma 8 with $X = dT(Z)/dP_g(Z)$, $Y = dP_d(Z)/\big((1 - \beta)dP_g(Z)\big)$, and $c = \lambda^*$, $Z \sim P_g$. We need to verify that assumptions of Lemma 8 are satisfied. Obviously, $Y \geq X$. We need to show that there is a constant $c$ such that

$$\int \min\left(c, \frac{dP_d}{(1 - \beta)dP_g}\right) dP_g = 1.$$

Rewriting this equation we get the following equivalent one:

$$\beta = \int \left(dP_d - \min\left(c(1 - \beta)P_g, dP_d\right)\right) = (1 - \beta) \int \left(\frac{1}{1 - \beta} - c\frac{dP_g}{dP_d}\right)_+ dP_d. \tag{21}$$

Using the fact that

$$P_d\left(\frac{dP_g}{dP_d} = 0\right) < \beta$$

[9] Generally it is not guaranteed that such a constant $c$ always exists. In this result we assume this is the case.

we may apply Lemma 7 and conclude that there is a unique $c \in [1, \infty)$ satisfying (21), which we denote $\lambda^\dagger$. ∎

To conclude the proof of Theorem 2, observe that from Lemma 9, by making the change of variable $T = (P_d - \beta Q)/(1-\beta)$ we can rewrite the minimization problem as follows:

$$\min_{Q:\, \beta dQ \le dP_d} D_{f^\circ}\left(P_g \,\|\, \frac{P_d - \beta Q}{1-\beta}\right)$$

and we verify that the solution has the form $dQ^\dagger_\beta = \frac{1}{\beta}\big(dP_d - \lambda^\dagger(1-\beta)dP_g\big)_+$. Since this solution does not depend on $f$, the fact that we optimized $D_{f^\circ}$ is irrelevant and we get the same solution for $D_f$.

# D $f$-Divergences

**Jensen-Shannon** This divergence corresponds to

$$D_f(P\|Q) = \mathrm{JS}(P,Q) = \int_{\mathcal{X}} f\left(\frac{dP}{dQ}(x)\right) dQ(x)$$

with

$$f(u) = -(u+1)\log\frac{u+1}{2} + u\log u.$$

Indeed,

$$\begin{aligned}
JS(P,Q) &:= \int_{\mathcal{X}} q(x)\left(-\left(\frac{p(x)}{q(x)}+1\right)\log\left(\frac{\frac{p(x)}{q(x)}+1}{2}\right) + \frac{p(x)}{q(x)}\log\frac{p(x)}{q(z)}\right)dx \\
&= \int_{\mathcal{X}} q(x)\left(\frac{p(x)}{q(x)}\log\frac{2q(x)}{p(x)+q(x)} + \log\frac{2q(x)}{p(x)+q(x)} + \frac{p(x)}{q(z)}\log\frac{p(x)}{q(z)}\right)dx \\
&= \int_{\mathcal{X}} p(x)\log\frac{2q(x)}{p(x)+q(x)} + q(x)\log\frac{2q(x)}{p(x)+q(x)} + p(x)\log\frac{p(x)}{q(z)}dx \\
&= \mathrm{KL}\left(Q, \frac{P+Q}{2}\right) + \mathrm{KL}\left(P, \frac{P+Q}{2}\right).
\end{aligned}$$

# E Additional experimental results

At each iteration of the boosting approach, different reweighting heuristics are possible. This section contains more complete results about the following three heuristics:

- Constant $\beta$, and using the proposed reweighting scheme given $\beta$. See Table 3.
- Reweighting similar to "Cascade GAN" from [14], i.e. keep the top $x$% of examples, based on the discriminator corresponding to the *previous* generator. See Table 4.
- Keep the top $x$% of examples, based on the discriminator corresponding to the mixture of *all previous* generators. See Table 5.

Note that when properly tuned, each reweighting scheme outperforms the baselines, and have similar performances when used with few iterations. However, they require an additional parameter to tune, and are worse than the simple $\beta = 1/t$ heuristic proposed above.

| | $Modes: 1$ | $Modes: 2$ | $Modes: 3$ | $Modes: 5$ | $Modes: 7$ | $Modes: 10$ |
|---|---|---|---|---|---|---|
| Vanilla | 0.97 (0.9; 1.0) | 0.88 (0.4; 1.0) | 0.63 (0.5; 1.0) | 0.72 (0.5; 0.8) | 0.58 (0.4; 0.8) | 0.59 (0.2; 0.7) |
| Best of T (T=3) | 0.99 (1.0; 1.0) | 0.96 (0.9; 1.0) | 0.91 (0.7; 1.0) | 0.80 (0.7; 0.9) | 0.84 (0.7; 0.9) | 0.70 (0.6; 0.8) |
| Best of T (T=10) | 0.99 (1.0; 1.0) | 0.99 (1.0; 1.0) | 0.98 (0.8; 1.0) | 0.80 (0.8; 0.9) | 0.87 (0.8; 0.9) | 0.71 (0.7; 0.8) |
| Ensemble (T=3) | 0.99 (1.0; 1.0) | 0.98 (0.9; 1.0) | 0.93 (0.8; 1.0) | 0.78 (0.6; 1.0) | 0.85 (0.6; 1.0) | 0.80 (0.6; 1.0) |
| Ensemble (T=10) | 1.00 (1.0; 1.0) | 0.99 (1.0; 1.0) | 1.00 (1.0; 1.0) | 0.91 (0.8; 1.0) | 0.88 (0.8; 1.0) | 0.89 (0.7; 1.0) |
| Boosted (T=3) | 0.99 (1.0; 1.0) | 0.99 (0.9; 1.0) | 0.98 (0.9; 1.0) | 0.91 (0.8; 1.0) | 0.91 (0.8; 1.0) | 0.86 (0.7; 1.0) |
| Boosted (T=10) | 1.00 (1.0; 1.0) | 1.00 (1.0; 1.0) | 1.00 (1.0; 1.0) | 1.00 (1.0; 1.0) | 1.00 (1.0; 1.0) | 1.00 (1.0; 1.0) |

| | $Modes: 1$ | $Modes: 2$ | $Modes: 3$ | $Modes: 5$ | $Modes: 7$ | $Modes: 10$ |
|---|---|---|---|---|---|---|
| Vanilla | −4.49 (−5.4; −4.4) | −6.02 (−86.8; −5.3) | −16.03 (−59.6; −5.5) | −23.65 (−118.8; −5.7) | −126.87 (−250.4; −12.8) | −55.51 (−185.2; −11.2) |
| Best of T (T=3) | −4.39 (−4.6; −4.3) | −5.40 (−24.3; −5.2) | −5.57 (−23.5; −5.4) | −9.91 (−35.8; −5.1) | −36.94 (−90.0; −9.7) | −19.12 (−59.2; −9.7) |
| Best of T (T=10) | −4.34 (−4.4; −4.3) | −5.24 (−5.4; −5.2) | −5.45 (−5.6; −5.3) | −5.49 (−9.4; −5.0) | −9.72 (−17.3; −6.5) | −9.12 (−16.8; −6.6) |
| Ensemble (T=3) | −4.46 (−4.8; −4.4) | −5.59 (−6.6; −5.2) | −4.78 (−5.5; −4.6) | −14.71 (−51.9; −5.4) | −6.70 (−28.7; −5.5) | −8.59 (−25.4; −6.1) |
| Ensemble (T=10) | −4.52 (−4.7; −4.4) | −5.49 (−6.6; −5.2) | −4.98 (−6.5; −4.6) | −5.44 (−6.0; −5.2) | −5.82 (−6.4; −5.5) | −6.08 (−6.3; −5.7) |
| Boosted (T=3) | −4.50 (−4.8; −4.4) | −5.32 (−5.8; −5.2) | −4.80 (−5.8; −4.6) | −5.39 (−19.3; −5.1) | −5.56 (−12.4; −5.2) | −8.03 (−28.7; −6.1) |
| Boosted (T=10) | −4.55 (−4.6; −4.4) | −5.30 (−5.5; −5.2) | −5.07 (−5.6; −4.7) | −5.25 (−5.5; −4.6) | −5.03 (−5.5; −4.8) | −5.92 (−6.2; −5.6) |

Table 2: Performance of the different algorithms on varying number of mixtures of Gaussians. The reported scores are the median and interval defined by the 5% and 95% percentile (in parenthesis) (see Section 4.1.2), over 35 runs for each setting. The top table reports the coverage $C$, probability mass of $P_d$ covered by the $5th$ percentile of $P_g$ defined in Section 4.1.2. The bottom table reports the log likelihood of the true data under the model $P_g$. Note that the 95% interval is not the usual confidence interval measuring the variance of the experiment itself, but rather measures the stability of the different algorithms (would remain even if each experiment was run an infinite number of times). Both the ensemble and the boosting approaches significantly outperform the vanilla GAN even with just three iterations (i.e. just two additional components). The boosting approach converges faster to the optimal coverage.

| | $Modes: 1$ | $Modes: 2$ | $Modes: 3$ | $Modes: 5$ | $Modes: 7$ | $Modes: 10$ |
|---|---|---|---|---|---|---|
| Vanilla | 0.98 (0.9; 1.0) | 0.86 (0.5; 1.0) | 0.66 (0.5; 1.0) | 0.61 (0.5; 0.8) | 0.55 (0.4; 0.7) | 0.58 (0.3; 0.8) |
| Boosted (T=3) | 0.99 (1.0; 1.0) | 0.98 (0.9; 1.0) | 0.98 (0.9; 1.0) | 0.93 (0.8; 1.0) | 0.97 (0.8; 1.0) | 0.87 (0.6; 1.0) |
| Boosted (T=10) | 1.00 (1.0; 1.0) | 0.99 (1.0; 1.0) | 1.00 (1.0; 1.0) | 0.99 (0.9; 1.0) | 0.99 (0.8; 1.0) | 0.97 (0.8; 1.0) |
| Beta0.2 (T=3) | 0.99 (1.0; 1.0) | 0.97 (0.9; 1.0) | 0.97 (0.9; 1.0) | 0.95 (0.8; 1.0) | 0.96 (0.7; 1.0) | 0.88 (0.7; 1.0) |
| Beta0.2 (T=10) | 0.99 (1.0; 1.0) | 0.99 (1.0; 1.0) | 0.99 (1.0; 1.0) | 1.00 (1.0; 1.0) | 1.00 (0.9; 1.0) | 1.00 (0.9; 1.0) |
| Beta0.3 (T=3) | 0.99 (1.0; 1.0) | 0.98 (0.9; 1.0) | 0.98 (0.9; 1.0) | 0.96 (0.8; 1.0) | 0.96 (0.6; 1.0) | 0.88 (0.7; 1.0) |
| Beta0.3 (T=10) | 1.00 (1.0; 1.0) | 0.99 (1.0; 1.0) | 0.99 (1.0; 1.0) | 1.00 (1.0; 1.0) | 1.00 (0.9; 1.0) | 0.99 (0.9; 1.0) |
| Beta0.4 (T=3) | 0.99 (1.0; 1.0) | 0.98 (0.9; 1.0) | 0.95 (0.9; 1.0) | 0.94 (0.8; 1.0) | 0.89 (0.7; 1.0) | 0.89 (0.7; 1.0) |
| Beta0.4 (T=10) | 0.99 (1.0; 1.0) | 0.99 (0.9; 1.0) | 0.96 (0.9; 1.0) | 0.97 (0.8; 1.0) | 0.99 (0.8; 1.0) | 0.90 (0.8; 1.0) |
| Beta0.5 (T=3) | 0.99 (1.0; 1.0) | 0.98 (0.9; 1.0) | 0.97 (0.8; 1.0) | 0.82 (0.8; 1.0) | 0.86 (0.7; 1.0) | 0.81 (0.6; 1.0) |
| Beta0.5 (T=10) | 0.99 (1.0; 1.0) | 0.98 (0.9; 1.0) | 0.97 (0.9; 1.0) | 0.84 (0.8; 1.0) | 0.87 (0.7; 1.0) | 0.91 (0.8; 1.0) |

| | $Modes: 1$ | $Modes: 2$ | $Modes: 3$ | $Modes: 5$ | $Modes: 7$ | $Modes: 10$ |
|---|---|---|---|---|---|---|
| Vanilla | −4.50 (−5.0; −4.4) | −5.65 (−72.7; −5.1) | −19.63 (−62.1; −5.6) | −28.16 (−293.1; −16.3) | −56.94 (−248.1; −14.3) | −71.11 (−184.8; −12.5) |
| Boosted (T=3) | −4.56 (−4.9; −4.4) | −5.55 (−5.9; −5.2) | −5.01 (−6.7; −4.7) | −5.49 (−18.7; −4.9) | −5.60 (−14.5; −5.0) | −6.86 (−47.3; −5.6) |
| Boosted (T=10) | −4.56 (−4.7; −4.5) | −5.46 (−5.6; −5.3) | −5.08 (−5.8; −4.7) | −5.04 (−5.5; −4.6) | −5.51 (−5.9; −5.1) | −5.51 (−6.0; −5.2) |
| Beta0.2 (T=3) | −4.52 (−4.8; −4.4) | −5.31 (−5.6; −5.1) | −4.85 (−6.3; −4.6) | −5.33 (−14.4; −4.8) | −5.68 (−26.2; −5.2) | −6.13 (−32.7; −5.7) |
| Beta0.2 (T=10) | −4.58 (−4.8; −4.5) | −5.30 (−5.5; −5.2) | −4.94 (−6.6; −4.6) | −5.23 (−5.5; −4.7) | −5.60 (−6.0; −5.3) | −5.98 (−6.1; −5.7) |
| Beta0.3 (T=3) | −4.60 (−4.9; −4.4) | −5.34 (−5.7; −5.2) | −5.41 (−5.7; −5.1) | −5.33 (−12.9; −4.9) | −5.68 (−11.0; −5.4) | −6.41 (−29.2; −5.6) |
| Beta0.3 (T=10) | −4.57 (−4.8; −4.4) | −5.37 (−5.5; −5.2) | −5.27 (−5.6; −5.0) | −5.26 (−5.6; −5.0) | −5.71 (−6.0; −5.3) | −5.82 (−6.1; −5.4) |
| Beta0.4 (T=3) | −4.62 (−4.9; −4.4) | −5.36 (−5.6; −5.1) | −4.74 (−5.3; −4.6) | −5.34 (−26.2; −4.9) | −5.77 (−37.3; −5.1) | −12.37 (−75.9; −5.9) |
| Beta0.4 (T=10) | −4.49 (−4.7; −4.4) | −5.40 (−5.7; −5.3) | −5.08 (−6.9; −4.7) | −5.49 (−5.9; −5.2) | −5.43 (−6.0; −5.1) | −5.68 (−6.2; −5.2) |
| Beta0.5 (T=3) | −4.60 (−4.9; −4.4) | −5.40 (−5.7; −5.3) | −4.77 (−5.4; −4.6) | −5.63 (−24.5; −5.2) | −6.05 (−17.9; −5.5) | −8.29 (−23.1; −6.1) |
| Beta0.5 (T=10) | −4.62 (−4.8; −4.4) | −5.43 (−5.7; −5.2) | −5.12 (−6.6; −4.7) | −5.48 (−8.4; −5.1) | −5.85 (−6.1; −5.3) | −6.31 (−7.7; −6.0) |

Table 3: Performance with constant $\beta$, exploring a range of possible values. The reported scores are the median and interval defined by the 5% and 95% percentile (in parenthesis) (see Section 4.1.2), over 35 runs for each setting. The top table reports the coverage $C$, probability mass of $P_d$ covered by the $5th$ percentile of $P_g$ defined in Section 4.1.2. The bottom table reports the log likelihood of the true data under $P_g$.

| | $Modes: 1$ | $Modes: 2$ | $Modes: 3$ | $Modes: 5$ | $Modes: 7$ | $Modes: 10$ |
|---|---|---|---|---|---|---|
| Vanilla | 0.96 (0.9; 1.0) | 0.90 (0.5; 1.0) | 0.65 (0.5; 1.0) | 0.61 (0.5; 0.8) | 0.69 (0.3; 0.8) | 0.59 (0.3; 0.7) |
| Boosted (T=3) | 0.99 (1.0; 1.0) | 0.98 (0.9; 1.0) | 0.98 (0.9; 1.0) | 0.93 (0.8; 1.0) | 0.97 (0.8; 1.0) | 0.87 (0.6; 1.0) |
| Boosted (T=10) | 1.00 (1.0; 1.0) | 0.99 (1.0; 1.0) | 1.00 (1.0; 1.0) | 0.99 (0.9; 1.0) | 0.99 (0.8; 1.0) | 0.97 (0.8; 1.0) |
| TopKLast0.1 (T=3) | 0.98 (0.9; 1.0) | 0.93 (0.8; 1.0) | 0.89 (0.6; 1.0) | 0.72 (0.5; 1.0) | 0.68 (0.5; 0.9) | 0.51 (0.4; 0.7) |
| TopKLast0.1 (T=10) | 0.99 (0.9; 1.0) | 0.97 (0.8; 1.0) | 0.90 (0.7; 1.0) | 0.67 (0.4; 0.9) | 0.61 (0.5; 0.8) | 0.58 (0.4; 0.8) |
| TopKLast0.3 (T=3) | 0.99 (0.9; 1.0) | 0.97 (0.9; 1.0) | 0.93 (0.7; 1.0) | 0.81 (0.7; 1.0) | 0.84 (0.7; 1.0) | 0.78 (0.5; 1.0) |
| TopKLast0.3 (T=10) | 0.99 (1.0; 1.0) | 0.98 (0.9; 1.0) | 0.95 (0.7; 1.0) | 0.94 (0.7; 1.0) | 0.89 (0.7; 1.0) | 0.88 (0.7; 1.0) |
| TopKLast0.5 (T=3) | 0.98 (0.9; 1.0) | 0.98 (0.9; 1.0) | 0.95 (0.9; 1.0) | 0.95 (0.8; 1.0) | 0.86 (0.7; 1.0) | 0.86 (0.6; 0.9) |
| TopKLast0.5 (T=10) | 0.99 (1.0; 1.0) | 0.98 (0.9; 1.0) | 0.98 (1.0; 1.0) | 0.99 (0.8; 1.0) | 0.99 (0.8; 1.0) | 1.00 (0.8; 1.0) |
| TopKLast0.7 (T=3) | 0.98 (1.0; 1.0) | 0.98 (0.9; 1.0) | 0.94 (0.9; 1.0) | 0.83 (0.7; 1.0) | 0.87 (0.6; 1.0) | 0.82 (0.7; 1.0) |
| TopKLast0.7 (T=10) | 0.99 (1.0; 1.0) | 0.99 (1.0; 1.0) | 1.00 (1.0; 1.0) | 0.98 (0.8; 1.0) | 0.99 (0.9; 1.0) | 0.95 (0.8; 1.0) |

| | $Modes: 1$ | $Modes: 2$ | $Modes: 3$ | $Modes: 5$ | $Modes: 7$ | $Modes: 10$ |
|---|---|---|---|---|---|---|
| Vanilla | −4.94 (−5.5; −4.4) | −6.18 (−51.7; −5.6) | −31.85 (−100.3; −5.8) | −47.73 (−155.1; −14.2) | −107.36 (−390.8; −14.8) | −59.19 (−264.3; −18.8) |
| Boosted (T=3) | −4.56 (−4.9; −4.4) | −5.55 (−5.9; −5.2) | −5.01 (−6.7; −4.7) | −5.49 (−18.7; −4.9) | −5.60 (−14.5; −5.0) | −6.86 (−47.3; −5.6) |
| Boosted (T=10) | −4.56 (−4.7; −4.5) | −5.46 (−5.6; −5.3) | −5.08 (−5.8; −4.7) | −5.04 (−5.5; −4.6) | −5.51 (−5.9; −5.1) | −5.51 (−6.0; −5.2) |
| TopKLast0.1 (T=3) | −4.98 (−5.2; −4.7) | −5.64 (−6.1; −5.4) | −5.70 (−6.3; −5.2) | −5.39 (−38.4; −5.0) | −7.00 (−66.6; −5.4) | −12.70 (−44.2; −6.7) |
| TopKLast0.1 (T=10) | −4.98 (−5.3; −4.7) | −5.57 (−5.9; −5.3) | −5.37 (−6.0; −5.0) | −5.57 (−45.1; −4.7) | −7.34 (−16.1; −5.3) | −8.86 (−27.6; −5.5) |
| TopKLast0.3 (T=3) | −4.73 (−5.1; −4.5) | −5.48 (−6.0; −5.2) | −5.22 (−5.7; −4.8) | −5.42 (−21.6; −5.0) | −5.76 (−13.6; −5.1) | −7.26 (−36.2; −5.5) |
| TopKLast0.3 (T=10) | −4.62 (−4.8; −4.5) | −5.41 (−5.7; −5.2) | −4.90 (−5.2; −4.7) | −5.24 (−5.8; −4.9) | −5.71 (−6.2; −5.1) | −5.75 (−7.4; −5.1) |
| TopKLast0.5 (T=3) | −4.59 (−4.9; −4.4) | −5.29 (−5.7; −5.2) | −5.41 (−5.9; −4.9) | −5.48 (−18.5; −5.0) | −5.82 (−15.6; −5.2) | −6.78 (−18.7; −6.0) |
| TopKLast0.5 (T=10) | −4.59 (−4.8; −4.5) | −5.35 (−5.6; −5.2) | −5.12 (−5.5; −4.9) | −5.35 (−5.6; −4.8) | −5.34 (−5.8; −4.9) | −6.00 (−6.3; −5.6) |
| TopKLast0.7 (T=3) | −4.56 (−4.7; −4.4) | −5.37 (−5.5; −5.2) | −5.05 (−11.1; −4.7) | −5.63 (−43.1; −5.0) | −5.99 (−24.8; −5.4) | −7.76 (−25.2; −5.9) |
| TopKLast0.7 (T=10) | −4.52 (−4.7; −4.5) | −5.29 (−5.4; −5.2) | −5.05 (−6.6; −4.7) | −5.38 (−5.9; −5.1) | −5.77 (−6.3; −5.3) | −6.10 (−6.4; −6.0) |

Table 4: Reweighting similar to "Cascade GAN" from [14], i.e. keep the top $r$ fraction of examples, based on the discriminator corresponding to the previous generator. The mixture weights are all equal (i.e. $\beta = 1/t$). The reported scores are the median and interval defined by the 5% and 95% percentile (in parenthesis) (see Section 4.1.2), over 35 runs for each setting. The top table reports the coverage $C$, probability mass of $P_d$ covered by the $5th$ percentile of $P_g$ defined in Section 4.1.2. The bottom table reports the log likelihood of the true data under $P_g$.

| | $Modes: 1$ | $Modes: 2$ | $Modes: 3$ | $Modes: 5$ | $Modes: 7$ | $Modes: 10$ |
|---|---|---|---|---|---|---|
| Vanilla | 0.97 (0.9; 1.0) | 0.77 (0.5; 1.0) | 0.65 (0.5; 0.9) | 0.70 (0.5; 0.8) | 0.61 (0.5; 0.8) | 0.58 (0.3; 0.8) |
| Boosted (T=3) | 0.99 (1.0; 1.0) | 0.99 (0.9; 1.0) | 0.97 (0.9; 1.0) | 0.95 (0.8; 1.0) | 0.91 (0.8; 1.0) | 0.89 (0.8; 1.0) |
| Boosted (T=10) | 0.99 (1.0; 1.0) | 0.99 (1.0; 1.0) | 1.00 (1.0; 1.0) | 1.00 (1.0; 1.0) | 1.00 (1.0; 1.0) | 1.00 (1.0; 1.0) |
| TopK0.1 (T=3) | 0.98 (0.9; 1.0) | 0.98 (0.8; 1.0) | 0.91 (0.7; 1.0) | 0.84 (0.7; 1.0) | 0.80 (0.5; 0.9) | 0.60 (0.4; 0.7) |
| TopK0.1 (T=10) | 0.99 (1.0; 1.0) | 1.00 (1.0; 1.0) | 0.98 (1.0; 1.0) | 1.00 (1.0; 1.0) | 1.00 (1.0; 1.0) | 0.96 (0.8; 1.0) |
| TopK0.3 (T=3) | 0.98 (0.9; 1.0) | 0.98 (0.9; 1.0) | 0.95 (0.9; 1.0) | 0.95 (0.8; 1.0) | 0.84 (0.6; 1.0) | 0.79 (0.5; 1.0) |
| TopK0.3 (T=10) | 0.99 (1.0; 1.0) | 0.99 (1.0; 1.0) | 0.98 (1.0; 1.0) | 1.00 (1.0; 1.0) | 1.00 (1.0; 1.0) | 1.00 (1.0; 1.0) |
| TopK0.5 (T=3) | 0.99 (0.9; 1.0) | 0.99 (1.0; 1.0) | 0.96 (0.9; 1.0) | 0.98 (0.8; 1.0) | 0.88 (0.7; 1.0) | 0.88 (0.6; 1.0) |
| TopK0.5 (T=10) | 1.00 (1.0; 1.0) | 0.99 (1.0; 1.0) | 0.99 (1.0; 1.0) | 1.00 (1.0; 1.0) | 1.00 (1.0; 1.0) | 1.00 (1.0; 1.0) |
| TopK0.7 (T=3) | 0.98 (1.0; 1.0) | 0.98 (0.9; 1.0) | 0.94 (0.8; 1.0) | 0.84 (0.8; 1.0) | 0.86 (0.7; 1.0) | 0.81 (0.7; 1.0) |
| TopK0.7 (T=10) | 0.99 (1.0; 1.0) | 0.99 (1.0; 1.0) | 0.99 (1.0; 1.0) | 1.00 (0.8; 1.0) | 1.00 (0.9; 1.0) | 1.00 (0.9; 1.0) |

| | $Modes: 1$ | $Modes: 2$ | $Modes: 3$ | $Modes: 5$ | $Modes: 7$ | $Modes: 10$ |
|---|---|---|---|---|---|---|
| Vanilla | −4.61 (−5.5; −4.4) | −5.92 (−94.2; −5.2) | −12.40 (−53.1; −5.3) | −59.62 (−154.6; −9.8) | −66.95 (−191.5; −9.7) | −63.49 (−431.6; −14.5) |
| Boosted (T=3) | −4.59 (−4.9; −4.4) | −5.32 (−5.7; −5.2) | −5.60 (−5.8; −5.5) | −5.40 (−24.2; −4.5) | −5.71 (−14.0; −5.1) | −6.96 (−17.1; −5.9) |
| Boosted (T=10) | −4.61 (−4.7; −4.5) | −5.30 (−5.4; −5.2) | −5.48 (−5.6; −5.2) | −4.84 (−5.1; −4.3) | −5.25 (−5.9; −4.8) | −5.95 (−6.1; −5.5) |
| TopK0.1 (T=3) | −4.93 (−5.3; −4.7) | −5.85 (−6.2; −5.4) | −5.38 (−5.7; −5.0) | −5.34 (−5.8; −4.8) | −5.79 (−32.1; −5.2) | −7.09 (−20.7; −5.9) |
| TopK0.1 (T=10) | −4.60 (−4.8; −4.5) | −5.47 (−5.7; −5.3) | −4.81 (−5.1; −4.7) | −4.90 (−5.3; −4.2) | −4.85 (−5.6; −4.1) | −4.57 (−5.3; −4.2) |
| TopK0.3 (T=3) | −4.65 (−4.9; −4.4) | −5.40 (−5.9; −5.3) | −4.98 (−6.2; −4.7) | −5.25 (−11.4; −4.7) | −5.96 (−28.0; −5.5) | −7.34 (−25.4; −5.9) |
| TopK0.3 (T=10) | −4.56 (−4.8; −4.5) | −5.32 (−5.5; −5.2) | −5.07 (−5.9; −4.7) | −5.08 (−5.4; −4.5) | −5.16 (−5.9; −4.9) | −5.82 (−6.2; −5.3) |
| TopK0.5 (T=3) | −4.60 (−4.8; −4.5) | −5.34 (−5.6; −5.2) | −5.34 (−5.7; −5.0) | −5.42 (−19.0; −5.0) | −5.59 (−34.7; −4.9) | −6.15 (−14.8; −5.6) |
| TopK0.5 (T=10) | −4.59 (−4.7; −4.5) | −5.31 (−5.4; −5.2) | −5.13 (−5.5; −4.9) | −5.35 (−5.7; −4.8) | −5.33 (−5.8; −4.8) | −5.72 (−6.2; −5.3) |
| TopK0.7 (T=3) | −4.60 (−5.0; −4.4) | −5.44 (−5.6; −5.2) | −5.62 (−6.0; −5.4) | −5.49 (−22.2; −5.0) | −5.64 (−27.7; −5.3) | −7.17 (−22.5; −6.0) |
| TopK0.7 (T=10) | −4.59 (−4.7; −4.5) | −5.34 (−5.5; −5.2) | −5.51 (−5.6; −5.4) | −5.35 (−5.8; −5.0) | −5.32 (−6.0; −5.1) | −6.11 (−6.4; −5.9) |

Table 5: Reweighting using the top $r$ fraction of examples, based on the discriminator corresponding to the mixture of all previous generators. The mixture weights are all equal (i.e. $\beta = 1/t$). The reported scores are the median and interval defined by the 5% and 95% percentile (in parenthesis) (see Section 4.1.2), over 35 runs for each setting. The top table reports the coverage $C$, probability mass of $P_d$ covered by the $5th$ percentile of $P_g$ defined in Section 4.1.2. The bottom table reports the log likelihood of the true data under $P_g$.