AdaGAN: Boosting Generative Models

Ilya Tolstikhin
MPI for Intelligent Systems
Tübingen, Germany
`ilya@tue.mpg.de`

Sylvain Gelly
Google Brain
Zürich, Switzerland
`sylvaingelly@google.com`

Olivier Bousquet
Google Brain
Zürich, Switzerland
`obousquet@google.com`

Carl-Johann Simon-Gabriel
MPI for Intelligent Systems
Tübingen, Germany
`cjsimon@tue.mpg.de`

Bernhard Schölkopf
MPI for Intelligent Systems
Tübingen, Germany
`bs@tue.mpg.de`

Abstract

Generative Adversarial Networks (GAN) 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 *re-weighted* sample. This is inspired by *boosting* algorithms, where many potentially weak individual predictors are greedily aggregated to form a strong composite predictor. We prove analytically 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.

31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA.

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 [1, 2]. One popular approach are *Generative Adversarial Networks* (GAN) [2], where the generator 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 [1], 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 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.

ALGORITHM 1 AdaGAN, a meta-algorithm to construct a "strong" mixture of T individual generative models (f.ex. GANs), trained sequentially.

```
Input: Training sample $S_N := \{X_1, \dots, X_N\}$.
Output: Mixture generative model $G = G_T$.
  Train vanilla GAN $G_1 = \mathrm{GAN}(S_N, W_1)$ with a uniform weight $W_1 = (1/N, \dots, 1/N)$ over the training points
  for $t = 2, \dots, T$ do
    #Choose the overall weight of the next mixture component
    $\beta_t = \mathrm{ChooseMixtureWeight}(t)$
    #Update the weight of each training example
    $W_t = \mathrm{UpdateTrainingWeights}(G_{t-1}, S_N, \beta_t)$
    #Train t-th "weak" component generator $G_t^c$
    $G_t^c = \mathrm{GAN}(S_N, W_t)$
    #Update the overall generative model:
    #Form 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
```

Related Work Several authors [6, 7, 8] have proposed to use boosting techniques in the context of density estimation by incrementally adding components in the log domain. This idea was applied to GANs in [8]. 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 techniques such as Annealed Importance Sampling [9] for the normalization.

When the log likelihood can be computed, [10] proposed to use an additive mixture model. They derived the update rule via computing the steepest descent direction when adding a component with infinitesimal weight. However, their results do not apply once the weight β 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) for a specified f-divergence. In both theories, improvements of the mixture are guaranteed only if the new "weak" learner is still good enough (see Conditions 10&11)

Similarly, [11] 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.

An additive procedure similar to ours was proposed in [12] but with a different re-weighting scheme, which is not motivated by a theoretical analysis of optimality conditions. On every new iteration the authors run GAN on the k training examples with maximal values of the discriminator from the last iteration.

---

[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.

Finally, many papers investigate completely different approaches for addressing the same issue by directly modifying the training objective of an individual GAN. For instance, [5] add an autoencoding cost to the training objective of GAN, while [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 iterative optimization of mixture models under general f-divergences. In Section 2.4 we show that if 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 2.5 that imperfect solutions still lead to the exponential rate of convergence under certain "weak learnability" conditions. These results naturally lead to a new boosting-style iterative procedure for constructing generative models. When used with GANs, it results in our *AdaGAN* algorithm, detailed in Section 3 . Finally, we report initial empirical results in Section 4, where we compare AdaGAN with several benchmarks, including original GAN and uniform mixture of multiple independently trained GANs. Part of new theoretical results are reported without proofs, which can be found in appendices.

2 Minimizing f-divergence with Mixtures

2.1 Preliminaries and notations

Generative Density Estimation In density estimation, one tries to approximate a real data distribution P_d, defined over the data space $\mathcal{X}$, by a model distribution P_{model}. In the generative approach 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. $P_{model}(A) = P_Z(G^{-1}(A))$. With this approach it is in general impossible to compute the density $dP_{model}(x)$ and the log-likelihood of the training data under the model, but one can easily sample from P_{model} by sampling from P_Z and applying G. Thus, to construct G, instead of comparing P_{model} directly with P_d, one compares their samples. To do so, one uses a similarity measure $D(P_{model}\|P_d)$ 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 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 μ (we refer to Appendix D for more details about such divergences and their domain of definition). Here we assume that f is convex, defined on $(0, \infty)$, and satisfies $f(1) = 0$. We will denote by $\mathcal{F}$ the set of such functions. [2]

As demonstrated in [16, 17], 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[3], the Hellinger distance and the Total Variation among others. We will denote by $\mathcal{F}_H$ the set of functions f such that D_f is a Hilbertian metric.

GAN and f-divergences The original GAN algorithm [2] optimizes 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. This optimization is performed on a pair of samples (a training sample from P_d and a "fake" sample from P_Z), which corresponds to approximating the above criterion by using the empirical distributions. In the non-parametric limit for D, this is equivalent to minimizing the Jensen-Shannon divergence [2]. This point of view can be generalized to any other f-divergence [13]. Because of this strong connection between adversarial

---

[2] Examples of f-divergences include the Kullback-Leibler divergence (obtained for $f(x) = x \log x$) and Jensen-Shannon divergence ($f(x) = -(x+1)\log\frac{x+1}{2} + x\log x$). Other examples can be found in [13]. For further details we refer to Section 1.3 of [14] and [15].

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

training of generative models and minimization of f-divergences, we cast the results of this section into the context of general f-divergences.

Generative Mixture Models In order to model complex data distributions, it can be convenient to use a mixture model of the following form: $P^T_{model} := \sum_{i=1}^T \alpha_i P_i$, where $\alpha_i \geq 0$, $\sum_i \alpha_i = 1$, and each of the T components is a generative density model. This is 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 with parameters α_i) and then samples from it. Also, this allows to construct complex models from simpler ones.

2.2 Incremental Mixture Building

We restrict ourselves to the case of f-divergences and 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[4]

$$\min_{Q \in \mathcal{G}} D_f(Q \,\|\, P). \tag{3}$$

Instead of modelling the data with a single distribution, we now want to model it with a mixture of distributions P_i,where each P_i is obtained by a training procedure of the form (3) 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 $P^1_{model} = P_1$. 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 β as follows:

$$P^{t+1}_{model} := \sum_{i=1}^{t} (1-\beta)\alpha_i P_i + \beta Q. \tag{4}$$

where $\beta \in [0, 1]$ and $Q \in \mathcal{G}$ is computed by minimizing:

$$\min_{Q} D_f((1-\beta)P_g + \beta Q \,\|\, P_d), \tag{5}$$

where we denoted $P_g := P^t_{model}$ the current generative mixture model before adding the new component. We do not expect to find the optimal Q that minimizes (5) at each step, but we aim at constructing some Q that slightly improves our current approximation of P_d, i.e. such that for $c < 1$

$$D_f((1-\beta)P_g + \beta Q \,\|\, P_d) \leq c \cdot D_f(P_g \,\|\, P_d)\,. \tag{6}$$

This greedy approach has a significant drawback in practice. As we build up the mixture, we need to make β 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 (5) using samples from both distributions, this means that the sample from the mixture will only contain a fraction β 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 (5) which involves a term of the form $D_f(Q \,\|\, R)$ for some distribution R, which can be computed as a re-weighting of the original data distribution P_d. This procedure is reminiscent of the AdaBoost algorithm [18], which combines multiple *weak* predictors into one *strong* composition. On each step AdaBoost adds new predictor to the current composition, which is trained to minimize the binary loss on the re-weighted training set. The weights are constantly updated to bias the next weak learner towards "hard" examples, which were incorrectly classified during previous stages.

In the following we will analyze the properties of (5) and derive upper bounds that provide practical optimization criteria for building the mixture. We will also show that under certain assumptions, the minimization of the upper bound leads to the optimum of the original criterion.

2.3 Upper Bounds

We provide 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.

---

[4] One example of such a setting is running GANs.

Lemma 1 *Given two distributions* P_d, P_g *and some* $\beta \in [0,1]$*, then, for any* Q *and* R*, and* $f \in \mathcal{F}_H$*:*

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

If, more generally, $f \in \mathcal{F}$*, but* $\beta dR \leq dP_d$*, then:*

$$D_f((1-\beta)P_g + \beta Q \,\|\, P_d) \leq \beta D_f(Q \,\|\, R) + (1-\beta) D_f\left(P_g \,\|\, \frac{P_d - \beta R}{1-\beta}\right). \tag{8}$$

We can thus exploit those bounds by introducing some well-chosen distribution R and then minimizing them 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). Our main result indicates the shape of the distributions minimizing the right-most terms in those bounds.

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 minimizer of* (5) *over all probability distributions* $\mathbb{P}$ *has density*

$$dQ^*_\beta(x) = \frac{1}{\beta}\left(\lambda^* dP_d(x) - (1-\beta)dP_g(x)\right)_+ = \frac{dP_d}{\beta}\left(\lambda^* - (1-\beta)\frac{dP_g}{dP_d}\right)_+. \tag{9}$$

for the unique $\lambda^* \in [\beta, 1]$ *satisfying* $\int dQ^*_\beta = 1$*. Also,* $\lambda^* = 1$ *if and only if* $P_d((1-\beta)dP_g > dP_d) = 0$*, which is equivalent to* $\beta dQ^*_\beta = dP_d - (1-\beta)dP_g$*.*

Theorem 2 *Given two distributions* P_d, P_g *and some* $\beta \in (0,1]$*, assume* $P_d\,(dP_g = 0) < \beta$*. Let* $f \in \mathcal{F}$*. The problem*

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

has a solution with the density $dQ^\dagger_\beta(x) = \frac{1}{\beta}\big(dP_d(x) - \lambda^\dagger(1-\beta)dP_g(x)\big)_+$ *for the unique* $\lambda^\dagger \geq 1$ *that satisfies* $\int dQ^\dagger_\beta = 1$*.*

Surprisingly, in both Theorems 1 and 2, the solutions do not depend on the choice of the function f, which means that the solution is the same for *any* f-divergence[5]. Note that λ^* is implicitly defined by a fixed-point equation. In Section 3 we will show how it can be computed efficiently in the case of empirical distributions.

2.4 Convergence Analysis for Optimal Updates

In previous section we derived analytical expressions for the distributions R minimizing last terms in upper bounds (8) and (7). Assuming Q can perfectly match R, i.e. $D_f(Q \,\|\, R) = 0$, we are now interested in the convergence of the mixture (4) to the true data distribution P_d when $Q = Q^*_\beta$ or $Q = Q^\dagger_\beta$. We start with simple results showing that adding Q^*_β or $Q^\dagger_\beta$ to the current mixture would yield a strict improvement of the divergence.

Lemma 2 (Property 6: exponential improvements) *Under the conditions of Theorem 1, we have*

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

Under the conditions of Theorem 2, we have

$$D_f\left(P_g \,\|\, \frac{P_d - \beta Q^\dagger_\beta}{1-\beta}\right) \leq D_f(P_g \,\|\, P_d) \text{ and } D_f\big((1-\beta)P_g + \beta Q^\dagger_\beta \,\big\|\, P_d\big) \leq (1-\beta) D_f(P_g \,\|\, P_d).$$

Imagine repeatedly adding T new components to the current mixture P_g, where on every step we use the same weight β 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 $(1-\beta)^T D_f(P_g \,\|\, P_d)$.

---

[5] in particular, by replacing f with $f^\circ(x) := xf(1/x)$, we get the same solution for the criterion written in the other direction. Hence the order in which we write the divergence does not matter and the optimal solution is optimal for both orders.

This exponential rate of convergence, which at first may look surprisingly good, is simply explained by the fact that Q^*_β depends on the true distribution P_d, which is of course unknown.

Lemma 2 also suggests setting β as large as possible since we assume we can compute the optimal mixture component (which for $\beta = 1$ is P_d). However, in practice we may prefer to keep β 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 β in Section 3.

2.5 Weak to Strong Learnability

In practice the component Q that we add to the mixture is not exactly Q^*_β or $Q^\dagger_\beta$, but rather an approximation to them. In this section we show that if this approximation is good enough, then we retain the property (6) (exponential improvements).

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. $D_f(Q \,\|\, R) = 0$. In this case both sides of (7) are equal to $D_f((1-\beta)P_g + \beta Q^*_\beta \,\|\, P_d)$, which is the optimal value for the original objective (5). On the other hand, (8) does not become an equality and the r.h.s. is not the optimal one for (5). However, earlier we agreed that our aim is to reach the modest goal (6) and next we show that this is indeed possible.Corollaries 1 and 2 provide sufficient conditions for strict improvements when we use the upper bounds (8) and (7) respectively.

Corollary 1 *Given P_d, P_g, and some $\beta \in (0,1]$, assume $P_d\left(\frac{dP_g}{dP_d} = 0\right) < \beta$. Let $Q^\dagger_\beta$ be as defined in Theorem 2. If Q is such that*

$$D_f(Q \,\|\, Q^\dagger_\beta) \le \gamma D_f(P_g \,\|\, P_d) \tag{10}$$

for $\gamma \in [0,1]$, then $D_f((1-\beta)P_g + \beta Q \,\|\, P_d) \le (1 - \beta(1-\gamma))D_f(P_g \,\|\, P_d)$.

Corollary 2 *Let $f \in \mathcal{F}_H$. Take any $\beta \in (0,1]$, P_d, P_g, and let Q^*_β be as defined in Theorem 1. If Q is such that*

$$D_f(Q \,\|\, Q^*_\beta) \le \gamma D_f(P_g \,\|\, P_d) \tag{11}$$

for some $\gamma \in [0,1]$, then $D_f((1-\beta)P_g + \beta Q \,\|\, P_d) \le C_{\gamma,\beta} \cdot D_f(P_g \,\|\, P_d)$, where $C_{\gamma,\beta} = \left(\sqrt{\gamma\beta} + \sqrt{1-\beta}\right)^2$ is strictly smaller than 1 as soon as $\gamma < \beta/4$ (and $\beta > 0$).

Conditions 10 and 11 may be compared to the "weak learnability" condition of AdaBoost. As long as our weak learner is able to solve the surrogate problem (3) of matching respectively $Q^\dagger_\beta$ or Q^*_β accurately enough, the original objective (5) is guaranteed to decrease as well. It should be however noted that Condition 11 with $\gamma < \beta/4$ is perhaps too strong to call it "weak learnability". Indeed, as already mentioned before, the weight β 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 11. This obstacle may be partially resolved by the fact that we will use a GAN to fit Q, which corresponds to a relatively rich[6] class of models $\mathcal{G}$ in (3). In other words, our weak learner is not so weak. On the other hand, Condition 10 of Corollary 1 is milder. No matter what $\gamma \in [0,1]$ and $\beta \in (0,1]$ are, the new component Q is guaranteed to strictly improve the objective functional. This comes at the price of the additional condition $P_d(dP_g/dP_d = 0) < \beta$, which asserts that β 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 β to the new component Q. However, in practice, both Conditions 10 and 11 are difficult to check. A rigorous analysis of situations when they are guaranteed is a direction for future research.

---

[6] The hardness of meeting Condition 11 of course largely depends on the class of models $\mathcal{G}$ used to fit Q in (3). For now we ignore this question and leave it for future research.

3 AdaGAN

We now describe the functions *ChooseMixtureWeight* and *UpdateTrainingWeights* of Algorithm 1. The complete AdaGAN meta-algorithm with the details of *UpdateTrainingWeight* and *ChooseMixtureWeight*, is summarized in Algorithm 3 of Appendix A.

UpdateTrainingWeights At each iteration we add a new component Q to the current mixture P_g with weight β. The component Q should approach the "optimal target" Q_β^* provided by (9) in Theorem 1. This distribution depends on the density ratio dP_g/dP_d, which is not directly accessible, but it can be estimated using adversarial training. Indeed, we can train a separate *mixture discriminator* D_M to distinguish between samples from P_d and samples from the current mixture P_g. It is known [13] that for an arbitrary f-divergence, there exists a corresponding function h such that the values of the optimal discriminator D_M are related to the density ratio by

$$\frac{dP_g}{dP_d}(x) = h\big(D_M(x)\big). \tag{12}$$

We can replace $dP_g(x)/dP_d(x)$ in (9) with $h\big(D_M(x)\big)$. For the Jensen-Shannon divergence, used by the original GAN algorithm, $h(z) = \frac{1-z}{z}$. In practice, when we compute dQ_β^* on the training sample $S_N = (X_1, \ldots, X_N)$, each example X_i receives weight

$$w_i = \frac{1}{\beta N}\big(\lambda^* - (1-\beta)h(d_i)\big)_+ , \quad \text{where} \quad d_i = D_M(X_i)\,. \tag{13}$$

The only remaining task is to determine λ^*. As the weights w_i in (13) must sum to 1, we get:

$$\lambda^* = \frac{\beta}{\sum_{i \in \mathcal{I}(\lambda^*)} p_i}\left(1 + \frac{(1-\beta)}{\beta}\sum_{i \in \mathcal{I}(\lambda^*)} p_i h(d_i)\right) \tag{14}$$

where $\mathcal{I}(\lambda) := \{i : \lambda > (1-\beta)h(d_i)\}$. To find $\mathcal{I}(\lambda^*)$, we sort $h(d_i)$ in increasing order: $h(d_1) \leq \ldots \leq h(d_N)$. Then $\mathcal{I}(\lambda^*)$ is a set consisting of the first k indices. We then successively test all k-s until the λ given by (14) verifies $(1-\beta)h(d_k) < \lambda \leq (1-\beta)h(d_{k+1})$. This procedure is guaranteed to converge by Theorem 1. It is summarized in Algorithm 2 of Appendix A

ChooseMixtureWeight For every β there is an optimal re-weighting scheme with weights given by (13). If the GAN could perfectly approximate its target Q_β^*, then choosing $\beta = 1$ would be optimal, because $Q_1^* = P_d$. But in practice, GANs cannot do that. So we propose to choose β heuristically by imposing that each generator of the final mixture model has same weight. This yields $\beta_t = 1/t$, where t is the iteration index. Other heuristics are proposed in Appendix B, but did not lead to any significant difference.

The optimal discriminator In practice it is of course hard to find the optimal discriminator D_M achieving the global maximum of the variational representation for the f-divergence and verifying (12). For the JS-divergence this would mean that D_M is the classifier achieving minimal expected cross-entropy loss in the binary classification between P_g and P_d. In practice, we observed that the reweighting (13) leads to the desired property of emphasizing at least some of the missing modes as long as D_M distinguishes reasonably between data points already covered by the current model P_g and those which are still missing. We found an early stopping (while training D_M) sufficient to achieve this. In the *worst case*, when D_M overfits and returns 1 for all true data points, the reweighting simply leads to the uniform distribution over the training set.

4 Experiments

We ran AdaGAN[7] 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 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.

---

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

Toy Datasets Our target distribution is a mixture of isotropic Gaussians over $\mathcal{R}^2$. The distances between the means are large enough to roughly avoid overlaps between different Gaussian components. We vary the number of modes to test how well each algorithm performs when there are fewer or more expected modes. We compare the baseline GAN algorithm with AdaGAN variations, and with other meta-algorithms that all use the same underlying GAN procedure. For details on these algorithms and on the architectures of the underlying generator and discriminator, see Appendix B.

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 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, where the bandwidth is chosen by cross validation. 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.

Figure 2 summarizes the performance of algorithms as a function of the number of iterations T. 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 3). Our boosting approach converges much faster. In addition, its 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 bounds of the intervals). Further empirical results are available in Appendix B, where we compared AdaGAN variations to several other baseline meta-algorithms in more details (Table 1) and combined AdaGAN with the unrolled GANs (UGAN) [4] (Figure 3). Interestingly, Figure 3 shows that AdaGAN ran with UGAN outperforms the vanilla UGAN on the toy datasets, demonstrating the advantage of using AdaGAN as a way to further improve the mode coverage of any existing GAN implementations.

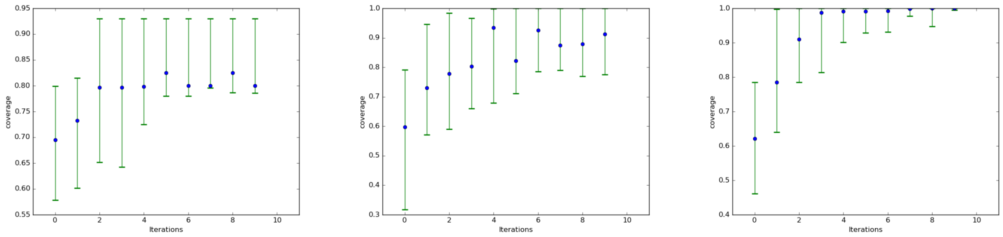

Figure 1: Coverage C of the true data by the model distribution P^T_{model}, as a function of iterations T. Experiments correspond to the data distribution with 5 modes. Each blue point is the median over 35 runs. Green intervals are defined by the 5% and 95% percentiles (see Section 4). Iteration 0 is equivalent to one vanilla GAN. The left plot corresponds to taking the best generator out of T runs. The middle plot is an "ensemble" GAN, simply taking a uniform mixture of T independently trained GAN generators. The right plot corresponds to our boosting approach (AdaGAN), with $\beta_t = 1/t$.

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 [19]. We used $T \in \{5, 10\}$, tried models of various sizes and performed a reasonable amount of hyperparameter search.

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 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 2). 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 ℓ_2-reconstruction sense. The "inversion" metric tests whether the trained model can generate certain examples or not, but unfortunately it does not take into account *the probabilities* of doing so.

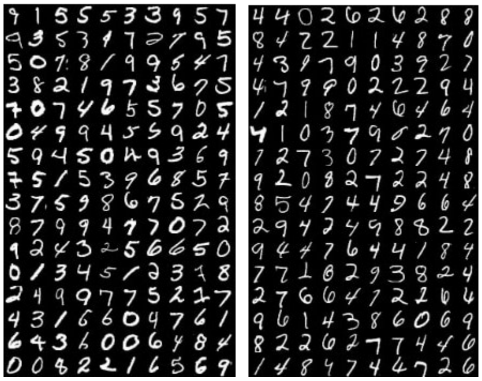

Figure 2: Digits from the MNIST dataset corresponding to the smallest (**left**) and largest (**right**) weights, obtained by the AdaGAN procedure (see Section 3) in one of the runs. Bold digits (left) are already covered and next GAN will concentrate on thin (right) digits.

5 Conclusion

We studied the problem of minimizing general f-divergences with additive mixtures of distributions. The main contribution of this work is a detailed theoretical analysis, which naturally leads to an iterative greedy procedure. On every iteration the mixture is updated with a new component, which minimizes f-divergence with a re-weighted target distribution. We provided conditions under which this procedure is guaranteed to converge to the target distribution at an exponential rate. While our results can be combined with any generative modelling techniques, we focused on GANs and provided a boosting-style algorithm *AdaGAN*. Preliminary experiments show that AdaGAN successfully produces a mixture which iteratively covers the missing modes.

References

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

[2] 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.

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

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

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

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

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

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

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

[13] 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.

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

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

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

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

[18] 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.

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

A Algorithms

ALGORITHM 2 Determining λ^*

```
  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
    $k \leftarrow k+1$
    $\lambda \leftarrow \frac{\beta}{\sum_{i=1}^{k} p_i}\left(1 + \frac{(1-\beta)}{\beta}\sum_{i=1}^{k} p_i h(d_i)\right)$
  end while
```

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). This is an instance of the high level Algorithm 1, instantiating UpdateTrainingWeights.

```
Input: Training sample $S_N := \{X_1, \ldots, X_N\}$.
Output: Mixture generative model $G = G_T$.
  Train vanilla GAN: $G_1 = \mathrm{GAN}(S_N)$
  for $t = 2, \ldots, T$ do
    #Choose a mixture weight for the next component
    $\beta_t = \mathrm{ChooseMixtureWeight}(t)$
    #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 $i = 1, \ldots, N$ 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 $G_t^c$
    $G_t^c = \mathrm{GAN}(S_N, W_t)$
    #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
```

B Details on the Toy Experiments

GAN architectures In all our experiments, the GAN's generator uses the 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. The optimizer is a simple SGD: Adam was also tried but gave slightly less stable results. All networks converge properly and overfitting is never an issue.

Details on the tested algorithms and more tests In our experiments, we compared the following algorithms:

- The baseline GAN algorithm, called **Vanilla GAN** in the results.

	$Modes: 1$	$Modes: 2$	$Modes: 3$	$Modes: 5$	$Modes: 10$
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)	0.59 (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)	0.70 (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)	0.71 (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)	0.80 (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)	0.89 (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 (0.6; 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)	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)	0.86 (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)

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. The reported scores are the median and interval defined by the 5% and 95% percentile (in parenthesis) (see Section 4), over 35 runs for each setting. 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.

(a) 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.

(b) 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 re-weighting:
  (c) The AdaGAN algorithm (Algorithm 1), with $\beta = 1/t$. Thus each component will have the same weight in the resulting mixture (see Equation 3). This procedure is called **Boosted** in the results.
  - The AdaGAN algorithm (Algorithm 1), but with a constant β, exploring several values. This procedure is called for example **Beta0.3** for $\beta = 0.3$ in the results. Note that in this setting, not all components of the mixture have the same weight.
  - Reweighting similar to "Cascade GAN" from [12], 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 $r = 0.3$.

The left, middle, and right panels in Figure 2 of Section 4 respectively correspond to the settings (a), (b) and (c).

Experiments with unrolled GAN To illustrate the 'meta-algorithm aspect' of AdaGAN, we also performed experiments with an unrolled GAN (UGAN) [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 [2] (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 3 illustrates our results. We find that AdaGAN works with all UGAN algorithms. Note that, where the usual GAN updates the generator and the discriminator once, an UGAN 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 UGAN roughly corresponds to doing 3 steps of AdaGAN with a usual GAN. In that sense, Figure 3 shows that AdaGAN (with a usual GAN)

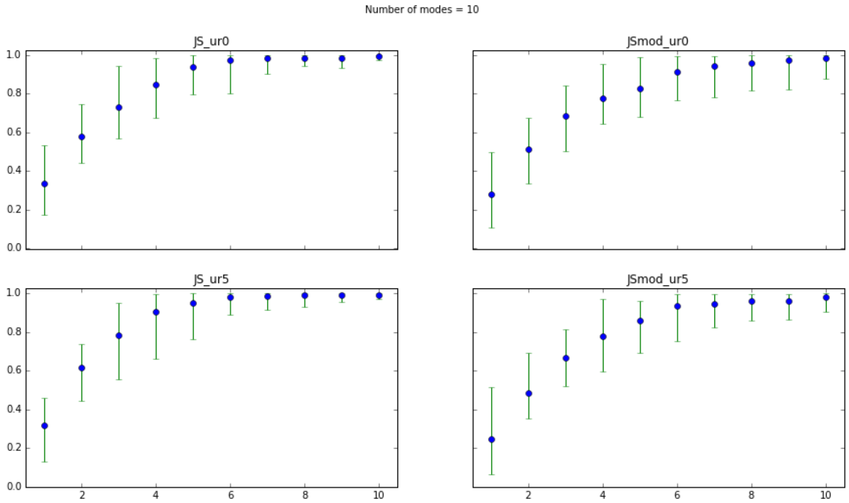

Figure 3: Comparison of AdaGAN ran with a GAN (top row) and with an unrolled GAN with 5 unrolling steps [4] (bottom). Coverage C of the true data by the model distribution P^T_{model}, as a function of iterations T. Experiments are similar to those of Figure 2, but with 10 modes. Left figures used the Jensen-Shannon objective (2), while right figures used the modified objective originally proposed by [2]. In terms of computation time, one step of AdaGAN with unrolled GAN corresponds to roughly 3 steps of AdaGAN with a usual GAN.

significantly outperforms a single unrolled GAN ($T = 1$ on bottom pictures). Also note that AdaGAN ran with UGAN outperforms a single UGAN and keeps improving its performance as we increase the number of iterations. Additionally, we note that using the Jensen-Shannon objective (rather than the modified version) seems to have some mode-regularizing effect.

C Details for AdaGAN on MNIST

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 4 for example, we see that after 1 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.

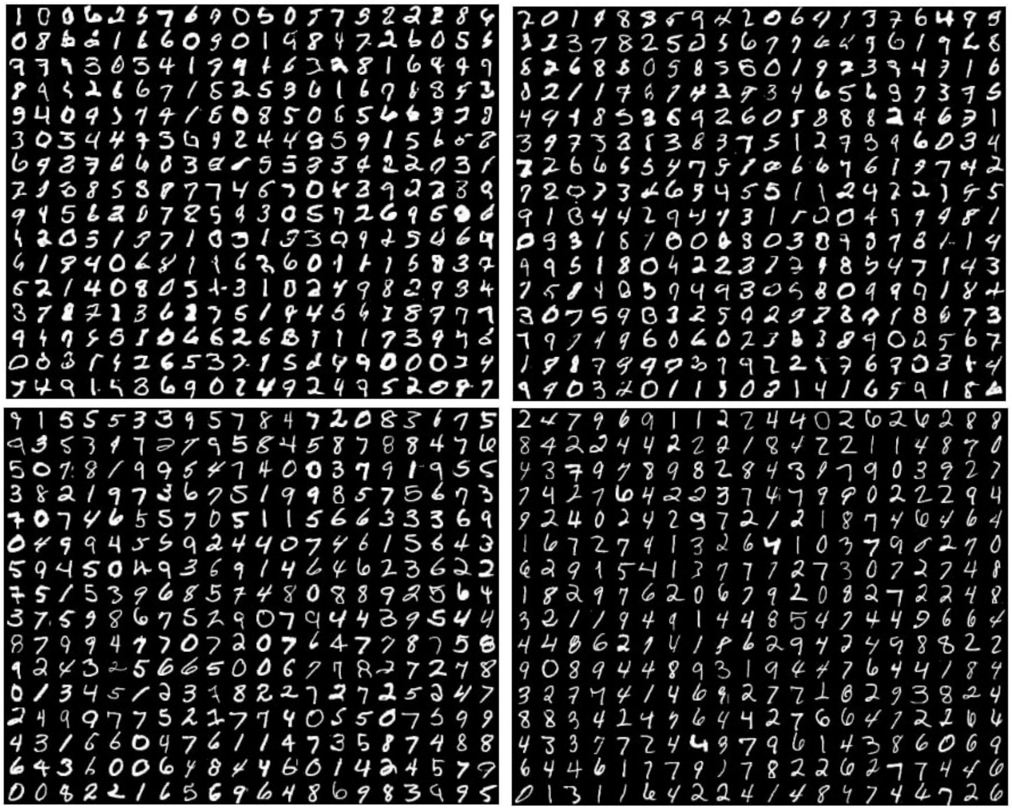

Figure 4: AdaGAN on MNIST. Bottom row are true MNIST digits with smallest (left) and highest (right) weights after re-weighting at the end of the first AdaGAN step. Those with small weight are thick and resemble those generated by the GAN after the first AdaGAN step (top left). After training with the re-weighted dataset during the second iteration of AdaGAN, the new mixture produces more thin digits (top right).

D Details on f-divergences

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 ∞ [14]. 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.

E Refinement of Lemma 2

If the ratio dP_g/dP_d is almost surely bounded, the first inequality of Lemma 2 can be refined as follows.

Lemma 3 *Under the conditions of Theorem 1*

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

given there exists $M > 1$ such that $P_d((1-\beta)dP_g > M dP_d) = 0$.

This upper bound can be tighter than that of Lemma 2 when λ^* 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.

Proof We use Inequality (18) 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:

$$D_f((1-\beta)P_g + \beta Q^*_\beta \,\|\, P_d) \leq f(\lambda^*) + \frac{f(M) - f(\lambda^*)}{M - \lambda^*}(1 - \lambda^*)\,. \tag{15}$$

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

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

■

F Conditions for finite steps convergence

Here we study the convergence of (5) to 0 in the case where, while performing the iterations, we use the upper bound (7) and the weight β is fixed (i.e. the same value at each iteration). We will provide necessary and sufficient conditions for the iterative process to converge to the data distribution P_d in finite number of steps. The analysis can easily be extended to a non-constant (variable) weight scheduling β. We start with the following result.

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 cannot 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.

We now provide the conditions for the convergence of the iterative process in a 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 3 *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^*_β 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*

$$P_d((1-\beta)dP_1 > M dP_d) = 0\,. \tag{16}$$

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 (16) 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 (16). ∎

It is also important to keep in mind that even if (16) is not satisfied the process still converges to the true distribution at exponential rate (see Lemma 2 as well as Corollaries 1 and 2 below)

G Proofs

G.1 Proof of Lemma 1

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 $\sqrt{D_f}$ and using convexity of D_f in its first argument.

G.2 Proof of Theorem 1

Before proving Theorem 1, we introduce two lemmas. The first one is about the determination of the constant λ, 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 $g(\lambda) \le \lambda$, and its right derivative is given by

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

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

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:

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

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

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

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 λ^* 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:

$$\mathbb{E}\left[f(\max(c, Y))\right] \leq \mathbb{E}\left[f(X + Y)\right] - \mathbb{E}\left[X(f'(Y) - f'(c))_+\right] \leq \mathbb{E}\left[f(X + Y)\right]. \tag{17}$$

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

$$\mathbb{E}\left[f(\max(c, Y))\right] \leq f(c) + \frac{f(M) - f(c)}{M - c}(\mathbb{E}\left[X + Y\right] - c). \tag{18}$$

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

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

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 λ^* in the interval $(\beta, 1]$, which shows that Q^*_β is indeed well-defined as a distribution.

Then we use Inequality (17) 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^*_β. ∎

G.3 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:

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

which shows that h is continuous. Also

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

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*[8]

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

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:

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

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:

$$\begin{aligned}
\beta &= \int \left(dP_d - \min\left(c(1 - \beta)P_g, dP_d\right)\right) \\
&= (1 - \beta) \int \left(\frac{1}{1 - \beta} - c\frac{dP_g}{dP_d}\right)_+ dP_d.
\end{aligned} \tag{19}$$

---

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

Using the fact that

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

we may apply Lemma 7 and conclude that there is a unique $c \in [1, \infty)$ satisfying (19), 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:

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

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

G.4 Proof of Lemma 2

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

G.5 Proof of Corollaries 1 and 2

For Corrollay 1, combine Lemma 1, Theorem 1, and Lemma 2. Corollary 2 immediately follows from Lemma 1, Theorem 2, and Lemma 2. It is easy to verify that for $\gamma < \beta/4$, the coefficient is less than $(\beta/2 + \sqrt{1-\beta})^2 < 1$ (for $\beta > 0$).