Improved Techniques for Training GANs

Tim Salimans tim@openai.com
Ian Goodfellow ian@openai.com
Wojciech Zaremba woj@openai.com
Vicki Cheung vicki@openai.com
Alec Radford alec@openai.com
Xi Chen peter@openai.com

Abstract

We present a variety of new architectural features and training procedures that we apply to the generative adversarial networks (GANs) framework. Using our new techniques, we achieve state-of-the-art results in semi-supervised classification on MNIST, CIFAR-10 and SVHN. The generated images are of high quality as confirmed by a visual Turing test: our model generates MNIST samples that humans cannot distinguish from real data, and CIFAR-10 samples that yield a human error rate of 21.3%. We also present ImageNet samples with unprecedented resolution and show that our methods enable the model to learn recognizable features of ImageNet classes.

1 Introduction

Generative adversarial networks [1] (GANs) are a class of methods for learning generative models based on game theory. The goal of GANs is to train a generator network $G(\boldsymbol{z}; \boldsymbol{\theta}^{(G)})$ that produces samples from the data distribution, $p_{\text{data}}(\boldsymbol{x})$, by transforming vectors of noise $\boldsymbol{z}$ as $\boldsymbol{x} = G(\boldsymbol{z}; \boldsymbol{\theta}^{(G)})$. The training signal for G is provided by a discriminator network $D(\boldsymbol{x})$ that is trained to distinguish samples from the generator distribution $p_{\text{model}}(\boldsymbol{x})$ from real data. The generator network G in turn is then trained to fool the discriminator into accepting its outputs as being real.

Recent applications of GANs have shown that they can produce excellent samples [2, 3]. However, training GANs requires finding a Nash equilibrium of a non-convex game with continuous, high-dimensional parameters. GANs are typically trained using gradient descent techniques that are designed to find a low value of a cost function, rather than to find the Nash equilibrium of a game. When used to seek for a Nash equilibrium, these algorithms may fail to converge [4].

In this work, we introduce several techniques intended to encourage convergence of the GANs game. These techniques are motivated by a heuristic understanding of the non-convergence problem. They lead to improved semi-supervised learning peformance and improved sample generation. We hope that some of them may form the basis for future work, providing formal guarantees of convergence.

All code and hyperparameters may be found at `https://github.com/openai/improved-gan`.

2 Related work

Several recent papers focus on improving the stability of training and the resulting perceptual quality of GAN samples [2, 3, 5, 6]. We build on some of these techniques in this work. For instance, we use some of the "DCGAN" architectural innovations proposed in Radford et al. [3], as discussed below.

One of our proposed techniques, *feature matching*, discussed in Sec. 3.1, is similar in spirit to approaches that use maximum mean discrepancy [7, 8, 9] to train generator networks [10, 11].

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

Another of our proposed techniques, *minibatch features*, is based in part on ideas used for batch normalization [12], while our proposed *virtual batch normalization* is a direct extension of batch normalization.

One of the primary goals of this work is to improve the effectiveness of generative adversarial networks for semi-supervised learning (improving the performance of a supervised task, in this case, classification, by learning on additional unlabeled examples). Like many deep generative models, GANs have previously been applied to semi-supervised learning [13, 14], and our work can be seen as a continuation and refinement of this effort. In concurrent work, Odena [15] proposes to extend GANs to predict image labels like we do in Section 5, but without our *feature matching* extension (Section 3.1) which we found to be critical for obtaining state-of-the-art performance.

3 Toward Convergent GAN Training

Training GANs consists in finding a Nash equilibrium to a two-player non-cooperative game. Each player wishes to minimize its own cost function, $J^{(D)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$ for the discriminator and $J^{(G)}(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$ for the generator. A Nash equilibirum is a point $(\boldsymbol{\theta}^{(D)}, \boldsymbol{\theta}^{(G)})$ such that $J^{(D)}$ is at a minimum with respect to $\boldsymbol{\theta}^{(D)}$ and $J^{(G)}$ is at a minimum with respect to $\boldsymbol{\theta}^{(G)}$. Unfortunately, finding Nash equilibria is a very difficult problem. Algorithms exist for specialized cases, but we are not aware of any that are feasible to apply to the GAN game, where the cost functions are non-convex, the parameters are continuous, and the parameter space is extremely high-dimensional.

The idea that a Nash equilibrium occurs when each player has minimal cost seems to intuitively motivate the idea of using traditional gradient-based minimization techniques to minimize each player's cost simultaneously. Unfortunately, a modification to $\boldsymbol{\theta}^{(D)}$ that reduces $J^{(D)}$ can increase $J^{(G)}$, and a modification to $\boldsymbol{\theta}^{(G)}$ that reduces $J^{(G)}$ can increase $J^{(D)}$. Gradient descent thus fails to converge for many games. For example, when one player minimizes xy with respect to x and another player minimizes $-xy$ with respect to y, gradient descent enters a stable orbit, rather than converging to $x = y = 0$, the desired equilibrium point [16]. Previous approaches to GAN training have thus applied gradient descent on each player's cost simultaneously, despite the lack of guarantee that this procedure will converge. We introduce the following techniques that are heuristically motivated to encourage convergence:

3.1 Feature matching

Feature matching addresses the instability of GANs by specifying a new objective for the generator that prevents it from overtraining on the current discriminator. Instead of directly maximizing the output of the discriminator, the new objective requires the generator to generate data that matches the statistics of the real data, where we use the discriminator only to specify the statistics that we think are worth matching. Specifically, we train the generator to match the expected value of the features on an intermediate layer of the discriminator. This is a natural choice of statistics for the generator to match, since by training the discriminator we ask it to find those features that are most discriminative of real data versus data generated by the current model.

Letting $\mathbf{f}(\boldsymbol{x})$ denote activations on an intermediate layer of the discriminator, our new objective for the generator is defined as: $||\mathbb{E}_{\boldsymbol{x} \sim p_{\text{data}}} \mathbf{f}(\boldsymbol{x}) - \mathbb{E}_{\boldsymbol{z} \sim p_{\boldsymbol{z}}(\boldsymbol{z})} \mathbf{f}(G(\boldsymbol{z}))||_2^2$. The discriminator, and hence $\mathbf{f}(\boldsymbol{x})$, are trained in the usual way. As with regular GAN training, the objective has a fixed point where G exactly matches the distribution of training data. We have no guarantee of reaching this fixed point in practice, but our empirical results indicate that feature matching is indeed effective in situations where regular GAN becomes unstable.

3.2 Minibatch discrimination

One of the main failure modes for GAN is for the generator to collapse to a parameter setting where it always emits the same point. When collapse to a single mode is imminent, the gradient of the discriminator may point in similar directions for many similar points. Because the discriminator processes each example independently, there is no coordination between its gradients, and thus no mechanism to tell the outputs of the generator to become more dissimilar to each other. Instead, all outputs race toward a single point that the discriminator currently believes is highly realistic. After collapse has occurred, the discriminator learns that this single point comes from the generator, but gradient descent is unable to separate the identical outputs. The gradients of the discriminator

then push the single point produced by the generator around space forever, and the algorithm cannot converge to a distribution with the correct amount of entropy. An obvious strategy to avoid this type of failure is to allow the discriminator to look at multiple data examples in combination, and perform what we call *minibatch discrimination*.

The concept of minibatch discrimination is quite general: any discriminator model that looks at multiple examples in combination, rather than in isolation, could potentially help avoid collapse of the generator. In fact, the successful application of batch normalization in the discriminator by Radford et al. [3] is well explained from this perspective. So far, however, we have restricted our experiments to models that explicitly aim to identify generator samples that are particularly close together. One successful specification for modelling the *closeness* between examples in a minibatch is as follows: Let $\mathbf{f}(\boldsymbol{x}_i) \in \mathbb{R}^A$ denote a vector of features for input $\boldsymbol{x}_i$, produced by some intermediate layer in the discriminator. We then multiply the vector $\mathbf{f}(\boldsymbol{x}_i)$ by a tensor $T \in \mathbb{R}^{A \times B \times C}$, which results in a matrix $M_i \in \mathbb{R}^{B \times C}$. We then compute the L_1-distance between the rows of the resulting matrix M_i across samples $i \in \{1, 2, \ldots, n\}$ and apply a negative exponential (Fig. 1): $c_b(\boldsymbol{x}_i, \boldsymbol{x}_j) = \exp(-||M_{i,b} - M_{j,b}||_{L_1}) \in \mathbb{R}$. The output $o(x_i)$ for this *minibatch layer* for a sample $\boldsymbol{x}_i$ is then defined as the sum of the $c_b(\boldsymbol{x}_i, \boldsymbol{x}_j)$'s to all other samples:

$$o(\boldsymbol{x}_i)_b = \sum_{j=1}^{n} c_b(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathbb{R}$$

$$o(\boldsymbol{x}_i) = \left[o(\boldsymbol{x}_i)_1, o(\boldsymbol{x}_i)_2, \ldots, o(\boldsymbol{x}_i)_B\right] \in \mathbb{R}^B$$

$$o(\mathbf{X}) \in \mathbb{R}^{n \times B}$$

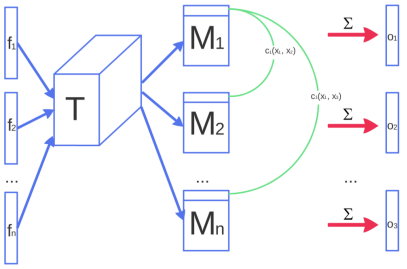


Figure 1: Figure sketches how minibatch discrimination works. Features $\mathbf{f}(\boldsymbol{x}_i)$ from sample $\boldsymbol{x}_i$ are multiplied through a tensor T, and cross-sample distance is computed.

Next, we concatenate the output $o(\boldsymbol{x}_i)$ of the minibatch layer with the intermediate features $\mathbf{f}(\boldsymbol{x}_i)$ that were its input, and we feed the result into the next layer of the discriminator. We compute these minibatch features separately for samples from the generator and from the training data. As before, the discriminator is still required to output a single number for each example indicating how likely it is to come from the training data: The task of the discriminator is thus effectively still to classify single examples as real data or generated data, but it is now able to use the other examples in the minibatch as *side information*. Minibatch discrimination allows us to generate visually appealing samples very quickly, and in this regard it is superior to feature matching (Section 6). Interestingly, however, feature matching was found to work much better if the goal is to obtain a strong classifier using the approach to semi-supervised learning described in Section 5.

3.3 Historical averaging

When applying this technique, we modify each player's cost to include a term $||\boldsymbol{\theta} - \frac{1}{t}\sum_{i=1}^{t} \boldsymbol{\theta}[i]||^2$, where $\boldsymbol{\theta}[i]$ is the value of the parameters at past time i. The historical average of the parameters can be updated in an online fashion so this learning rule scales well to long time series. This approach is loosely inspired by the fictitious play [17] algorithm that can find equilibria in other kinds of games. We found that our approach was able to find equilibria of low-dimensional, continuous non-convex games, such as the minimax game with one player controlling x, the other player controlling y, and value function $(f(x) - 1)(y - 1)$, where $f(x) = x$ for $x < 0$ and $f(x) = x^2$ otherwise. For these same toy games, gradient descent fails by going into extended orbits that do not approach the equilibrium point.

3.4 One-sided label smoothing

Label smoothing, a technique from the 1980s recently independently re-discovered by Szegedy et. al [18], replaces the 0 and 1 targets for a classifier with smoothed values, like .9 or .1, and was recently shown to reduce the vulnerability of neural networks to adversarial examples [19].

Replacing positive classification targets with α and negative targets with β, the optimal discriminator becomes $D(\boldsymbol{x}) = \frac{\alpha p_{\text{data}}(\boldsymbol{x}) + \beta p_{\text{model}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{\text{model}}(\boldsymbol{x})}$. The presence of p_{model} in the numerator is problematic because, in areas where p_{data} is approximately zero and p_{model} is large, erroneous samples from

p_{model} have no incentive to move nearer to the data. We therefore smooth *only* the positive labels to α, leaving negative labels set to 0.

3.5 Virtual batch normalization

Batch normalization greatly improves optimization of neural networks, and was shown to be highly effective for DCGANs [3]. However, it causes the output of a neural network for an input example $\boldsymbol{x}$ to be highly dependent on several other inputs $\boldsymbol{x}'$ in the same minibatch. To avoid this problem we introduce *virtual batch normalization* (VBN), in which each example $\boldsymbol{x}$ is normalized based on the statistics collected on a *reference batch* of examples that are chosen once and fixed at the start of training, and on $\boldsymbol{x}$ itself. The reference batch is normalized using only its own statistics. VBN is computationally expensive because it requires running forward propagation on two minibatches of data, so we use it only in the generator network.

4 Assessment of image quality

Generative adversarial networks lack an objective function, which makes it difficult to compare performance of different models. One intuitive metric of performance can be obtained by having human annotators judge the visual quality of samples [2]. We automate this process using Amazon Mechanical Turk (MTurk), using the web interface in figure Fig. 2 (live at `http://infinite-chamber-35121.herokuapp.com/cifar-minibatch/`), which we use to ask annotators to distinguish between generated data and real data. The resulting quality assessments of our models are described in Section 6.

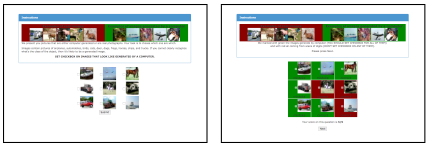


Figure 2: Web interface given to annotators. Annotators are asked to distinguish computer generated images from real ones.

A downside of using human annotators is that the metric varies depending on the setup of the task and the motivation of the annotators. We also find that results change drastically when we give annotators feedback about their mistakes: By learning from such feedback, annotators are better able to point out the flaws in generated images, giving a more pessimistic quality assessment. The left column of Fig. 2 presents a screen from the annotation process, while the right column shows how we inform annotators about their mistakes.

As an alternative to human annotators, we propose an automatic method to evaluate samples, which we find to correlate well with human evaluation: We apply the Inception model[1] [20] to every generated image to get the conditional label distribution $p(y|\boldsymbol{x})$. Images that contain meaningful objects should have a conditional label distribution $p(y|\boldsymbol{x})$ with low entropy. Moreover, we expect the model to generate varied images, so the marginal $\int p(y|\boldsymbol{x} = G(z))dz$ should have high entropy. Combining these two requirements, the metric that we propose is: $\exp(\mathbb{E}_{\boldsymbol{x}}\text{KL}(p(y|\boldsymbol{x})||p(y)))$, where we exponentiate results so the values are easier to compare. Our *Inception score* is closely related to the objective used for training generative models in CatGAN [14]: Although we had less success using such an objective for training, we find it is a good metric for evaluation that correlates very well with human judgment. We find that it's important to evaluate the metric on a large enough number of samples (i.e. $50k$) as part of this metric measures diversity.

5 Semi-supervised learning

Consider a standard classifier for classifying a data point $\boldsymbol{x}$ into one of K possible classes. Such a model takes in $\boldsymbol{x}$ as input and outputs a K-dimensional vector of logits $\{l_1, \ldots, l_K\}$, that can be turned into class probabilities by applying the softmax: $p_{\text{model}}(y = j|\boldsymbol{x}) = \frac{\exp(l_j)}{\sum_{k=1}^{K}\exp(l_k)}$. In supervised learning, such a model is then trained by minimizing the cross-entropy between the observed labels and the model predictive distribution $p_{\text{model}}(y|\boldsymbol{x})$.

[1] We use the pretrained Inception model from `http://download.tensorflow.org/models/image/imagenet/inception-2015-12-05.tgz`. Code to compute the Inception score with this model will be made available by the time of publication.

We can do semi-supervised learning with any standard classifier by simply adding samples from the GAN generator G to our data set, labeling them with a new "generated" class $y = K + 1$, and correspondingly increasing the dimension of our classifier output from K to $K + 1$. We may then use $p_{\text{model}}(y = K + 1 \mid \boldsymbol{x})$ to supply the probability that $\boldsymbol{x}$ is fake, corresponding to $1 - D(\boldsymbol{x})$ in the original GAN framework. We can now also learn from unlabeled data, as long as we know that it corresponds to one of the K classes of real data by maximizing $\log p_{\text{model}}(y \in \{1, \ldots, K\}|\boldsymbol{x})$. Assuming half of our data set consists of real data and half of it is generated (this is arbitrary), our loss function for training the classifier then becomes

$$\begin{aligned} L &= -\mathbb{E}_{\boldsymbol{x},y\sim p_{\text{data}}(\boldsymbol{x},y)}[\log p_{\text{model}}(y|\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim G}[\log p_{\text{model}}(y = K + 1|\boldsymbol{x})] \\ &= L_{\text{supervised}} + L_{\text{unsupervised}}, \text{ where} \\ L_{\text{supervised}} &= -\mathbb{E}_{\boldsymbol{x},y\sim p_{\text{data}}(\boldsymbol{x},y)} \log p_{\text{model}}(y|\boldsymbol{x}, y < K + 1) \\ L_{\text{unsupervised}} &= -\{\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})} \log[1 - p_{\text{model}}(y = K + 1|\boldsymbol{x})] + \mathbb{E}_{\boldsymbol{x}\sim G} \log[p_{\text{model}}(y = K + 1|\boldsymbol{x})]\}, \end{aligned}$$

where we have decomposed the total cross-entropy loss into our standard supervised loss function $L_{\text{supervised}}$ (the negative log probability of the label, given that the data is real) and an unsupervised loss $L_{\text{unsupervised}}$ which is in fact the standard GAN game-value as becomes evident when we substitute $D(\boldsymbol{x}) = 1 - p_{\text{model}}(y = K + 1|\boldsymbol{x})$ into the expression:

$$L_{\text{unsupervised}} = -\{\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}(\boldsymbol{x})} \log D(\boldsymbol{x}) + \mathbb{E}_{z\sim \text{noise}} \log(1 - D(G(\boldsymbol{z})))\}.$$

The optimal solution for minimizing both $L_{\text{supervised}}$ and $L_{\text{unsupervised}}$ is to have $\exp[l_j(\boldsymbol{x})] = c(\boldsymbol{x})p(y{=}j, \boldsymbol{x})\forall j{<}K{+}1$ and $\exp[l_{K+1}(\boldsymbol{x})] = c(\boldsymbol{x})p_G(\boldsymbol{x})$ for some undetermined scaling function $c(\boldsymbol{x})$. The unsupervised loss is thus consistent with the supervised loss in the sense of Sutskever et al. [13], and we can hope to better estimate this optimal solution from the data by minimizing these two loss functions jointly. In practice, $L_{\text{unsupervised}}$ will only help if it is not trivial to minimize for our classifier and we thus need to train G to approximate the data distribution. One way to do this is by training G to minimize the GAN game-value, using the discriminator D defined by our classifier. This approach introduces an interaction between G and our classifier that we do not fully understand yet, but empirically we find that optimizing G using feature matching GAN works very well for semi-supervised learning, while training G using GAN with minibatch discrimination does not work at all. Here we present our empirical results using this approach; developing a full theoretical understanding of the interaction between D and G using this approach is left for future work.

Finally, note that our classifier with $K + 1$ outputs is over-parameterized: subtracting a general function $f(\boldsymbol{x})$ from each output logit, i.e. setting $l_j(\boldsymbol{x}) \leftarrow l_j(\boldsymbol{x}) - f(\boldsymbol{x})\forall j$, does not change the output of the softmax. This means we may equivalently fix $l_{K+1}(\boldsymbol{x}) = 0\forall \boldsymbol{x}$, in which case $L_{\text{supervised}}$ becomes the standard supervised loss function of our original classifier with K classes, and our discriminator D is given by $D(\boldsymbol{x}) = \frac{Z(\boldsymbol{x})}{Z(\boldsymbol{x})+1}$, where $Z(\boldsymbol{x}) = \sum_{k=1}^{K} \exp[l_k(\boldsymbol{x})]$.

5.1 Importance of labels for image quality

Besides achieving state-of-the-art results in semi-supervised learning, the approach described above also has the surprising effect of improving the quality of generated images as judged by human annotators. The reason appears to be that the human visual system is strongly attuned to image statistics that can help infer what class of object an image represents, while it is presumably less sensitive to local statistics that are less important for interpretation of the image. This is supported by the high correlation we find between the quality reported by human annotators and the *Inception score* we developed in Section 4, which is explicitly constructed to measure the "objectness" of a generated image. By having the discriminator D classify the object shown in the image, we bias it to develop an internal representation that puts emphasis on the same features humans emphasize. This effect can be understood as a method for transfer learning, and could potentially be applied much more broadly. We leave further exploration of this possibility for future work.

6 Experiments

We performed semi-supervised experiments on MNIST, CIFAR-10 and SVHN, and sample generation experiments on MNIST, CIFAR-10, SVHN and ImageNet. We provide code to reproduce the majority of our experiments.

6.1 MNIST

The MNIST dataset contains $60,000$ labeled images of digits. We perform semi-supervised training with a small randomly picked fraction of these, considering setups with 20, 50, 100, and 200 labeled examples. Results are averaged over 10 random subsets of labeled data, each chosen to have a balanced number of examples from each class. The remaining training images are provided without labels. Our networks have 5 hidden layers each. We use weight normalization [21] and add Gaussian noise to the output of each layer of the discriminator. Table 1 summarizes our results.

Figure 3: *(Left)* samples generated by model during semi-supervised training. Samples can be clearly distinguished from images coming from MNIST dataset. *(Right)* Samples generated with minibatch discrimination. Samples are completely indistinguishable from dataset images.

Samples generated by the generator during semi-supervised learning using feature matching (Section 3.1) do not look visually appealing (left Fig. 3). By using minibatch discrimination instead (Section 3.2) we can improve their visual quality. On MTurk, annotators were able to distinguish samples in 52.4% of cases (2000 votes total), where 50% would be obtained by random guessing. Similarly, researchers in our institution were not able to find any artifacts that would allow them to distinguish samples. However, semi-supervised learning with minibatch discrimination does not produce as good a classifier as does feature matching.

Model	Number of incorrectly predicted test examples for a given number of labeled samples			
	20	50	100	200
DGN [22]			333 ± 14	
Virtual Adversarial [23]			212	
CatGAN [14]			191 ± 10	
Skip Deep Generative Model [24]			132 ± 7	
Ladder network [25]			106 ± 37	
Auxiliary Deep Generative Model [24]			96 ± 2	
Our model	1677 ± 452	221 ± 136	93 ± 6.5	90 ± 4.2
Ensemble of 10 of our models	1134 ± 445	142 ± 96	86 ± 5.6	81 ± 4.3

Table 1: Number of incorrectly classified test examples for the semi-supervised setting on permutation invariant MNIST. Results are averaged over 10 seeds.

6.2 CIFAR-10

Model	Test error rate for a given number of labeled samples			
	1000	2000	4000	8000
Ladder network [25]			20.40 ± 0.47	
CatGAN [14]			19.58 ± 0.46	
Our model	21.83 ± 2.01	19.61 ± 2.09	18.63 ± 2.32	17.72 ± 1.82
Ensemble of 10 of our models	19.22 ± 0.54	17.25 ± 0.66	15.59 ± 0.47	14.87 ± 0.89

Table 2: Test error on semi-supervised CIFAR-10. Results are averaged over 10 splits of data.

CIFAR-10 is a small, well studied dataset of 32×32 natural images. We use this data set to study semi-supervised learning, as well as to examine the visual quality of samples that can be achieved. For the discriminator in our GAN we use a 9 layer deep convolutional network with dropout and weight normalization. The generator is a 4 layer deep CNN with batch normalization. Table 2 summarizes our results on the semi-supervised learning task.

Figure 4: Samples generated during semi-supervised training on CIFAR-10 with feature matching (Section 3.1, *left*) and minibatch discrimination (Section 3.2, *right*).

When presented with 50% real and 50% fake data generated by our best CIFAR-10 model, MTurk users correctly categorized 78.7% of images correctly. However, MTurk users may not be sufficiently familiar with CIFAR-10 images or sufficiently motivated; we ourselves were able to categorize images with $> 95\%$ accuracy. We validated the Inception score described above by observing that MTurk accuracy drops to 71.4% when the data is filtered by using only the top 1% of samples according to the Inception score. We performed a series of ablation experiments to demonstrate that our proposed techniques improve the Inception score, presented in Table 3. We also present images for these ablation experiments—in our opinion, the Inception score correlates well with our subjective judgment of image quality. Samples from the dataset achieve the highest value. All the models that even partially collapse have relatively low scores. We caution that the Inception score should be used as a rough guide to evaluate models that were trained via some independent criterion; directly optimizing Inception score will lead to the generation of adversarial examples [26].

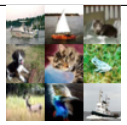
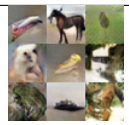
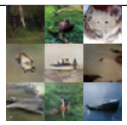
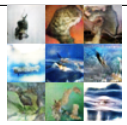
Samples							
Model	Real data	Our methods	-VBN+BN	-L+HA	-LS	-L	-MBF
Score $\pm$ std.	$11.24 \pm .12$	$8.09 \pm .07$	$7.54 \pm .07$	$6.86 \pm .06$	$6.83 \pm .06$	$4.36 \pm .04$	$3.87 \pm .03$

Table 3: Table of Inception scores for samples generated by various models for $50,000$ images. Score highly correlates with human judgment, and the best score is achieved for natural images. Models that generate collapsed samples have relatively low score. This metric allows us to avoid relying on human evaluations. "Our methods" includes all the techniques described in this work, except for feature matching and historical averaging. The remaining experiments are ablation experiments showing that our techniques are effective. "-VBN+BN" replaces the VBN in the generator with BN, as in DCGANs. This causes a small decrease in sample quality on CIFAR. VBN is more important for ImageNet. "-L+HA" removes the labels from the training process, and adds historical averaging to compensate. HA makes it possible to still generate some recognizable objects. Without HA, sample quality is considerably reduced (see "-L"). "-LS" removes label smoothing and incurs a noticeable drop in performance relative to "our methods." "-MBF" removes the minibatch features and incurs a very large drop in performance, greater even than the drop resulting from removing the labels. Adding HA cannot prevent this problem.

6.3 SVHN

For the SVHN data set, we used the same architecture and experimental setup as for CIFAR-10. Figure 5 compares against the previous state-of-the-art, where it should be noted that the model

of [24] is not convolutional, but does use an additional data set of 531131 unlabeled examples. The other methods, including ours, are convolutional and do not use this data.

Model	Percentage of incorrectly predicted test examples for a given number of labeled samples		
	500	1000	2000
Virtual Adversarial [23]		24.63	
Stacked What-Where Auto-Encoder [27]		23.56	
DCGAN [3]		22.48	
Skip Deep Generative Model [24]		16.61±0.24	
Our model	18.44 ± 4.8	8.11 ± 1.3	6.16 ± 0.58
Ensemble of 10 of our models		5.88 ± 1.0	

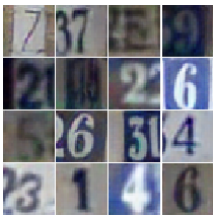


Figure 5: *(Left)* Error rate on SVHN. *(Right)* Samples from the generator for SVHN.

6.4 ImageNet

We tested our techniques on a dataset of unprecedented scale: 128×128 images from the ILSVRC2012 dataset with 1,000 categories. To our knowledge, no previous publication has applied a generative model to a dataset with both this large of a resolution and this large a number of object classes. The large number of object classes is particularly challenging for GANs due to their tendency to underestimate the entropy in the distribution. We extensively modified a publicly available implementation of DCGANs[2] using TensorFlow [28] to achieve high performance, using a multi-GPU implementation. DCGANs without modification learn some basic image statistics and generate contiguous shapes with somewhat natural color and texture but do not learn any objects. Using the techniques described in this paper, GANs learn to generate objects that resemble animals, but with incorrect anatomy. Results are shown in Fig. 6.

Figure 6: Samples generated from the ImageNet dataset. *(Left)* Samples generated by a DCGAN. *(Right)* Samples generated using the techniques proposed in this work. The new techniques enable GANs to learn recognizable features of animals, such as fur, eyes, and noses, but these features are not correctly combined to form an animal with realistic anatomical structure.

7 Conclusion

Generative adversarial networks are a promising class of generative models that has so far been held back by unstable training and by the lack of a proper evaluation metric. This work presents partial solutions to both of these problems. We propose several techniques to stabilize training that allow us to train models that were previously untrainable. Moreover, our proposed evaluation metric (the Inception score) gives us a basis for comparing the quality of these models. We apply our techniques to the problem of semi-supervised learning, achieving state-of-the-art results on a number of different data sets in computer vision. The contributions made in this work are of a practical nature; we hope to develop a more rigorous theoretical understanding in future work.

[2] https://github.com/carpedm20/DCGAN-tensorflow

References

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, et al. Generative adversarial nets. In *NIPS*, 2014.

[2] Emily Denton, Soumith Chintala, Arthur Szlam, and Rob Fergus. Deep generative image models using a laplacian pyramid of adversarial networks. *arXiv preprint arXiv:1506.05751*, 2015.

[3] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[4] Ian J Goodfellow. On distinguishability criteria for estimating generative models. *arXiv preprint arXiv:1412.6515*, 2014.

[5] Daniel Jiwoong Im, Chris Dongjoo Kim, Hui Jiang, and Roland Memisevic. Generating images with recurrent adversarial networks. *arXiv preprint arXiv:1602.05110*, 2016.

[6] Donggeun Yoo, Namil Kim, Sunggyun Park, Anthony S Paek, and In So Kweon. Pixel-level domain transfer. *arXiv preprint arXiv:1603.07442*, 2016.

[7] Arthur Gretton, Olivier Bousquet, Alex Smola, and Bernhard Schölkopf. Measuring statistical dependence with hilbert-schmidt norms. In *Algorithmic learning theory*, pages 63–77. Springer, 2005.

[8] Kenji Fukumizu, Arthur Gretton, Xiaohai Sun, and Bernhard Schölkopf. Kernel measures of conditional dependence. In *NIPS*, volume 20, pages 489–496, 2007.

[9] Alex Smola, Arthur Gretton, Le Song, and Bernhard Schölkopf. A hilbert space embedding for distributions. In *Algorithmic learning theory*, pages 13–31. Springer, 2007.

[10] Yujia Li, Kevin Swersky, and Richard S. Zemel. Generative moment matching networks. *CoRR*, abs/1502.02761, 2015.

[11] Gintare Karolina Dziugaite, Daniel M Roy, and Zoubin Ghahramani. Training generative neural networks via maximum mean discrepancy optimization. *arXiv preprint arXiv:1505.03906*, 2015.

[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[13] Ilya Sutskever, Rafal Jozefowicz, Karol Gregor, et al. Towards principled unsupervised learning. *arXiv preprint arXiv:1511.06440*, 2015.

[14] Jost Tobias Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks. *arXiv preprint arXiv:1511.06390*, 2015.

[15] Augustus Odena. Semi-supervised learning with generative adversarial networks. *arXiv preprint arXiv:1606.01583*, 2016.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. 2016. MIT Press.

[17] George W Brown. Iterative solution of games by fictitious play. *Activity analysis of production and allocation*, 13(1):374–376, 1951.

[18] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the Inception Architecture for Computer Vision. *ArXiv e-prints*, December 2015.

[19] David Warde-Farley and Ian Goodfellow. Adversarial perturbations of deep neural networks. In Tamir Hazan, George Papandreou, and Daniel Tarlow, editors, *Perturbations, Optimization, and Statistics*, chapter 11. 2016. Book in preparation for MIT Press.

[20] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *arXiv preprint arXiv:1512.00567*, 2015.

[21] Tim Salimans and Diederik P Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. *arXiv preprint arXiv:1602.07868*, 2016.

[22] Diederik P Kingma, Shakir Mohamed, Danilo Jimenez Rezende, and Max Welling. Semi-supervised learning with deep generative models. In *Neural Information Processing Systems*, 2014.

[23] Takeru Miyato, Shin-ichi Maeda, Masanori Koyama, Ken Nakae, and Shin Ishii. Distributional smoothing by virtual adversarial examples. *arXiv preprint arXiv:1507.00677*, 2015.

[24] Lars Maaløe, Casper Kaae Sønderby, Søren Kaae Sønderby, and Ole Winther. Auxiliary deep generative models. *arXiv preprint arXiv:1602.05473*, 2016.

[25] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. Semi-supervised learning with ladder networks. In *Advances in Neural Information Processing Systems*, 2015.

[26] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, et al. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199*, 2013.

[27] Junbo Zhao, Michael Mathieu, Ross Goroshin, and Yann Lecun. Stacked what-where auto-encoders. *arXiv preprint arXiv:1506.02351*, 2015.

[28] Martín Abadi, Ashish Agarwal, Paul Barham, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.