UNIVERSITÄT
BERN

Generative Adversarial Networks

Paolo Favaro

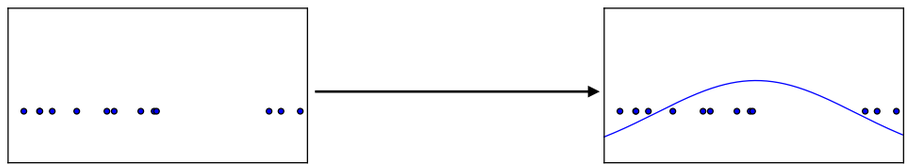
Contents

- Generative Adversarial Networks
  - Generative modeling, Principles of Adversarial Learning, Issues: Vanishing gradient and Mode Collapse
- Based on the tutorial paper
  - NIPS 2016 Tutorial: GAN by Goodfellow, 2016
- Other resources
  - How GAN and its Variants Work: An Overview of GAN by Hong, Hwang, You and Yoon, 2018

Note

Generative Modeling

- One approach is to learn an approximation of the probability density function (an explicit model)



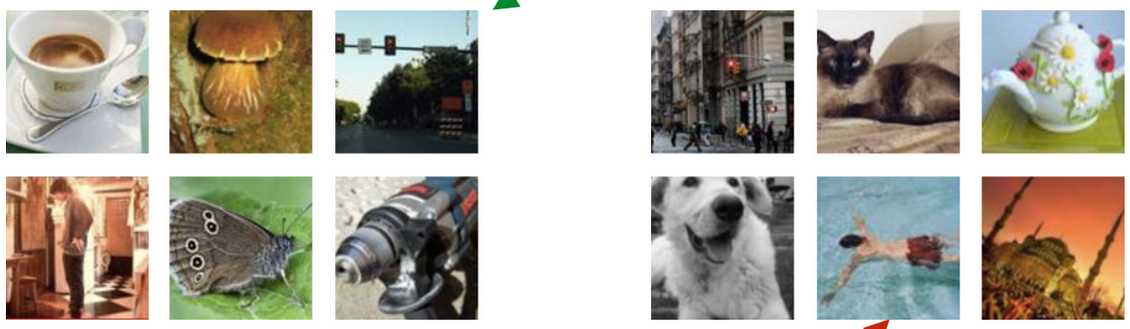
input samples

output distribution

We could build a function that is an approximation of the true data distribution.

Generative Modeling

- Another approach is to take **training samples**



and learn to generate new **synthetic samples***

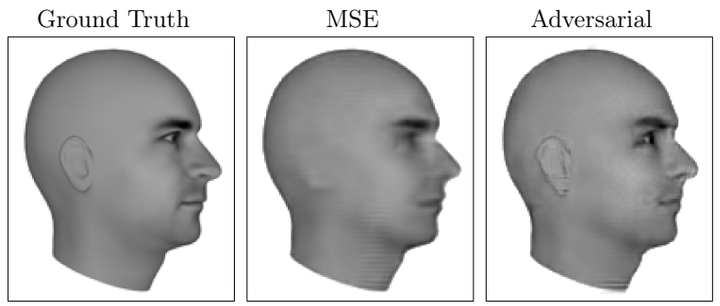
*These examples are also real images (used for illustration purposes)

Similarly to the VAEs, that we have already seen, we would like to build a function that is able to directly output valid samples.

Uses of Generative Models

- Part of representation learning
- Can be incorporated in other learning methods to impose data distribution constraints
  - The ultimate data augmentation method

GANs can deal with data that has a multi-modal distribution

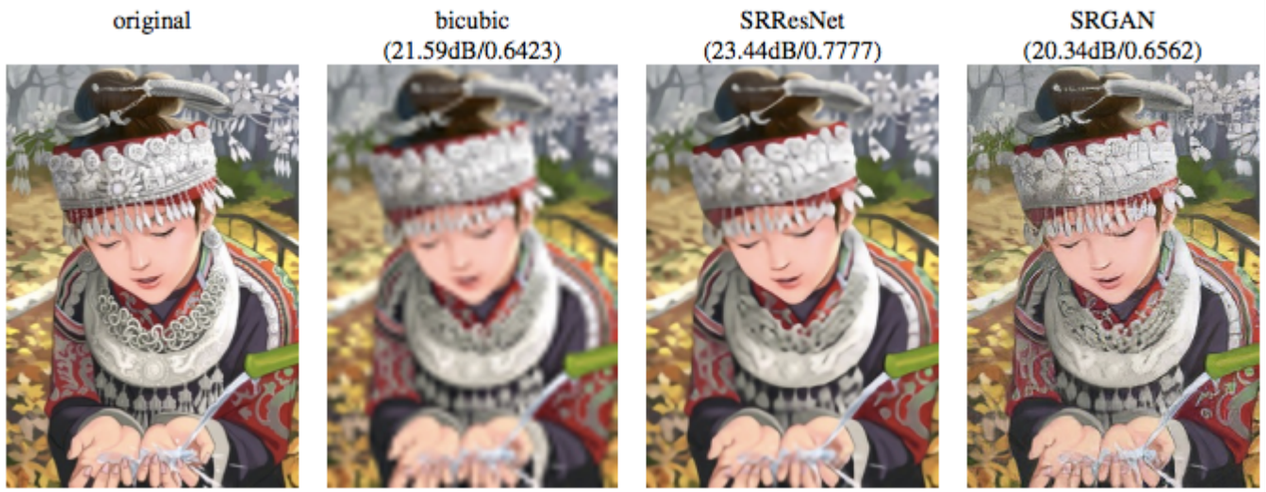


Next video frame prediction (Lotter et al 2016)

If we generate samples, we can use generative models for representation learning. In this field we look for a simple representation space that we can easily sample and for the transformation that maps this sample to a realistic image. The example shows video frame prediction with different losses (multi-modality is the key ingredient here).

Applications

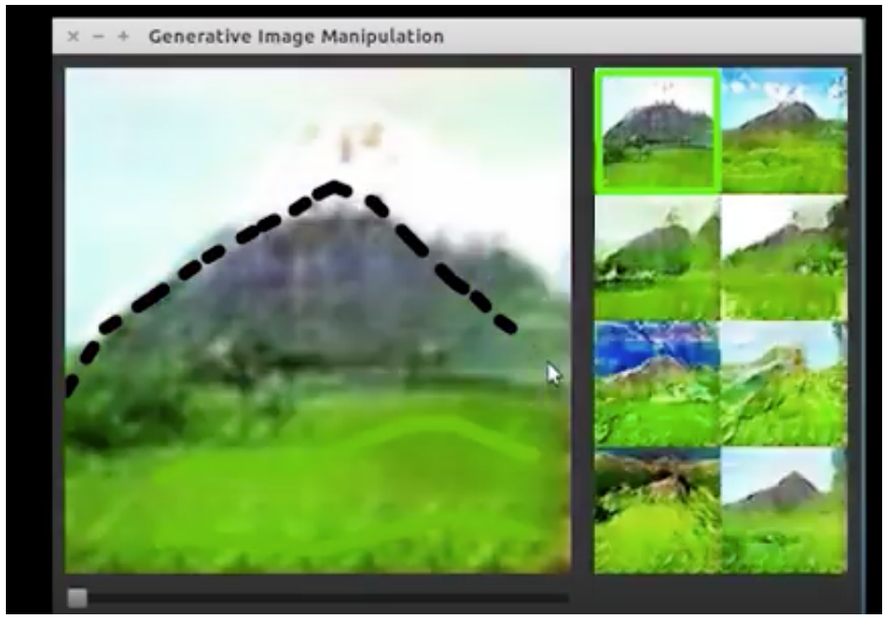
Image Superresolution



(Ledig et al 2016)

Note

Image Sketching



youtube link

(Zhu et al 2016)

Note

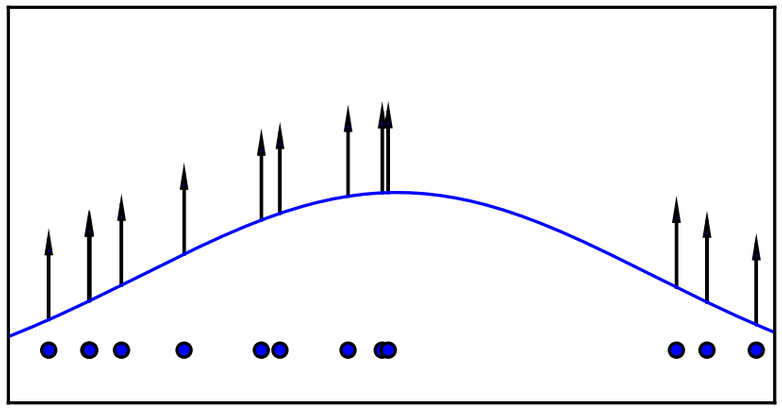
Image to Image Translations

(Isola et al 2016)

Note

Existing
Generative Models

Maximum Likelihood

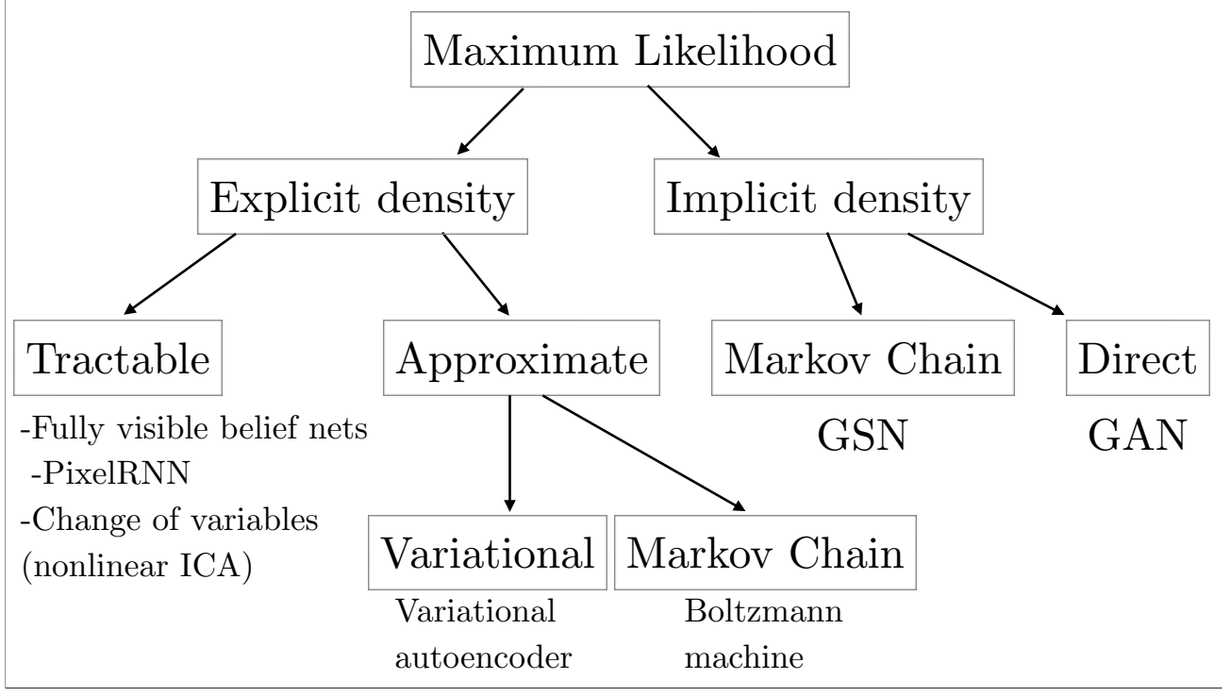


$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{x \sim p_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{\theta})$$

$$= \arg\min_{\boldsymbol{\theta}} D_{\text{KL}}(p_{\text{data}}(\boldsymbol{x}) \| p_{\text{model}}(\boldsymbol{x}; \boldsymbol{\theta}))$$

Let's consider only methods that solve Maximum Likelihood. Maximum likelihood can also be seen as matching the distribution of the data (p_data) to that produced by the model (p_model). D_KL is the Kullback-Leibler distance.

Overview of GM Methods



We can have an explicit model of the pdf or an implicit one. In the explicit case we can distinguish models that are built to have tractable training by restricting the distribution family and models that admit generic distributions but use numerically tractable approximations (VAE and BM).

Fully Visible Belief Nets

- Explicit formula based on chain rule:

$$p_{\text{model}}(\boldsymbol{x}) = p_{\text{model}}(x_1) \prod_{i=2}^{n} p_{\text{model}}(x_i \mid x_1, \ldots, x_{i-1})$$

- Disadvantages:
  - O(n) sample generation cost
  - Generation not controlled by a latent code

PixelCNN elephants
(van den Oord et al 2016)

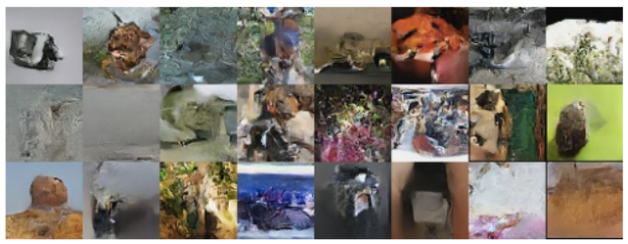
The fundamental idea is to use the probability chain rule. The problem is that the data generation is very slow (cannot be parallelized).

Change of Variables

$$y = g(x) \Rightarrow p_x(\boldsymbol{x}) = p_y(g(\boldsymbol{x})) \left| \det \left(\frac{\partial g(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) \right|$$

e.g. Nonlinear ICA (Hyvärinen 1999)

- Disadvantages:
  - Transformation must be invertible
  - Latent dimension must match visible dimension



64x64 ImageNet Samples
Real NVP (Dinh et al 2016)

The idea here is to use a simple distribution for y (p_y) and move all the complications to g (a deterministic mapping), which, however, must be differentiable and invertible.
The algorithm learns the transformation g.
Real NVP stands for real-valued non volume preserving.

Variational Autoencoder

$$\log p(\boldsymbol{x}) \geq \mathbb{E}_{\boldsymbol{z} \sim Q} \log p(\boldsymbol{x}|\boldsymbol{z}) - D_{\mathrm{KL}}(Q(\boldsymbol{z}|\boldsymbol{x})||p(\boldsymbol{z}))$$

- Disadvantages:
  - Not asymptotically consistent unless q is perfect
  - Samples tend to have lower quality

CIFAR-10 samples
(Kingma et al 2016)

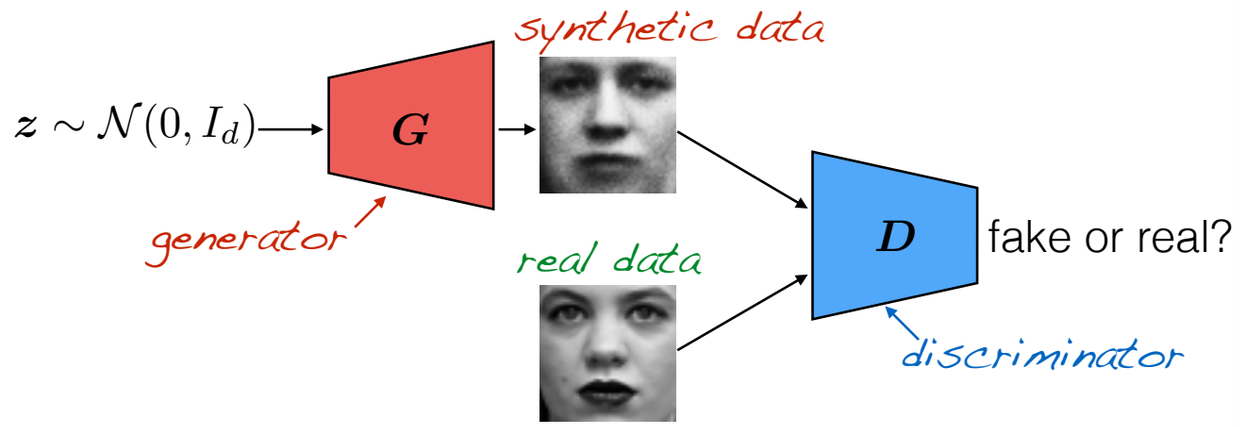
As we have already discussed, VAEs tend to generate blurry outputs because they use the Gaussian modeling assumption for the data domain.

Generative Adversarial Nets

- Can generate samples very efficiently
- Very flexible (designs of the generator and discriminators can be changed)
- At convergence it can learn the exact data distribution (if it converges correctly)
- The quality of the samples is currently better than in competing methods (some quality measures exist but it is still unclear how to measure quality)

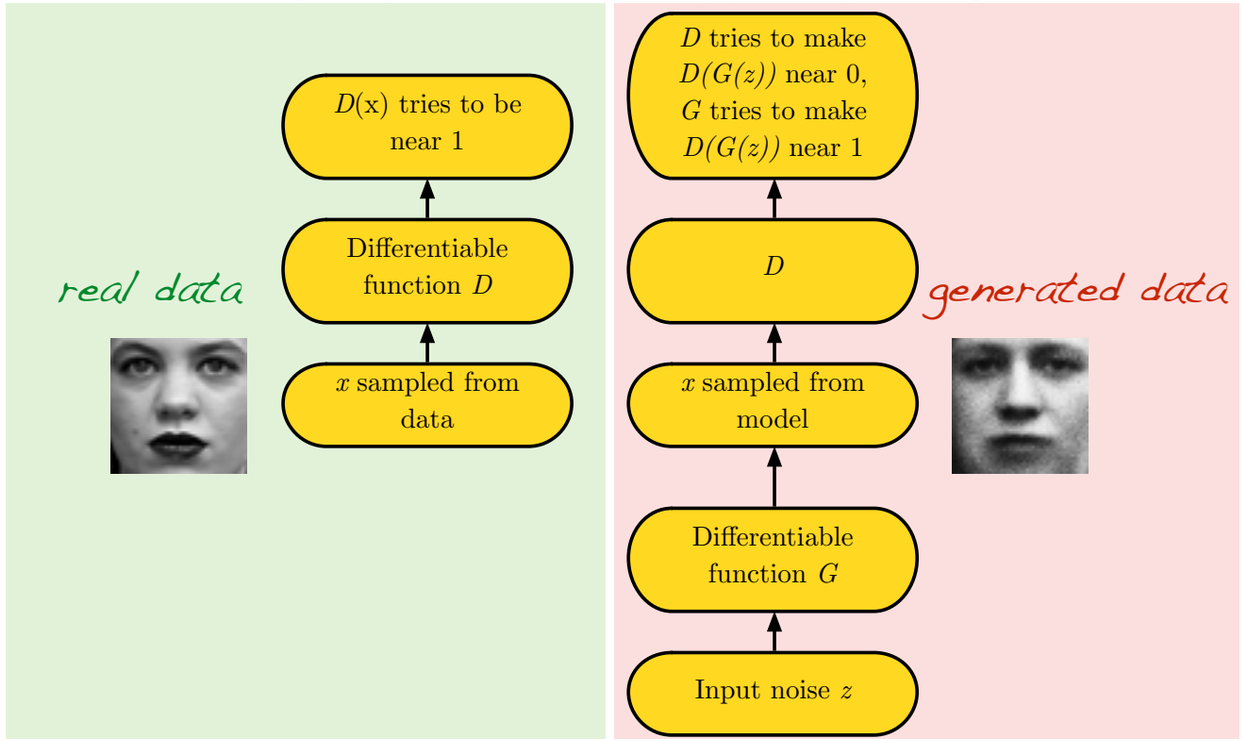
Also, VAEs cannot have discrete variables at the input to the generator, while GANs cannot have discrete variables at the output of the generator.
The Frechet Inception Distance seems to be the currently used measure. It takes a trained Inception network and applies it to both real images and synthetic images. The output should give a few object categories per image and span all categories across the whole dataset. In FID the distance between the means and covs of both sets (real and synth) are computed (assuming they have a Gaussian distribution).

The GAN Framework



A generator takes as input a random sample (Gaussian) and generates an (synthetic) image. A discriminator takes as input either a synthetic or a real image and tries to tell them apart.

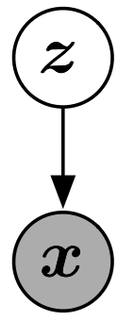
The GAN Framework



Note

Generator Network

$$\boldsymbol{x} = G(\boldsymbol{z}; \boldsymbol{\theta}^{(G)})$$

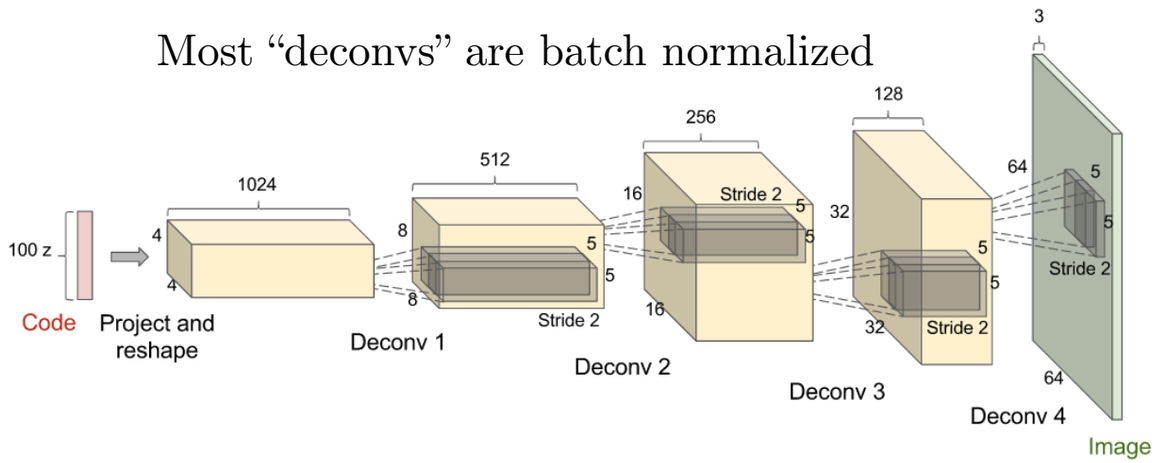


- Must be differentiable
- No invertibility requirement
- Trainable for any size of z
- Some guarantees require z to have same dimension as x
- Can make x conditionally Gaussian given z but need not do so

In some implementations noise is inserted also at intermediate layers. z is required to have the same dimension as x only when we want to be able to fully cover the space where x lives.

Example of a Generator: DCGAN Architecture

Most "deconvs" are batch normalized



(Radford et al 2015)

It uses fractionally-strided convolutions to upsample.

Discriminator Network

$$p(\boldsymbol{y} = \text{“real”}|\boldsymbol{x}) = D\left(\boldsymbol{x}; \boldsymbol{\theta}^{(D)}\right)$$

- Must be differentiable
- Trainable for any size of x
- Learns to classify images into “real” or “fake”
- Output is a binary probability, but can also be extended to multiple classes
- Can use an AlexNet-type network

Loss Function

$$\min_G \max_D \frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) + \frac{1}{2}\mathbb{E}_{\boldsymbol{z}\sim \mathcal{N}} \log\left(1 - D(G(\boldsymbol{z}))\right)$$

- G tries to minimize the cost
  - The minimum is achieved when D(x) = 0 and D(G(z)) = 1
    That is, when D thinks that x is fake and G(z) is real
- D tries to maximize the cost
  - The maximum is achieved when D(x) = 1 and D(G(z)) = 0
    That is, when D thinks that x is real and G(z) is fake

Note

Training

- Use SGD-like algorithm of choice (e.g., Adam) on two minibatches simultaneously:
  - A minibatch of training examples
  - A minibatch of generated samples
- Optional: run *k* steps of one player (Generator) for every step of the other player (Discriminator)

Goodfellow suggest that the best setting is simple SGD with one step each

A Minimax Game

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$
$$J^{(G)} = -J^{(D)}$$

- Equilibrium is a saddle point of the discriminator loss
- Resembles Jensen-Shannon divergence between data and model distribution

The Jensen-Shannon divergence is 1/2 D(p|q) + 1/2 D(q|p)
where D(p|q) = sum_i p_i log p_i / log q_i.

Variational Discriminator

- If we assume that the discriminator is a general function (not parametrized), we can compute the exact minimum of

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$

  by taking its variational derivative with respect to D

- First, notice that

$$\begin{aligned} J^{(D)} &= -\frac{1}{2}\int p_{\text{data}}(\boldsymbol{x}) \log D(\boldsymbol{x}) d\boldsymbol{x} - \frac{1}{2}\int p(\boldsymbol{z}) \log(1 - D(G(\boldsymbol{z}))) d\boldsymbol{z} \\ &= -\frac{1}{2}\int p_{\text{data}}(\boldsymbol{x}) \log D(\boldsymbol{x}) d\boldsymbol{x} - \frac{1}{2}\int p_{\text{model}}(\tilde{\boldsymbol{x}}) \log(1 - D(\tilde{\boldsymbol{x}})) d\tilde{\boldsymbol{x}} \end{aligned}$$

Variational Discriminator

$$J^{(D)} = -\frac{1}{2}\int p_{\text{data}}(\boldsymbol{x}) \log D(\boldsymbol{x}) d\boldsymbol{x} - \frac{1}{2}\int p(\boldsymbol{z}) \log(1 - D(G(\boldsymbol{z}))) d\boldsymbol{z}$$
$$= -\frac{1}{2}\int p_{\text{data}}(\boldsymbol{x}) \log D(\boldsymbol{x}) d\boldsymbol{x} - \frac{1}{2}\int p_{\text{model}}(\tilde{\boldsymbol{x}}) \log(1 - D(\tilde{\boldsymbol{x}})) d\tilde{\boldsymbol{x}}$$

- Now we can take the variational derivative in D

$$0 = \nabla_D J^{(D)} = -\frac{1}{2}\frac{p_{\text{data}}(\boldsymbol{x})}{D(\boldsymbol{x})} + \frac{1}{2}\frac{p_{\text{model}}(\boldsymbol{x})}{1 - D(\boldsymbol{x})}$$

and obtain

$$D(\boldsymbol{x}) = \frac{p_{\text{data}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{\text{model}}(\boldsymbol{x})}$$

key difference with other methods

If p_model matches p_data then D is fully confused and gives 1/2 all the time. If p_model does not match p_data then D can be confident (1) in the classification.

Optimal Generator

- If we now plug in the optimal discriminator and look for the optimal generator we need to solve

$$\min_G \frac{1}{2}\mathbb{E}_{p_{\text{data}}} \log \frac{p_{\text{data}}}{p_{\text{data}} + p_{\text{model}}} + \frac{1}{2}\mathbb{E}_{p_{\text{model}}} \log \frac{p_{\text{model}}}{p_{\text{data}} + p_{\text{model}}}$$

which is equivalent to solving

$$\min_G \frac{1}{2}\mathbb{E}_{p_{\text{data}}} \log \frac{p_{\text{data}}}{(p_{\text{data}} + p_{\text{model}})/2} + \frac{1}{2}\mathbb{E}_{p_{\text{model}}} \log \frac{p_{\text{model}}}{(p_{\text{data}} + p_{\text{model}})/2} - \log 2$$

or

$$\min_G \frac{1}{2} D_{\text{KL}} \left(p_{\text{data}} \middle\| \frac{p_{\text{data}} + p_{\text{model}}}{2} \right) + \frac{1}{2} D_{\text{KL}} \left(p_{\text{model}} \middle\| \frac{p_{\text{data}} + p_{\text{model}}}{2} \right) - \log 2$$

Note

Optimal Generator

- The generator that leads to the minimum of

$$\min_G \frac{1}{2} D_{\mathrm{KL}} \left(p_{\mathrm{data}} \middle\| \frac{p_{\mathrm{data}} + p_{\mathrm{model}}}{2} \right) + \frac{1}{2} D_{\mathrm{KL}} \left(p_{\mathrm{model}} \middle\| \frac{p_{\mathrm{data}} + p_{\mathrm{model}}}{2} \right) - \log 2$$

is the one yielding

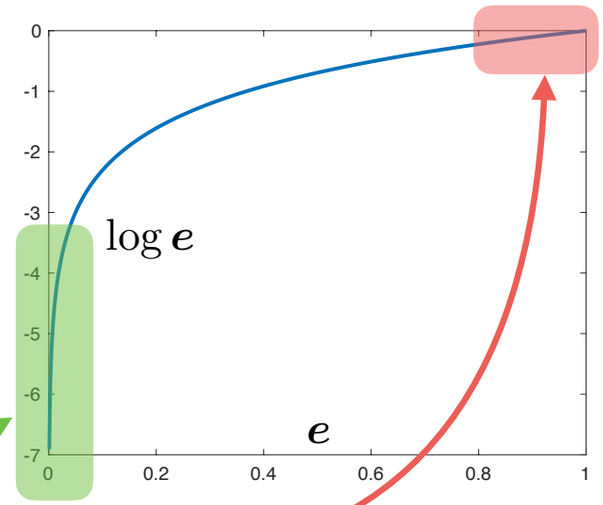
$$p_{\mathrm{model}} = p_{\mathrm{data}}$$

because both Kullback-Leibler distances are at zero (which is their minimum value)

Note

The Saturation Problem

- When the discriminator converges
  - the gradient of the first term $\log D(\boldsymbol{x})$ is large
  - the gradient of the second term $\log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$ is small → stops G from learning



Note

Heuristic Cost

- To avoid the vanishing gradient of G optimize instead

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log D\left(G(\boldsymbol{z})\right)$$

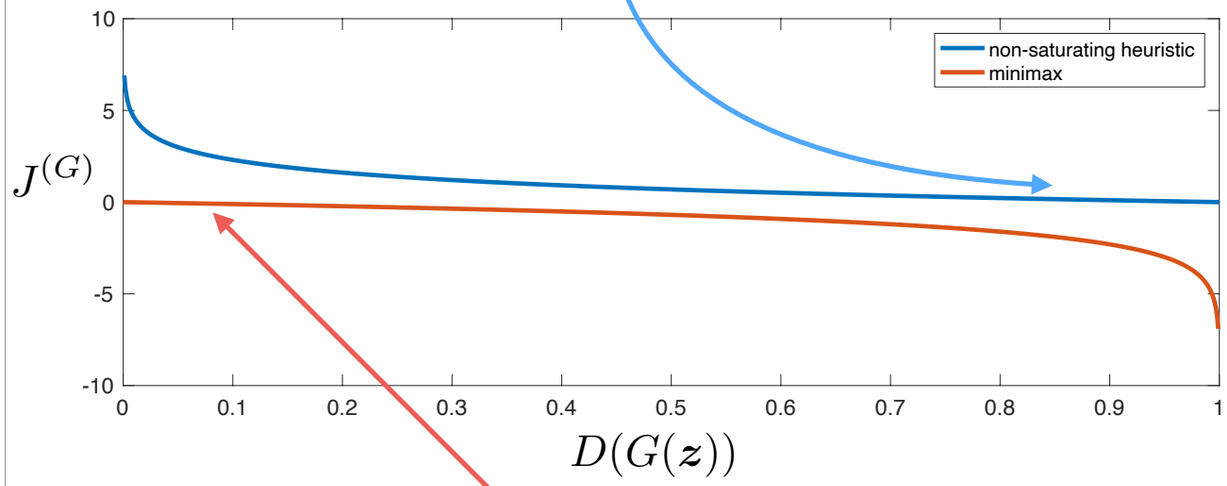
  obtained by flipping the two probabilities in the loss

- This moves the cost in G also to the high gradient region (when D converges)

Note

Heuristic vs Mimimax

With the heuristic the cost is flat when D considers G samples to be real (then G does not need to change)



With minimax the cost is flat when D considers G samples to be fake and this gives no gradient to G to further improve

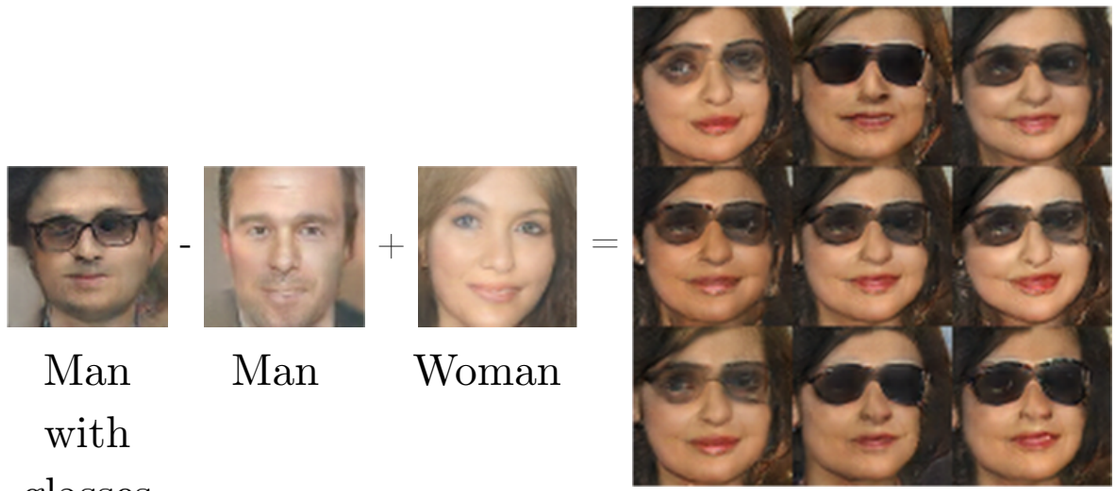
Note

Examples with DCGAN

(Radford et al 2015)

Note

Representation Linearity



(Radford et al, 2015)

Note

Tricks to Train GANs

- Using Labels
- Smoothing Labels
- Batch Normalization
- Balancing G and D

Note

Using Labels

- Learning a conditional model $p(y\,|x)$ often gives much better samples from all classes than learning $p(x)$ does (Denton et al 2015)
- Even just learning $p(y,\,x)$ makes samples from $p(x)$ look much better to a human observer (Salimans et al 2016)
- Note: this defines three categories of models (no labels, trained with labels, generating condition on labels) that should not be compared directly to each other

In Denton one feeds the category as input. In Salimans one asks the discriminator to classify images too.

One-Sided Label Smoothing

- Default discriminator cost:

```
cross_entropy(1., discriminator(data))
+ cross_entropy(0., discriminator(samples))
```

- One-sided label smoothed cost (Salimans et al 2016):

```
cross_entropy(.9, discriminator(data))
+ cross_entropy(0., discriminator(samples))
```

Note

No Negative Label Smoothing

cross_entropy(1.-alpha, discriminator(data))
+ cross_entropy(beta, discriminator(samples))

Reinforces current generator behavior

$$D(\boldsymbol{x}) = \frac{(1-\alpha)p_{\text{data}}(\boldsymbol{x}) + \beta p_{\text{model}}(\boldsymbol{x})}{p_{\text{data}}(\boldsymbol{x}) + p_{\text{model}}(\boldsymbol{x})}$$

optimal discriminator

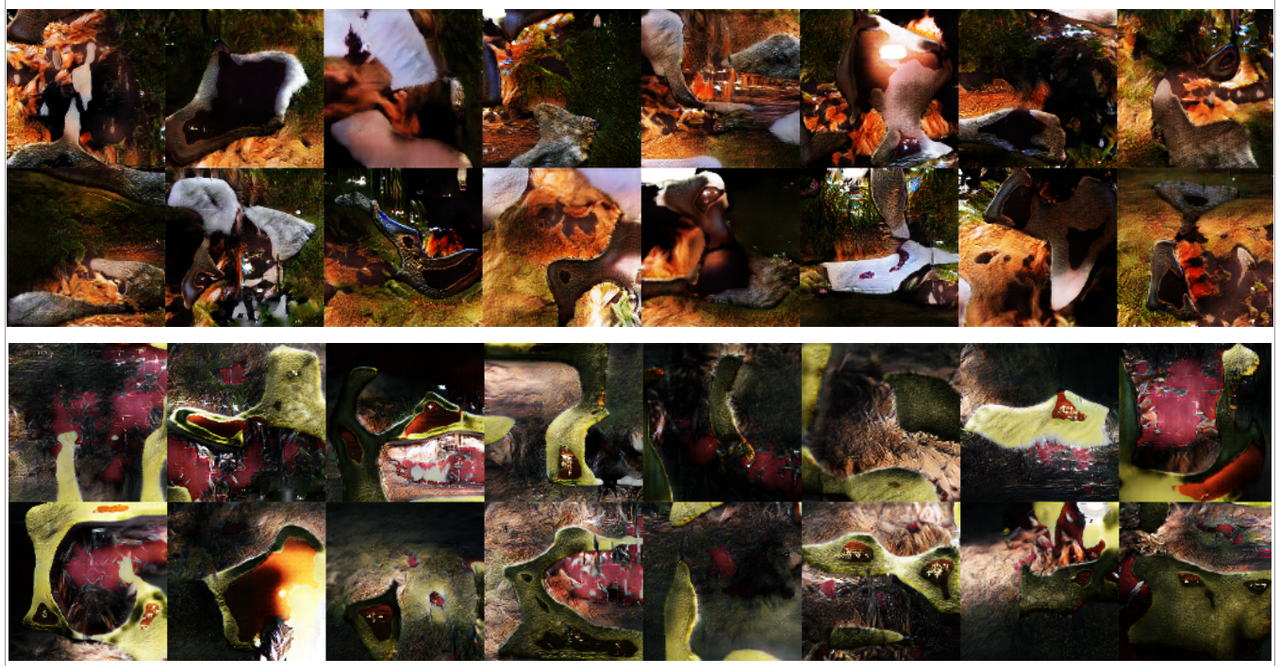
When \beta>0 then D converges to a mix of the two distributions that encourages G to keep generating its own bad examples.

Label Smoothing

- Good regularizer (Szegedy et al 2015)
- Does not reduce classification accuracy, only confidence
- Benefits specific to GANs:
  - Prevents discriminator from giving very large gradient signal to generator
  - Prevents linear extrapolation to encourage extreme samples (adversarially constructed)

Label smoothing works as a regularizer because it amounts to assuming noise in the labels.

Batch Normalization in G



Samples in the same minibatch have a strong correlation (although they should be independent) due to batch normalization.

Alternatives to Batch Normalization

- **Reference** and **virtual** batch normalization (apply computations of mean and std to a reference group or a mix of the reference group and the current batch)
- **Instance** normalization (batch norm is computed per sample within the minibatch)

Note

Balancing G and D

- Usually the discriminator “wins”: This is a good thing—the theoretical justifications are based on assuming D is perfect
- Usually D is bigger and deeper than G
- Sometimes run D more often than G. **Mixed results.**
- Do not try to limit D to avoid making it “too smart”
  - Use non-saturating cost
  - Use label smoothing

Note

Additional Material

- https://github.com/soumith/ganhacks

Note

Limitations

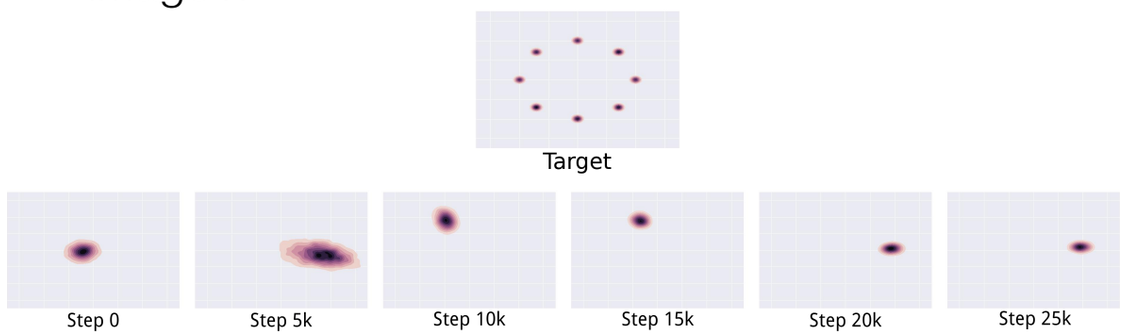
Non-Convergence

- Exploiting convexity in function space, GAN training is theoretically guaranteed to converge if we can modify the density functions directly, but:
  - Instead, we modify parameters of functions G (sample generation function) and D (density ratio), not densities
  - We represent G and D as highly non-linear parametric functions (kills convexity)
- “Oscillation”: can train for a very long time, generating very many different categories of samples, without clearly generating better samples
- Mode collapse: most severe form of non-convergence

Note

Mode Collapse

- Although our target is multimodal, the generator converges to only one or only some of the modes
- The generator learns to map multiple z to the same image x

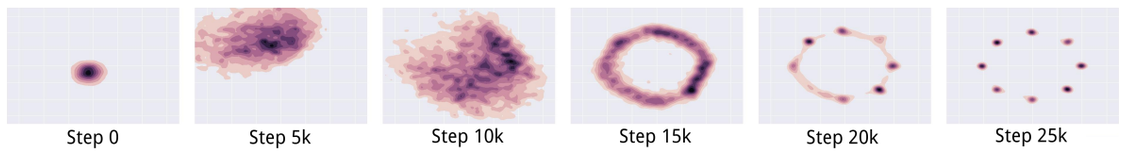


(Metz et al 2016)

Note

Unrolled GANs

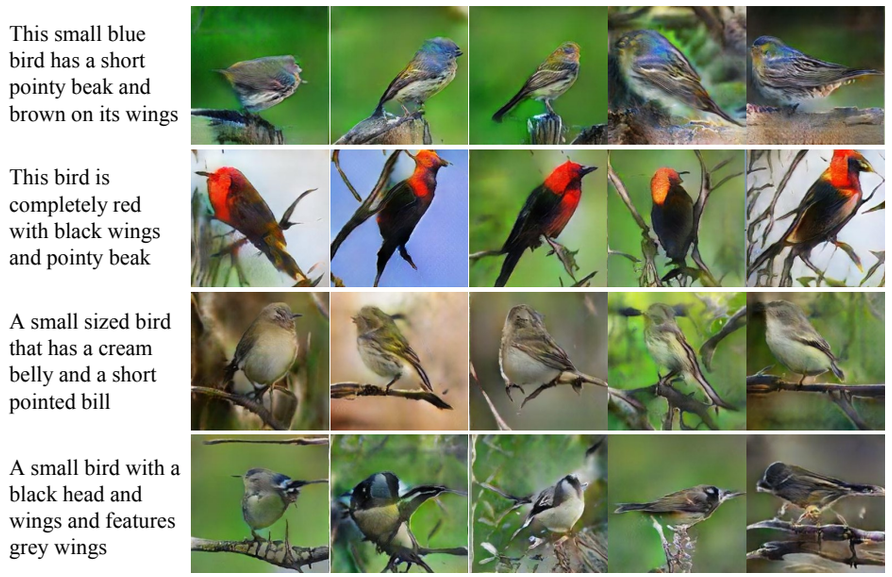
- Optimization in G should be done on the max in D
- Thus, backprop through k updates of the discriminator
- This seems to prevent mode collapse



(Metz et al 2016)

Note

StackGANs



(Zhang et al 2016)

They achieve more diversity by using a pyramid of GANs (small to large) and by conditioning GANs on the previous smaller scale results (from the GAN at the lower scale).

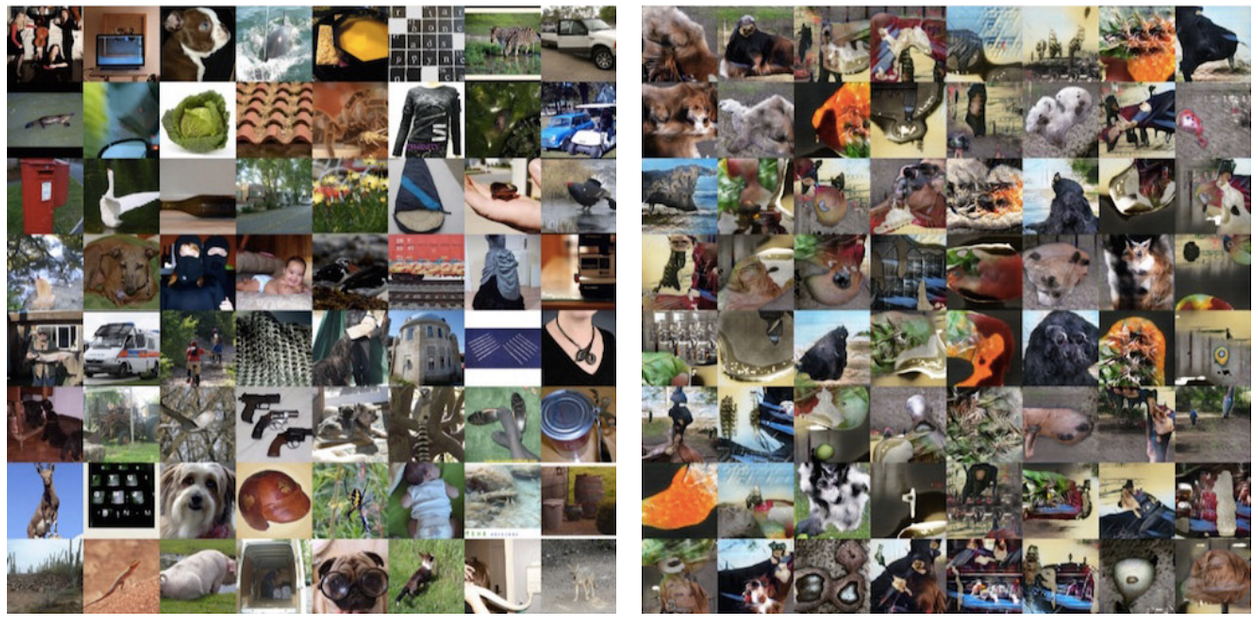
Minibatch GANs

- Add a loss term about matching features from the discriminator (at different layers) on real and generated samples
- The discriminator can detect mode collapse if it looks at multiple samples at once: With mode collapse the variability is more limited than in the minibatch of real images
- Nearest-neighbor style features detect if a minibatch contains samples that are too similar to each other

(Salimans et al 2016)

Note

Minibatch GAN on ImageNet



(Salimans et al 2016)

Note

Cherry-Picked Examples

Note

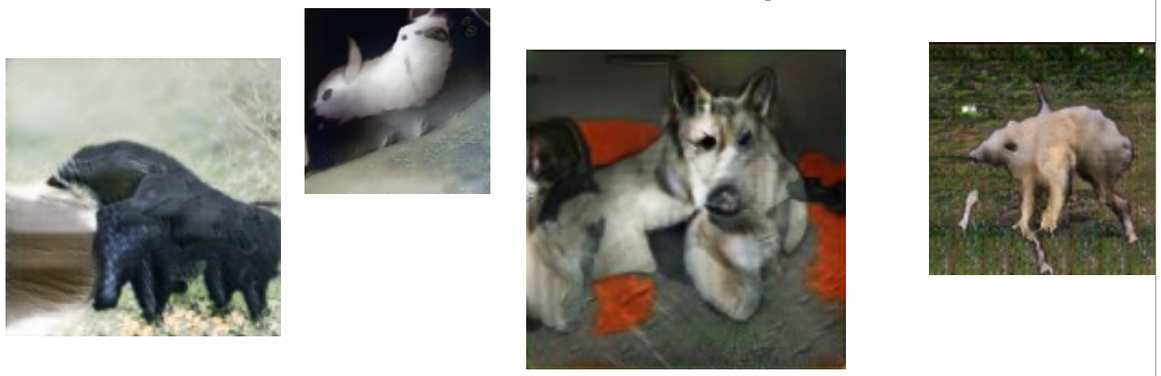
Problems with Counting






Note

Problems with Perspective



Note

Problems with Structure

Note