Introduction to Generative Adversarial Network

Qiang Luo
hzluoqiang@corp.netease.com
luoq08@gmail.com

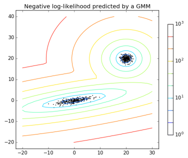
April 14, 2017

Outline

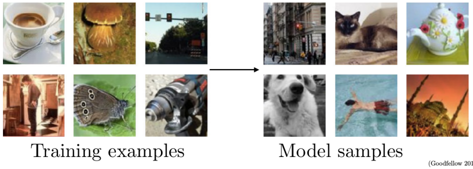
Introduction

Generative Model

- Supervised Learning: discriminative vs generative model $P(y|x)$ or $f(x)$ vs $P(x, y)$
- Unsupervised Learning: learn the true data distribution $P_r(x)$
    - explicitly learn the distribution

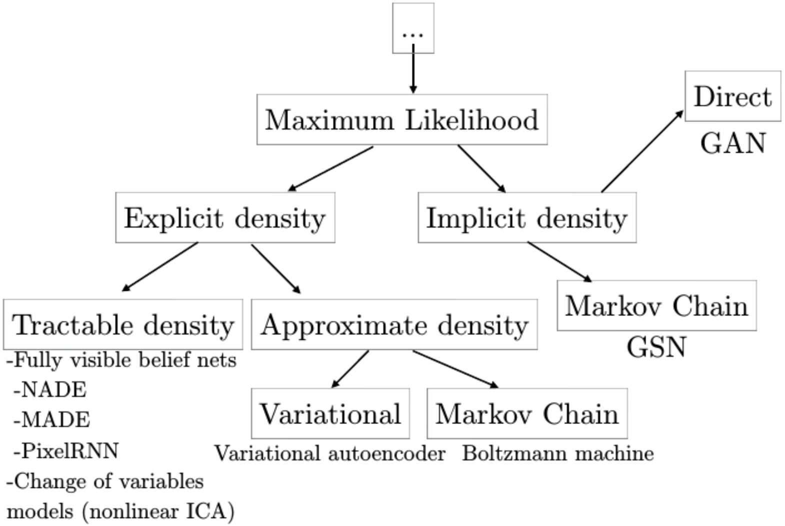


    - hard to define for natural data
    - generate sample directly



(Goodfellow 2016)

Taxonomy of Generative Models



Application

Super Resolution

Ledig et al 2016

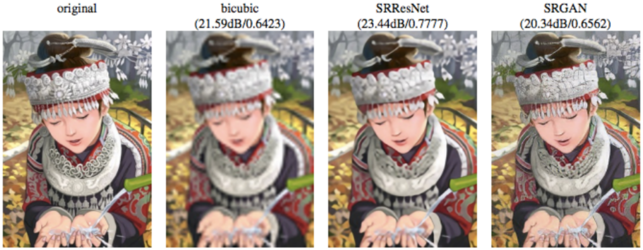
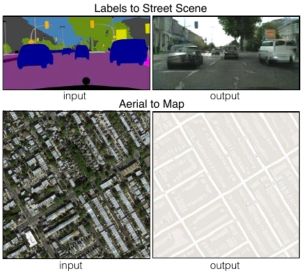
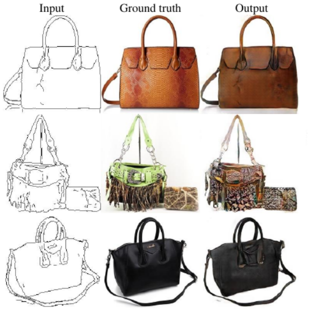


Image to Image Translation





Isola et al 2016

Text to Image Synthesis

Generative Adversarial Text-to-Image Synthesis, Reed et al 2016

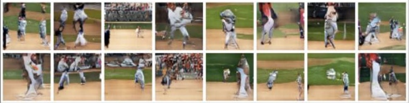
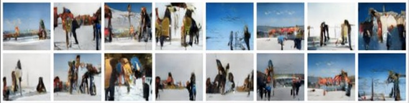
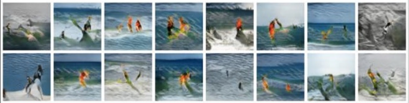
Caption	**Image**
a pitcher is about to throw the ball to the batter	
a group of people on skis stand in the snow	
a man in a wet suit riding a surfboard on a wave	

Image Generation

Use DCGAN

`https://www.facebook.com/soumith/posts/10154442598946002`

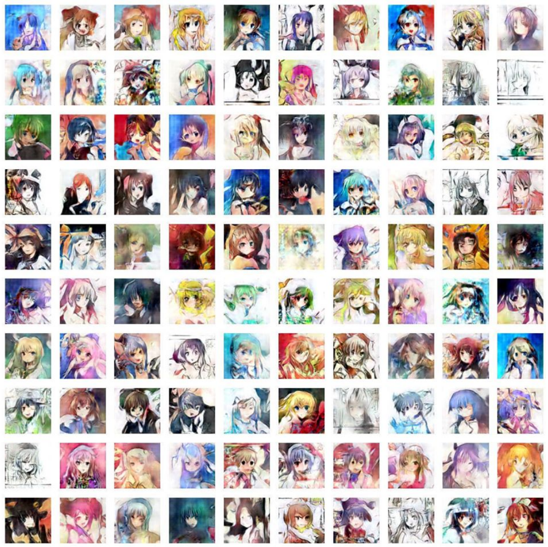
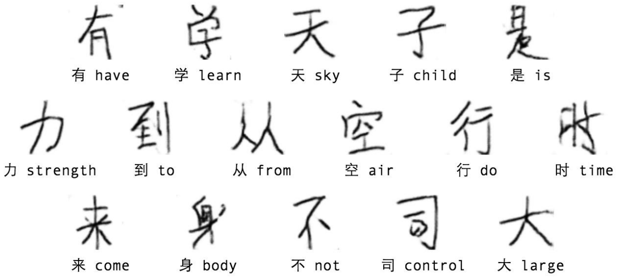


Image Generation

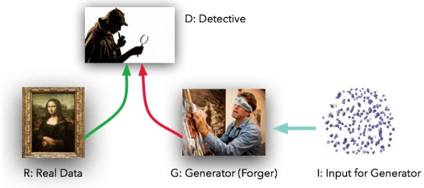
Image Generation

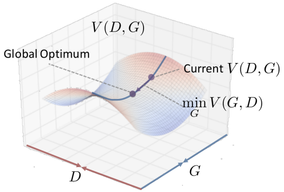


GAN

GAN

- *Generative adversarial nets (2014), I. Goodfellow et al.*
- min-max game for D and G





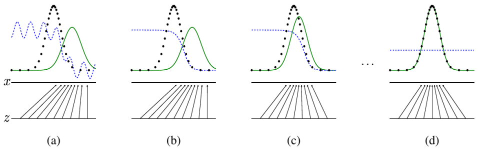
$$\min_G \max_D V(D,G) = E_{x\sim P_r(x)} \log(D(x)) + E_{z\in P_z(z)} \log(1-D(G(z)))$$

- Find the Nash Equilibrium
- Fixed G, D solve classification with log loss
- Fixed D, G minimize $\log(1-D(G(z)))$

Algorithm

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

```
for number of training iterations do
    for $k$ steps do
        • Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
        • Sample minibatch of $m$ examples $\{x^{(1)}, \dots, x^{(m)}\}$ from data generating distribution $p_{\text{data}}(x)$.
        • Update the discriminator by ascending its stochastic gradient:
            $$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(x^{(i)}\right) + \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right)\right].$$
    end for
    • Sample minibatch of $m$ noise samples $\{z^{(1)}, \dots, z^{(m)}\}$ from noise prior $p_g(z)$.
    • Update the generator by descending its stochastic gradient:
        $$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(z^{(i)}\right)\right)\right).$$
end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.
```



Global minimum

- Fixed G, optimal D is

$$D^*(x) = \frac{P_r(x)}{P_r(x) + P_g(x)}$$

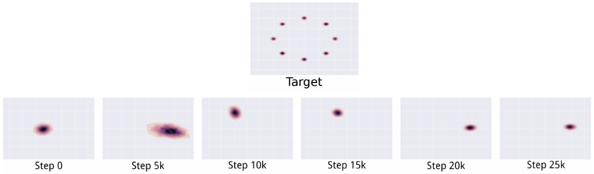
- With above solution, G minimize $E_{x \sim P_r(x)} \log \frac{P_r(x)}{P_r(x)+P_g(x)} + E_{x \sim P_g(x)} \log \frac{P_g(x)}{P_r(x)+P_g(x)} =$

$$2\mathrm{KS}(P_r, P_g) - \log(4)$$

- global minimum when $P_g = P_r$ with value of $-\log(4)$
- convergence in practice?

Problem of training

- In practice, we modify G (sample generation function) and D (density ratio), not densities
- We represent G and D as highly non-convex parametric functions
- "Oscillation": can train for a very long time, generating very many different categories of samples, without clearly generating better samples
- *Mode collapse*: generated data is not diverse



(Metz et al 2016)

(Goodfellow 2016)

Some tricks

- minimize $-\log(D(G(z)))$ instead of $\log(1 - D(G(z)))$ to prevent saturation when G is poor and D is confident
- D must be synchronized well with G during training
- in particular, G must not be trained too much without updating D to prevent mode collapse

Result

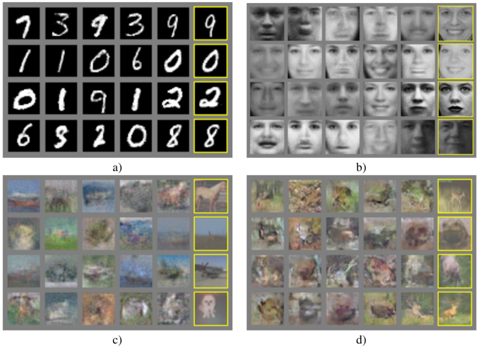


Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples are fair random draws, not cherry-picked. Unlike most other visualizations of deep generative models, these images show actual samples from the model distributions, not conditional means given samples of hidden units. Moreover, these samples are uncorrelated because the sampling process does not depend on Markov chain mixing. a) MNIST b) TFD c) CIFAR-10 (fully connected model) d) CIFAR-10 (convolutional discriminator and "deconvolutional" generator)

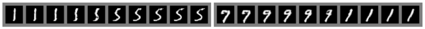


Figure 3: Digits obtained by linearly interpolating between coordinates in z space of the full model.

DCGAN

Contribution

- *Unsupervised representation learning with deep convolutional generative adversarial networks (2015), A. Radford et al.*
- GAN for image generation
- identified a family of architectures that resulted in stable training across a range of datasets
- use feature from D for classification
- arithmetic operation of latent vector

Architecture Guideline

- Replace any pooling layers with strided convolutions – this allows the network to learn its own spatial downsampling.
- Remove any fully connected hidden layers on top of convolutional features. The authors found that connecting the highest convolutional features to the input and output respectively of the generator and discriminator worked well.
- Use batch normalization, which stabilizes learning by normalizing the input to each unit to have zero mean and unit variance.
- Use ReLU activation for all generator layers, except for the final output layer where tanh works better Use Leaky ReLU activation for all discriminator layers. (We can update that advice to PReLUs now)

Architecture

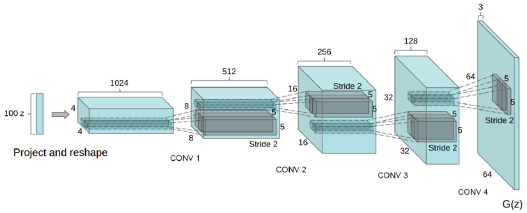


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions) then convert this high level representation into a 64×64 pixel image. Notably, no fully connected or pooling layers are used.

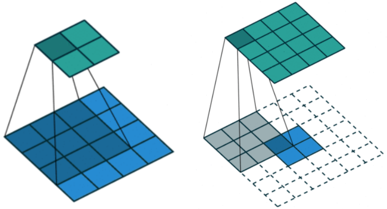
- some hyper parameter tuning
- trained on three datasets, Large-scale Scene Understanding (LSUN) (Yu et al., 2015), Imagenet-1k and a newly assembled Faces datase

Convolution and Transposed Convolution

- `https://github.com/vdumoulin/conv_arithmetic`
- Convolution with kernel 3×3, no padding, stride 1 for 4×4 input produce 2×2 output
- represent the transformation in matrix 4×9 matrix C

$$\begin{pmatrix} w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 & 0 \\ 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} & 0 \\ 0 & 0 & 0 & 0 & 0 & w_{0,0} & w_{0,1} & w_{0,2} & 0 & w_{1,0} & w_{1,1} & w_{1,2} & 0 & w_{2,0} & w_{2,1} & w_{2,2} \end{pmatrix}$$

- in forward pass $y = Cx$, in backward pass $dx = C^T dx$
- the transform defined by C^T is called *transposed convolution*, or *fractionally strided convolution*, ~~*deconvolution*~~
- It's another convolution with kernel 3×3, pad 2, stride 1



Generated Sample

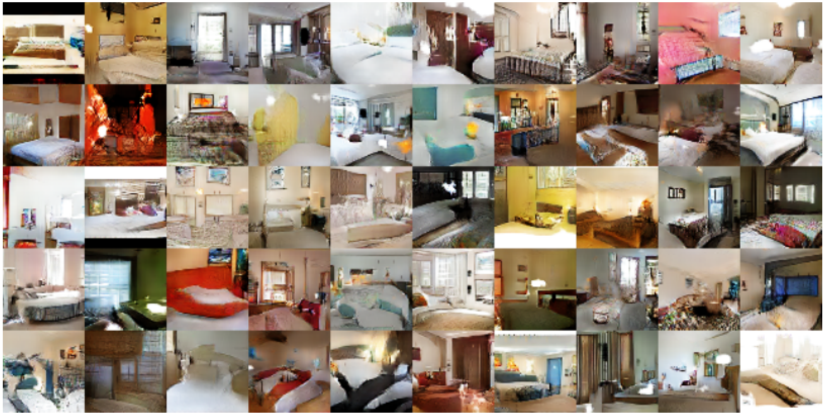
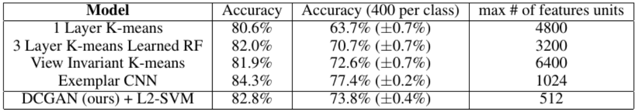


Figure 3: Generated bedrooms after five epochs of training. There appears to be evidence of visual under-fitting via repeated noise textures across multiple samples such as the base boards of some of the beds.

Usage in classification

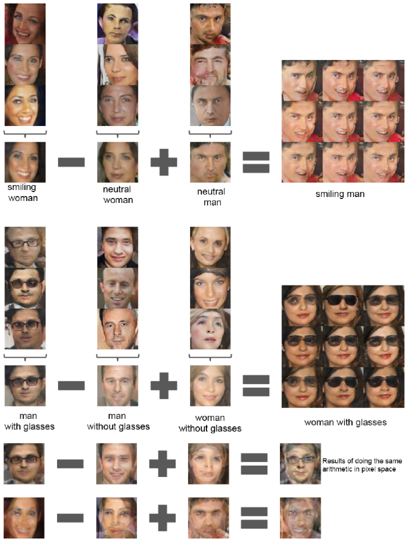
To evaluate the quality of the representations learned by DCGANs for supervised tasks, we train on Imagenet-1k and then use the discriminator's convolutional features from all layers, maxpooling each layers representation to produce a 4 $\times$ 4 spatial grid. These features are then flattened and concatenated to form a 28672 dimensional vector and a regularized linear L2-SVM classifier is trained on top of them.

Table 1: CIFAR-10 classification results using our pre-trained model. Our DCGAN is not pre-trained on CIFAR-10, but on Imagenet-1k, and the features are used to classify CIFAR-10 images.



Model	Accuracy	Accuracy (400 per class)	max # of features units
1 Layer K-means	80.6%	63.7% ($\pm$0.7%)	4800
3 Layer K-means Learned RF	82.0%	70.7% ($\pm$0.7%)	3200
View Invariant K-means	81.9%	72.6% ($\pm$0.7%)	6400
Exemplar CNN	84.3%	77.4% ($\pm$0.2%)	1024
DCGAN (ours) + L2-SVM	82.8%	73.8% ($\pm$0.4%)	512

Vector Arithmetic on face samples



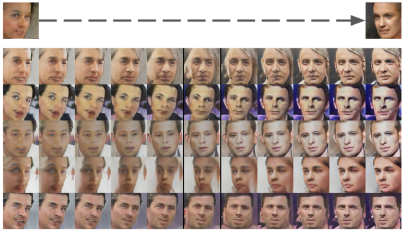
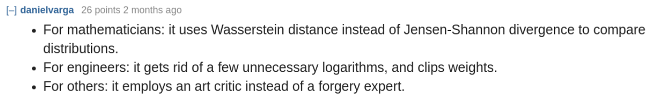


Figure 8: A "turn" vector was created from four averaged samples of faces looking left vs looking right. By adding interpolations along this axis to random samples we were able to reliably transform their pose.

W-GAN

Contribution

- *Towards principled methods for training generative adversarial networks(2017), M. Arjovsky et al*
- *Wasserstein GAN (2017), M. Arjovsky et al*
- find the reason why GAN training is hard
- algorithm with good theory and works
- no more mode collapse?
- no more architecture design
- sensible loss metric for training



[–] **danielvarga** 26 points 2 months ago

- For mathematicians: it uses Wasserstein distance instead of Jensen-Shannon divergence to compare distributions.
- For engineers: it gets rid of a few unnecessary logarithms, and clips weights.
- For others: it employs an art critic instead of a forgery expert.

Why is GAN hardy to train?

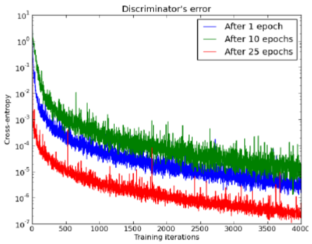
- MLE maximize $\frac{1}{m}\sum_{i=1}^{m}\log(P_g(x_i))$, which is minimize $\mathrm{KL}(P_r||P_g)$

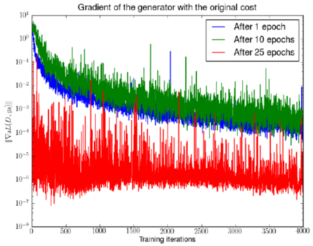
$$\mathrm{KL}(P_r, P_g) = \int P_r(x)\log\frac{P_r(x)}{P_g(x)}\mathrm{d}x$$

- GAN minimize $\mathrm{JS}(P_r, P_g)$(original, minimize $\log(1 - D(G(x)))$ for G)
- if $P_r(x) \gg P_g(x)$(mode dropping) KL goes to inf; if $P_r(x) \ll P_g(x)$(generate bad sample) KL goes to zero
- *Manifold hypothesis*: Natural data forms lower dimensional structures (manifolds) in the embedding space
- natural image embeded in space of all possible image $R^{3\times64\times64}$; g_θ map R^{100} to $R^{3\times64\times64}$
- support for P_r and P_g almost never intersect; $\mathrm{JS}(P_r, P_g) = \log 2$, $\mathrm{KL}(P_r, P_g) = \infty$, $\mathrm{KL}(P_g, P_r) = \infty$
- If D goes to optimal, the loss for G is KS and gradient from D will go to zero

Evidence from DCGAN training

- when D is near optimal, we can not train G
- when D is far from optimal, it's bad
- the coordinance is hard



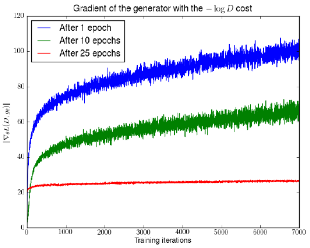


Why is GAN hard to train?

- usually $-\log(D(G(x)))$ is optimized for G
- in this case the loss is

$$\mathrm{KL}(P_g||P_r) - 2\mathrm{JS}(P_g||P_r)$$

- this loss(when stably trained) penalize generating bad sample and tolerate mode loss
- in this case the gradients for G is unstable



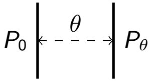
Wasserstein metric

- *Wasserstein metric* (transportation metric, earth mover's distance)

$$W(P, Q) = \inf_{\gamma \in \Gamma} E_{x,y \sim \gamma}(\|x - y\|) = \inf_{\gamma \in \Gamma} \int_{X,Y} \|x - y\| \gamma(x, y) \mathrm{d}x \mathrm{d}y$$

  where Γ is the joint distribution on (x, y) with marginal distribution P, Q
- $\gamma(x, y)$ is the quantity we transport from x to y, the marginal requirement makes sure P is transported to Q

Uniform distributions on parallel lines



  - $\mathrm{KL}(P_0, P_\theta) = +\infty$ if $\theta \neq 0$ else 0
  - $\mathrm{JS}(P_0, P_\theta) = \log(2)$ if $\theta \neq 0$ else 0
  - $W(P_0, P_\theta) = |\theta|$
- For P_r and P_θ generate from $g_\theta(z)$, $W(P_r, P_\theta)$ is good(continuous, differentiable, ...)
- sequence converges in KL, JS if it converges in W

Use Wasserstein metric

- a result from *Kantorovich-Rubinstein duality*

$$W(P, Q) = \frac{1}{K} \sup_{\|f\|_L \leqslant K} E_{x \in P} f(x) - E_{x \in Q} f(x)$$

  where $\|f\|_L \leqslant K$ means $\|f(x) - f(y)\| \leqslant K\|x - y\|, \forall x, y$ (K-Lipschitz functions)

- whose parameter w clamped to $[-c, c]$

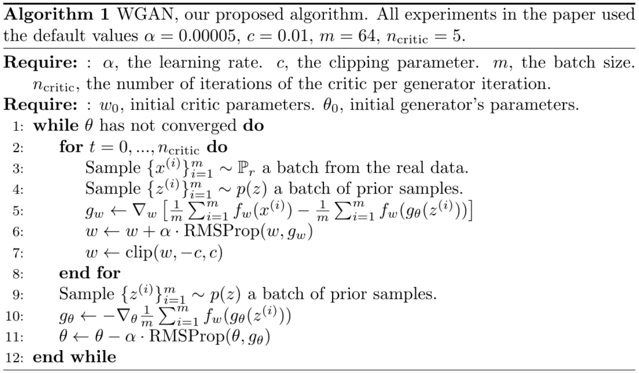
$$\min_{\theta} \max_{w} E_{x \sim P_r}(f_w(x)) - E_{z \sim P_z}(f_w(g_\theta(z)))$$

  The original GAN is

$$\min_{G} \max_{D} V(D, G) = E_{x \sim P_r} \log(D(x)) + E_{z \in P_z} \log(1 - D(G(z)))$$

Algorithm

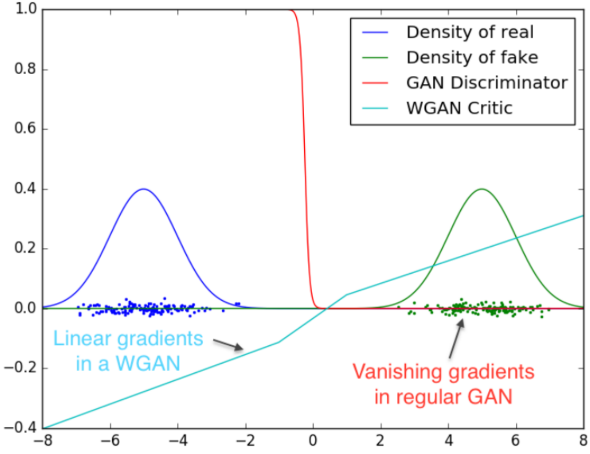
WGAN training becomes unstable at times when one uses a momentum based optimizer such as Adam, use momentum free optimizer



Algorithm 1 WGAN, our proposed algorithm. All experiments in the paper used the default values $\alpha = 0.00005$, $c = 0.01$, $m = 64$, $n_{\text{critic}} = 5$.

```
Require: : $\alpha$, the learning rate. $c$, the clipping parameter. $m$, the batch size.
    $n_{\text{critic}}$, the number of iterations of the critic per generator iteration.
Require: : $w_0$, initial critic parameters. $\theta_0$, initial generator's parameters.
 1: while $\theta$ has not converged do
 2:     for $t = 0, ..., n_{\text{critic}}$ do
 3:         Sample $\{x^{(i)}\}_{i=1}^m \sim \mathbb{P}_r$ a batch from the real data.
 4:         Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
 5:         $g_w \leftarrow \nabla_w \left[\frac{1}{m}\sum_{i=1}^m f_w(x^{(i)}) - \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))\right]$
 6:         $w \leftarrow w + \alpha \cdot \text{RMSProp}(w, g_w)$
 7:         $w \leftarrow \text{clip}(w, -c, c)$
 8:     end for
 9:     Sample $\{z^{(i)}\}_{i=1}^m \sim p(z)$ a batch of prior samples.
10:     $g_\theta \leftarrow -\nabla_\theta \frac{1}{m}\sum_{i=1}^m f_w(g_\theta(z^{(i)}))$
11:     $\theta \leftarrow \theta - \alpha \cdot \text{RMSProp}(\theta, g_\theta)$
12: end while
```

On 1d Gaussian



loss vs quality for WGAN

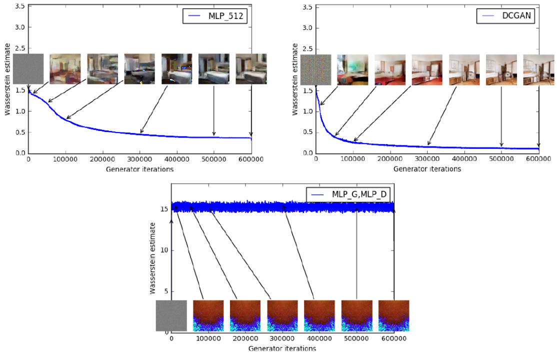
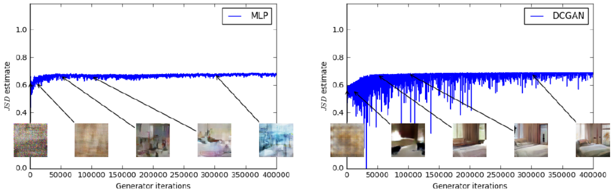


Figure 3: Training curves and samples at different stages of training. We can see a clear correlation between lower error and better sample quality. Upper left: the generator is an MLP with 4 hidden layers and 512 units at each layer. The loss decreases constistently as training progresses and sample quality increases. Upper right: the generator is a standard DCGAN. The loss decreases quickly and sample quality increases as well. In both upper plots the critic is a DCGAN without the sigmoid so losses can be subjected to comparison. Lower half: both the generator and the discriminator are MLPs with substantially high learning rates (so training failed). Loss is constant and samples are constant as well. The training curves were passed through a median filter for visualization purposes.

loss vs quality for original GAN



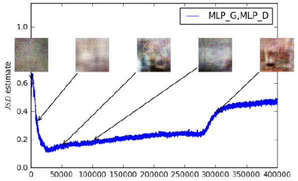


Figure 4: JS estimates for an MLP generator (upper left) and a DCGAN generator (upper right) trained with the standard GAN procedure. Both had a DCGAN discriminator. Both curves have increasing error. *Samples get better for the DCGAN but the JS estimate increases or stays constant, pointing towards no significant correlation between sample quality and loss. Bottom: MLP with both generator and discriminator. The curve goes up and down regardless of sample quality. All training curves were passed through the same median filter as in Figure 3.*

loss vs quality for GAN

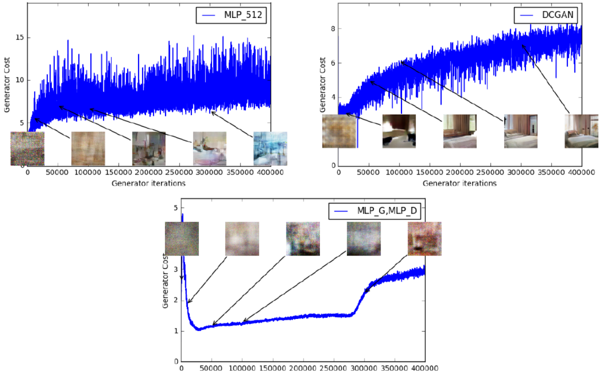
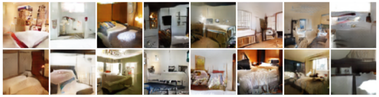
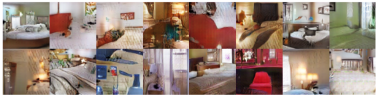


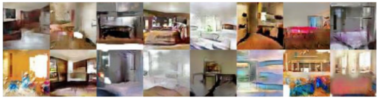
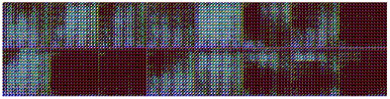
Figure 8: Cost of the generator during normal GAN training, for an MLP generator (upper left) and a DCGAN generator (upper right). Both had a DCGAN discriminator. ***Both curves have increasing error****. Samples get better for the DCGAN but the cost of the generator increases, pointing towards no significant correlation between sample quality and loss. Bottom: MLP with both generator and discriminator. The curve goes up and down regardless of sample quality. All training curves were passed through the same median filter as in Figure 3.*

image samples for different network

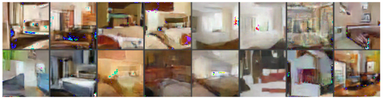
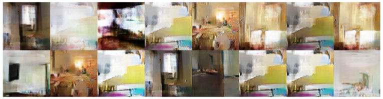
DCGAN

DCGAN without tricks

mlp

Discuss

- How to estimate K from c and architecture?
- How to compare between c and architectures?

Reference

paper

- *Generative adversarial nets (2014), I. Goodfellow et al.*
- *Unsupervised representation learning with deep convolutional generative adversarial networks (2015), A. Radford et al.*
- *Towards principled methods for training generative adversarial networks(2017), M. Arjovsky et al*
- *Wasserstein GAN (2017), M. Arjovsky et al*

web

- Unsupervised learning and GANs
- Read-through: Wasserstein GAN
- `https://zhuanlan.zhihu.com/p/25071913`
- `https://www.reddit.com/r/MachineLearning/comments/5qxoaz/r_170107875_wasserstein_gan/`
- (NIPS 2016 tutorial)Generative Adversarial Networks (GANs), slide
- `https://github.com/vdumoulin/conv_arithmetic`
- Awesome - Most Cited Deep Learning Papers
- `https://www.facebook.com/soumith/posts/10154442598946002`