Laurea Magistrale in Informatica
Dipartimento di Informatica
Università di **To**rino

GANs: Generative Adversarial Networks

Course

Neural Networks and Deep Learning

Professor

Roberto Esposito
roberto.esposito@unito.it

TODO: look at this article

Image dreamed by stable diffusion
Prompt: "abstract art about a network of points and bright lines with different colors competing for space, style cubist"

References

[1] Ian Goodfellow. *NIPS 2016 Tutorial: Generative Adversarial Networks*.

[7] Arjovsky, M., Chintala, S., and Bottou, *Wasserstein GAN*.

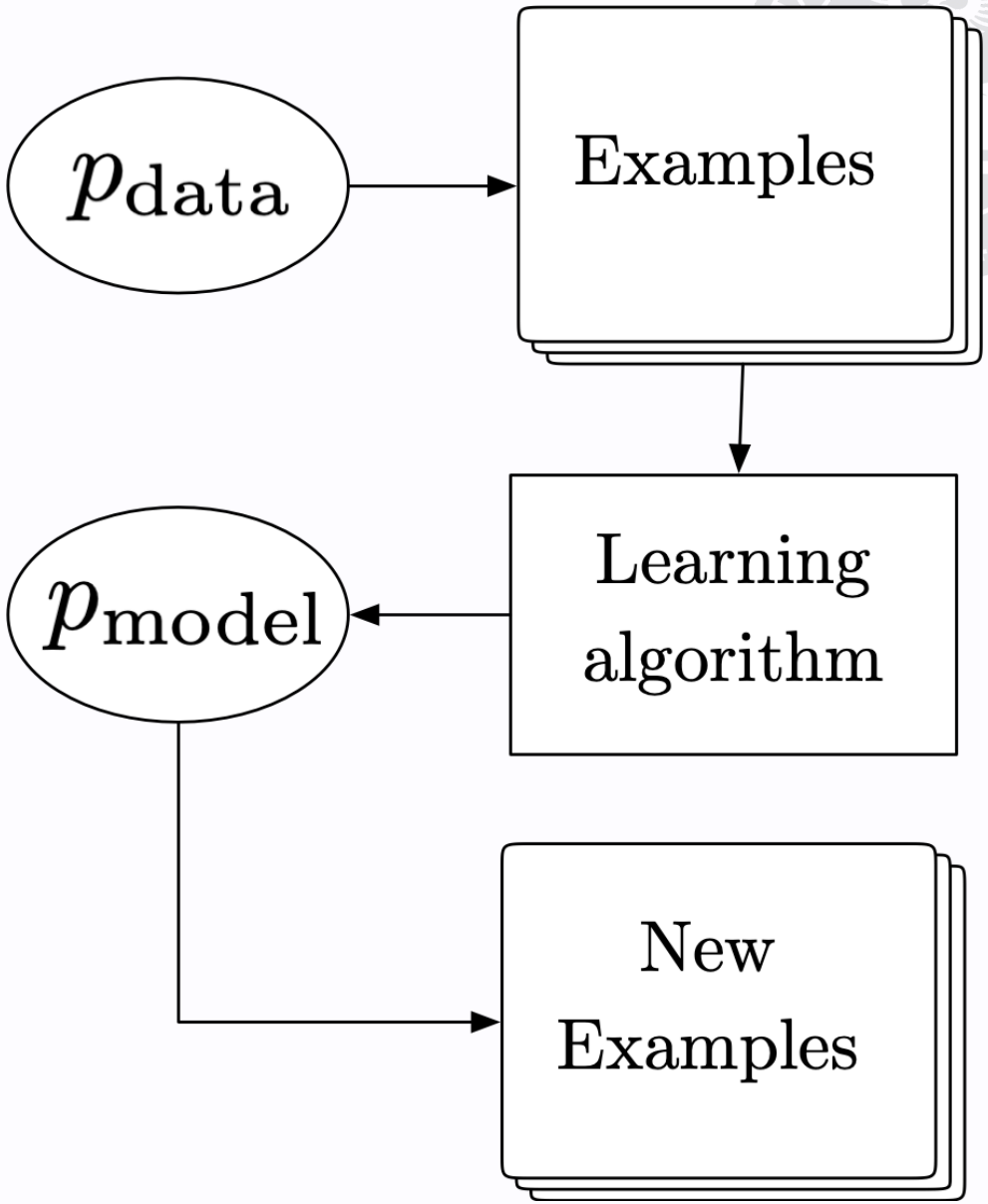
Generative Models

In the context of this lecture the term **Generative Model** refers to systems that take a training set of samples drawn from a distribution p_{data} and learns to represent an estimate of that distribution.

Generative Models

The distribution p_{model} can be estimated explicitly, or the model can only give the possibility to draw samples from it.

GANs are usually used to draw examples even if they can be designed to do both.

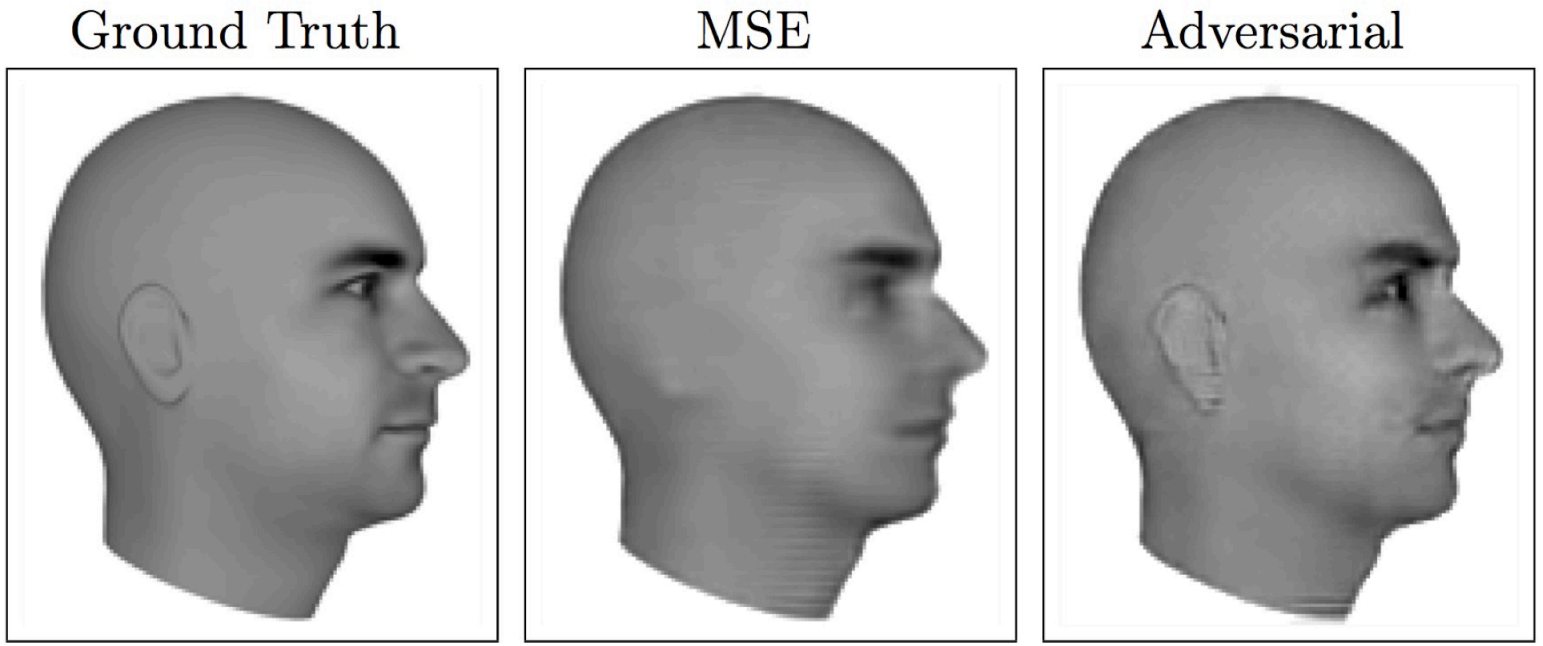


Why Study Generative Modeling?

- The ability to represent high-dimensionality distribution is important in a wide veriety of applied math and engineering domains.
- Generative models can be incorporated in (model based) *Reinforcement Learning* (RL). The generative model can be *queried* by the RL system to validate assumptions made about possible outcomes.
- They can be used as the basis of Semi Supervised Learning systems.

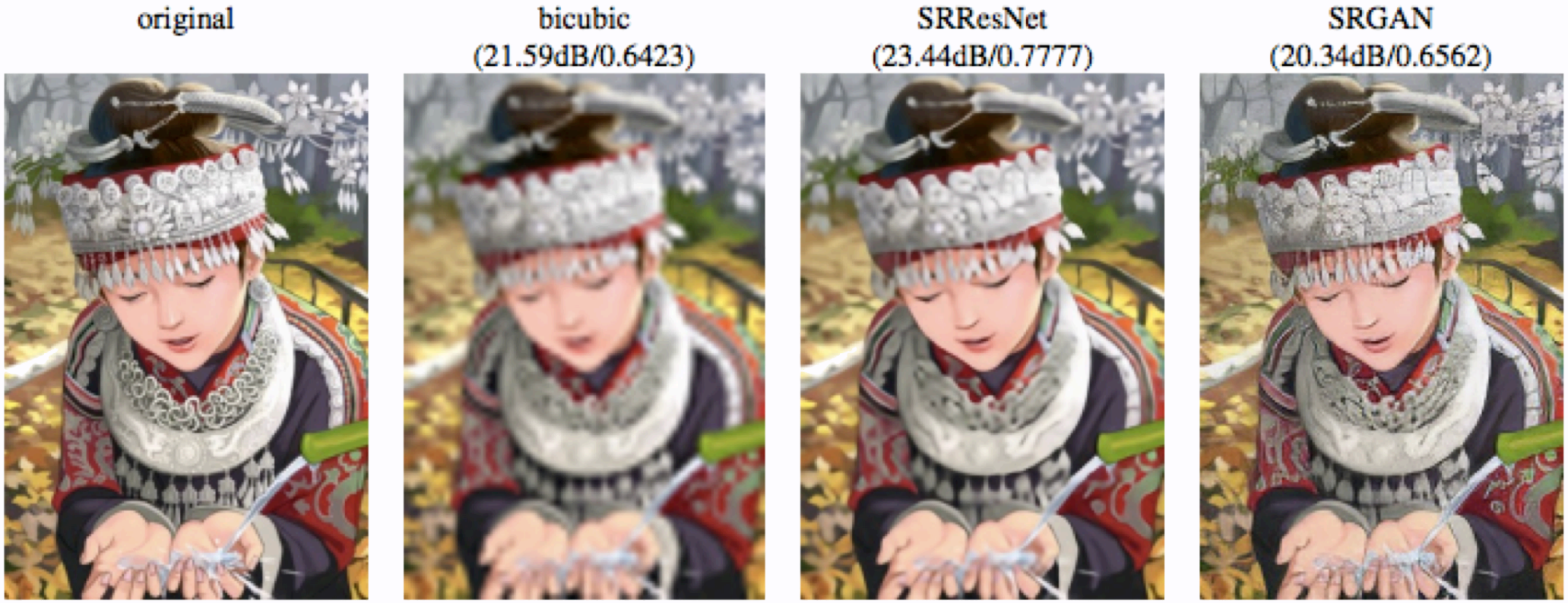
Why Study Generative Modeling?

- Generative models (GANs in particular) are good at predicting multi-modal outputs (image from [2]).



Some tasks require the generation of good samples

Single image super-resolution (image from: [3]).

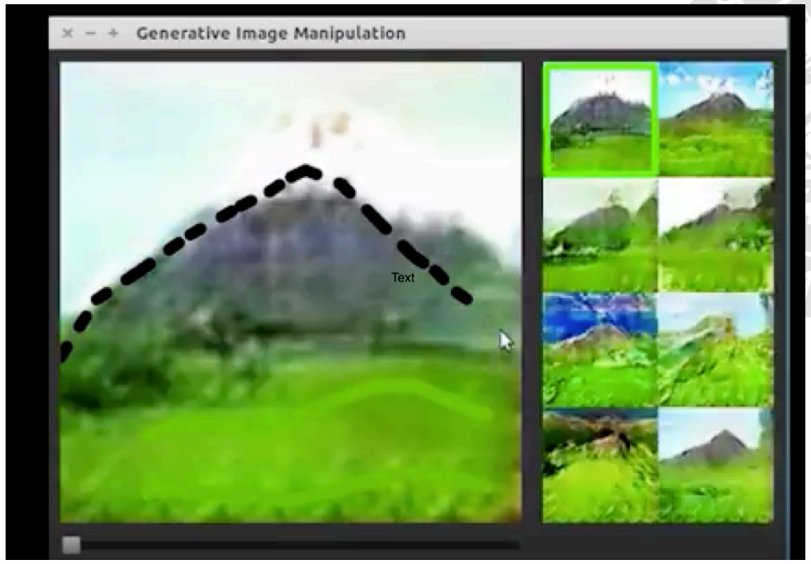


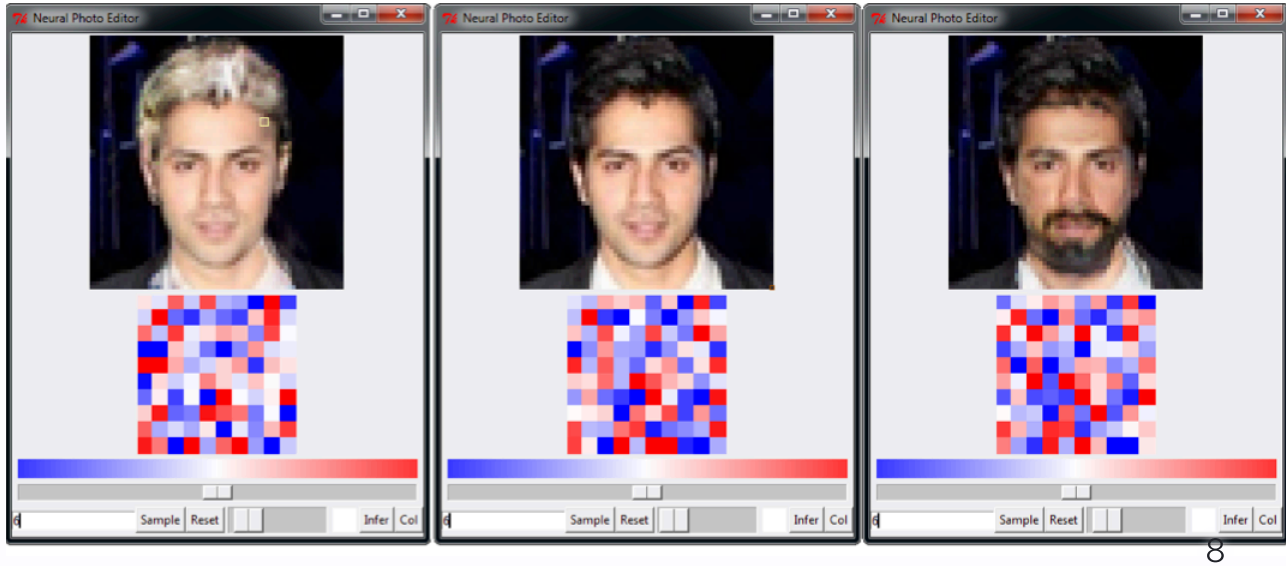
Projects where the task is to create art

Art creation (images from [4] and [5]).

YouTube demonstration 1

YouTube demonstration 2





How do GANs compare to other generative models?

Maximum Likelihood models

We will limit ourselves by comparing GANs with **maximum likelihood models**.

Let us refresh some notation and definitions.

Maximimum Likelihood Optimization

Any generative model can be trained by searching for the parameters θ that **maximize** the **likelihood of the training data given the model**.

$$\theta^{\star} = \arg\max_{\theta} p_{\text{model}}(\{\mathbf{x}\}_{i=1}^{m}; \theta)$$

Maximuml Likelihood Optimization

Almost always that optimization is made in **log-space**:

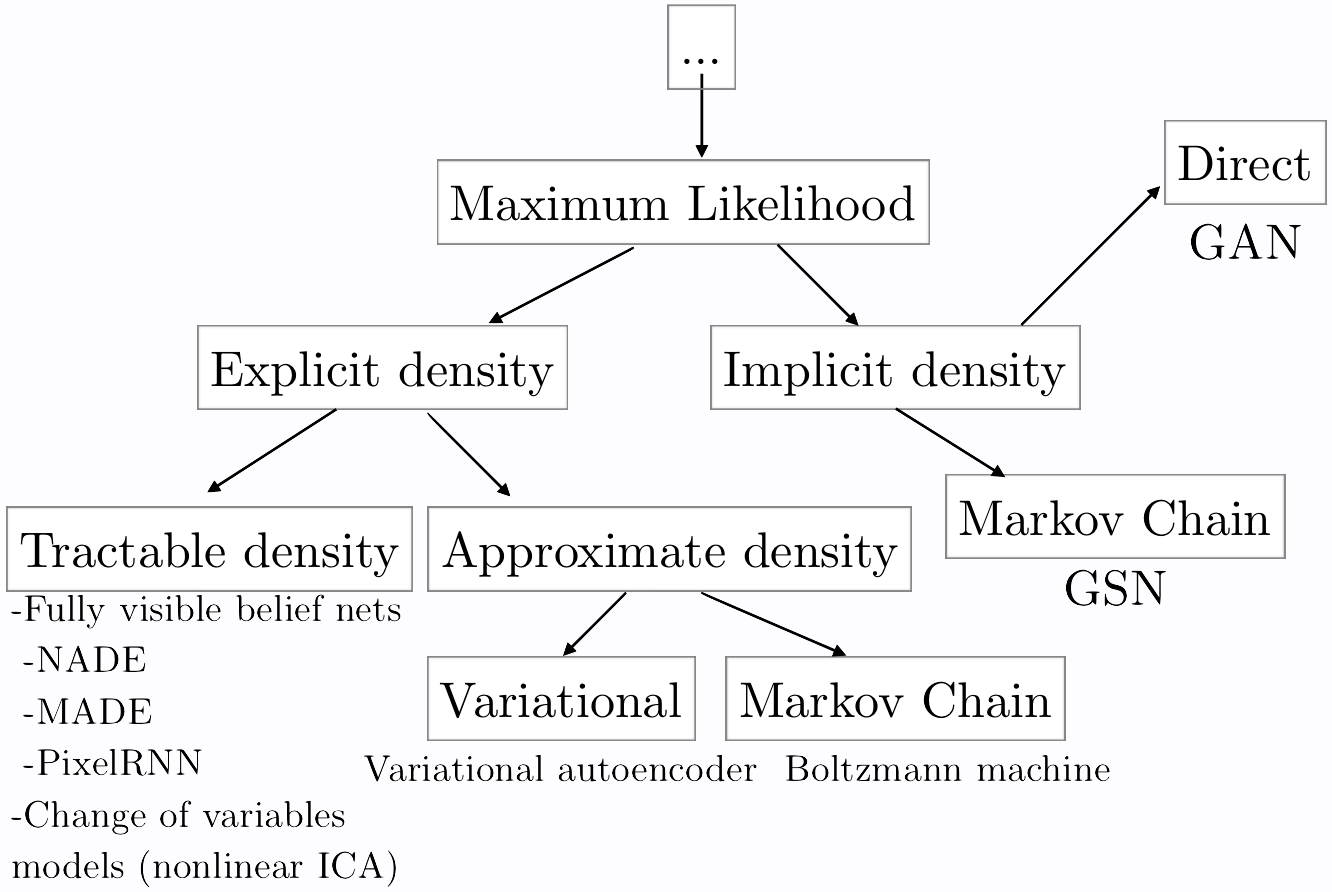
$$
\begin{aligned}
\theta^{\star} &= \arg\max_{\theta} p_{\text{model}}(\{\mathbf{x}\}_{i=1}^{m}; \theta) \\
&= \arg\max_{\theta} \log \left[p_{\text{model}}(\{\mathbf{x}\}_{i=1}^{m}; \theta) \right] \\
&= \arg\max_{\theta} \log \prod_{i=1}^{m} p_{\text{model}}(\mathbf{x}^{(i)}; \theta) \\
&= \arg\max_{\theta} \sum_{i=1}^{m} \log p_{\text{model}}(\mathbf{x}^{(i)}; \theta) \\
&\approx \arg\max_{\theta} \mathbb{E}_{\mathbf{x} \sim p_{\text{data}}} \left[\log p_{\text{model}}(\mathbf{x}; \theta) \right]
\end{aligned}
$$

Maximum Likelihood and KL divergence

Maximizing the Maximum Likelihood w.r.t. θ is the same as **minimizing** the KL divergence of p_{data} and p_{model}.

$$
\begin{aligned}
KL(p_{\text{data}} \| p_{\text{model}}) &= \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} \left[\log \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{model}}(\mathbf{x}; \theta)} \right] \\
&= \mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} \left[\log p_{\text{data}}(\mathbf{x}) - \log p_{\text{model}}(\mathbf{x}; \theta) \right] \\
&= \underbrace{\mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} \left[\log p_{\text{data}}(\mathbf{x}) \right]}_{\text{Likelihood of data}} - \underbrace{\mathbb{E}_{\mathbf{x} \sim P_{\text{data}}} \left[\log p_{\text{model}}(\mathbf{x}; \theta) \right]}_{\text{Likelihood of model}}
\end{aligned}
$$

A taxonomy of deep generative models



Explicit Density Models

trying to fully model the distribution

Fully Visible Belief Nets

They use the chain rule of probability to factor the probability of model $\mathbf{x}$ into a product of one-dimensional probabilities:

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

They are the basis of sophisticated generative models from the team of DeepMind such as WaveNet.

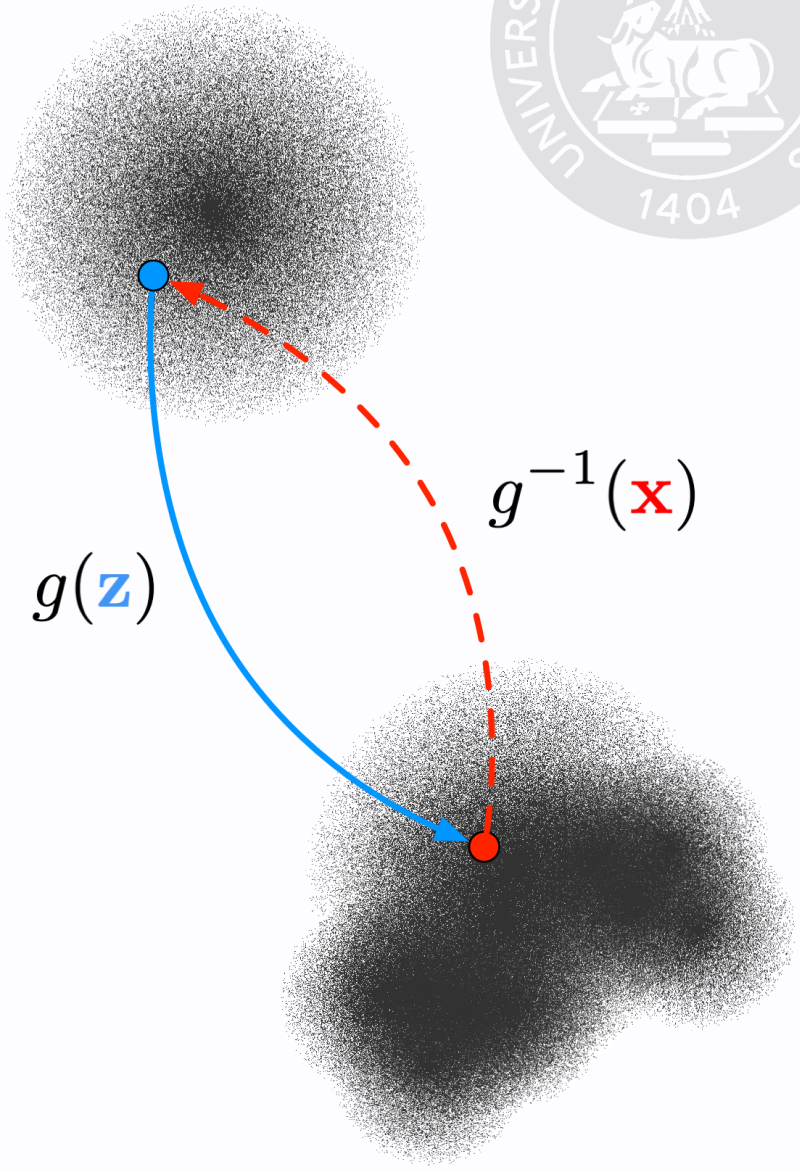
Problem: samples must be generated one entry at a time, so the cost of generating a new sample is $O(n)$.

Nonlinear Independent Components Analysis

Idea: A **simple distribution** over $\mathbf{z}$ coupled with a **non-linear transformation** g that warps space in complicated ways can yield a **complicated distribution** over $\mathbf{x}$.

If there is a vector of latent variables $\mathbf{z}$ and a continuous differentiable, invertible transformation g such that $g(\mathbf{z})$ yields a sample from the model in $\mathbf{x}$ space, then:

$$p_{\mathbf{x}}(\boldsymbol{x}) = p_{\mathbf{z}}(g^{-1}(\boldsymbol{x})) \left| \det \left(\frac{\partial g^{-1}(\boldsymbol{x})}{\partial \mathbf{x}} \right) \right|$$
$$= p_{\mathbf{z}}(g^{-1}(\boldsymbol{x})) \left| \det \left(\mathbf{J}_{\mathbf{x}}(g^{-1}(\boldsymbol{x})) \right) \right|.$$

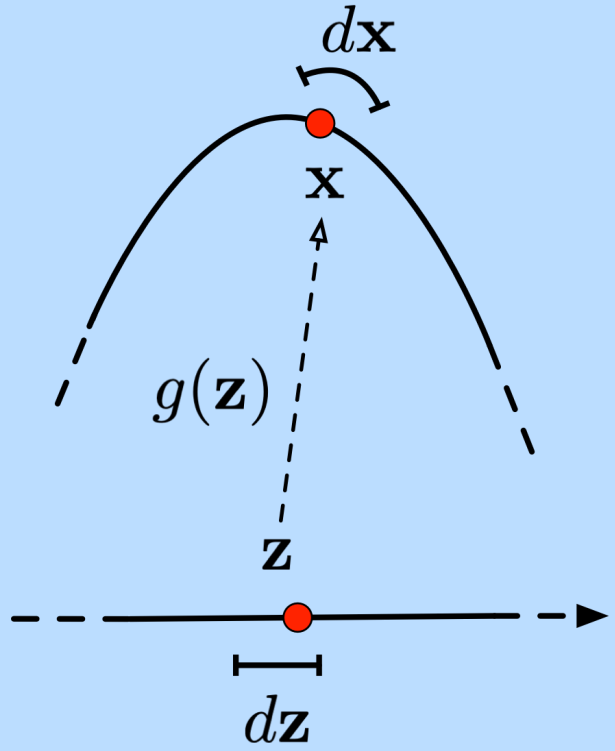


Refresher on why the Determinant appears in the formula

One might wonder **what is the role of the modulus of the determinant of the Jacobian of** g^{-1}. This is a consequence of standard probability theory for continuous variables.

Remember that the probability of a point $\mathbf{z}$ lying in an infinitesimal small region with volume $d\mathbf{z}$ is given by: $p(\mathbf{z})d\mathbf{z}$.

If $\mathbf{x} = g(\mathbf{z})$ one need to take into account the distortion caused by g to the volume $d\mathbf{x}$. Since otherwise the integral over the whole space would not be 1.



Refresher (cont.)

It is easier to see for a single random variable $\mathbf{x} \in \mathbb{R}$. We would require:

$$|p_{\mathrm{x}}(g(z))d\mathrm{x}| = |p_{\mathrm{z}}(z)d\mathrm{z}|$$

and solving from this, we get:

$$p_{\mathrm{x}}(x) = p_{\mathrm{z}}(g^{-1}(x)) \left|\frac{d\mathrm{z}}{d\mathrm{x}}\right| = p_{\mathrm{z}}(g^{-1}(x)) \left|\frac{dg^{-1}(x)}{d\mathrm{x}}\right|$$

Using:

$$x = g(z)$$
$$z = g^{-1}(x)$$

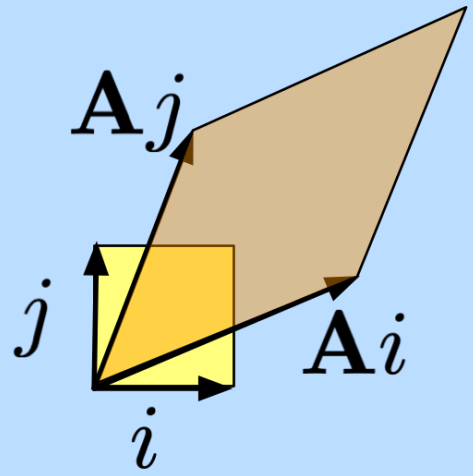
*Refresher (*cont.)*

The absolute value of the determinant of the Jacobian:

$$\left|\det\left(\frac{\partial g^{-1}(\boldsymbol{x})}{\partial \mathbf{x}}\right)\right|$$

is the extension to vector random variables of the term $\left|\frac{dg^{-1}(x)}{d\mathrm{x}}\right|$.

See [3Blue1Brown video on determinants]



$$\mathrm{vol}(\square) \cdot \det \mathbf{A} = \mathrm{vol}(\square)$$

Nonlinear Independent Components Analysis

The density $p_{\mathbf{x}}$ is tractable if the density of $p_{\mathbf{z}}$ is tractable and the Jacobian of g^{-1} is tractable.

Problem: These models impose constraints on the choice of g. In particular the invertibility restriction impose that the number of dimension of $\mathbf{x}$ is equal to the number of dimensions of $\mathbf{z}$.

Explicit Density Models

trying to approximate the distribution

Variational approximations

Variational methods use a deterministic approximation to overcome the problems of having to deal with an intractable distribution.

The main idea is to define and maximize a lower bound on the intractable distribution:

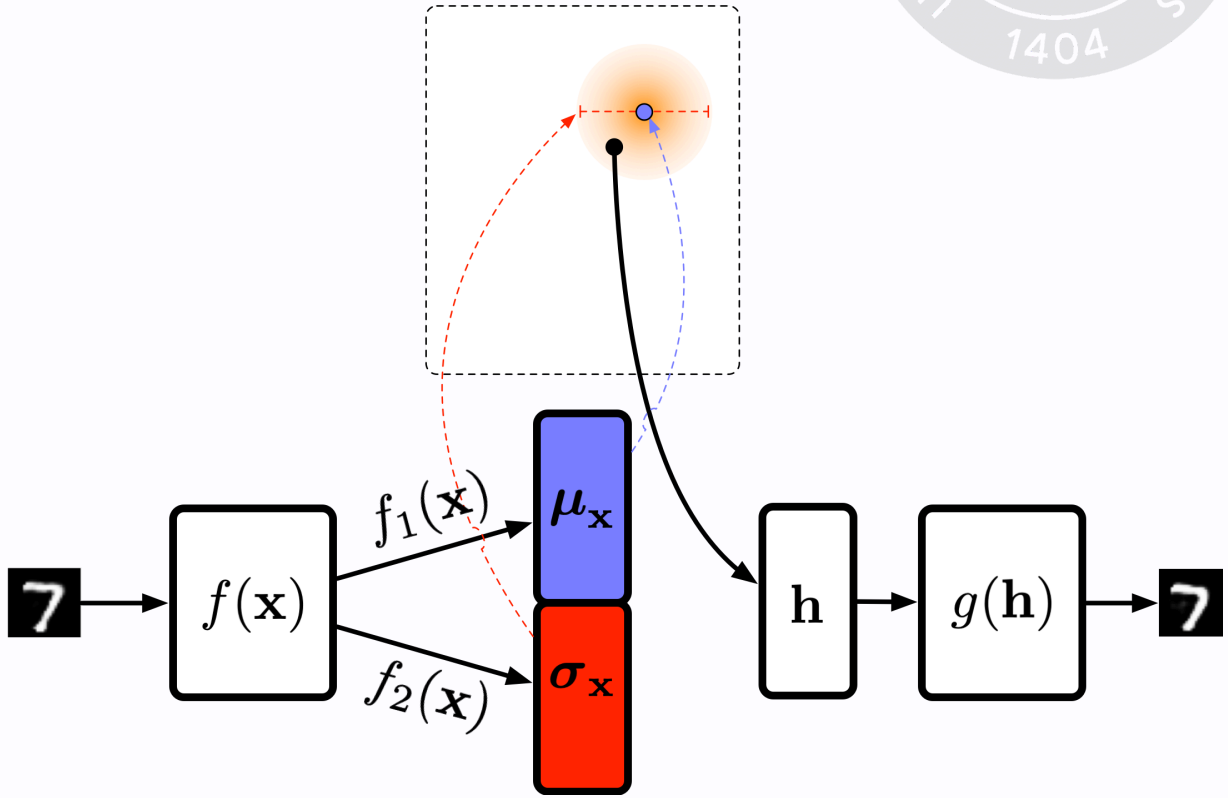
$$\mathcal{L}(\mathbf{x}; \theta) \leq \log p_{\text{model}}(\mathbf{x}; \theta)$$

This is the kind of approach taken by **variational autoencoder**.

Variational Autoencoders

Very often the approximation is based on multivariate gaussians.

As we have seen, Variational Autoencoders learns make several approximation to the true likelihood using gaussian distributions.



Variational approaches

Drawbacks: When the approximation is too crude even with a perfect optimization and infinite data the gap between $\mathcal{L}$ and the true likelihood can make the resuls poor.

For instance, variational autoencoders often obtain very good likelihood, but produce lower quality samples (w.r.t. GANs).

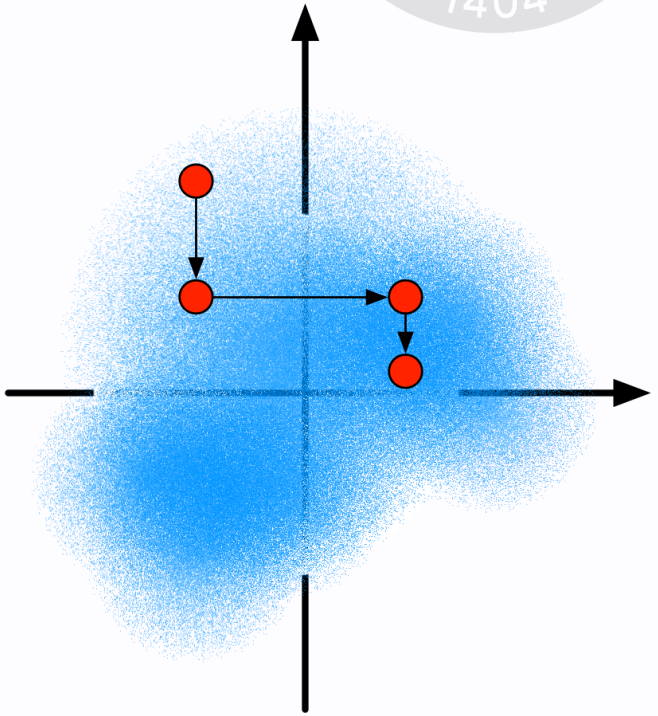
If compared to FVBNs, **variational autoencoders** are harder to train. Unfortunately GANs do not improve in this regards.

Markov Chain Approximations

Another idea to deal with intractable distributions is to use some form of stochastic approximation.

The problem here is to find a way to efficiently draw samples from $p_{\text{model}}(\mathbf{x})$ when the distribution is intractable.

Markov chain methods (**M**arkov **C**hain **M**onte **C**arlo) draw examples by repeatedly sampling from simpler distributions according to a transition operator: $\mathbf{x}' \sim q(\mathbf{x}'|\mathbf{x})$.



Markov Chain Approximations

By repeating updating $\mathbf{x}$ according to the transition operator q, Markov chains methods can sometimes guarantee that $\mathbf{x}$ will eventually converge to sampling from $p_{\text{model}}(\mathbf{x})$.

Challenge: Slow Convergence

- Learning Phase: drawing of examples via MCMC can be too costly.
- Inference Phase: Inefficient if compared to GANs.

Implicit Density Models

Generative Stochastic Networks

Some models model the distribution only implicitly. **Generative stochastic models** learns a Markov transition operator that allows one to draw from the implicit model.

Problem: By approximating the drawing of examples using a Markov Chain, these models have the same problems we saw for models using a Markov chain approximation.

Generative Adversarial Networks

GANs model the distribution only implicitly **and** offer a single step sample generation method.

Summarizing:

- GANs can generate samples in parallel
- The generator function has very few restrictions
- No costly Markov chain approximations
- No variational bound is needed
- Subjectively GANs are regarded to produce better samples.

How do GANs work?

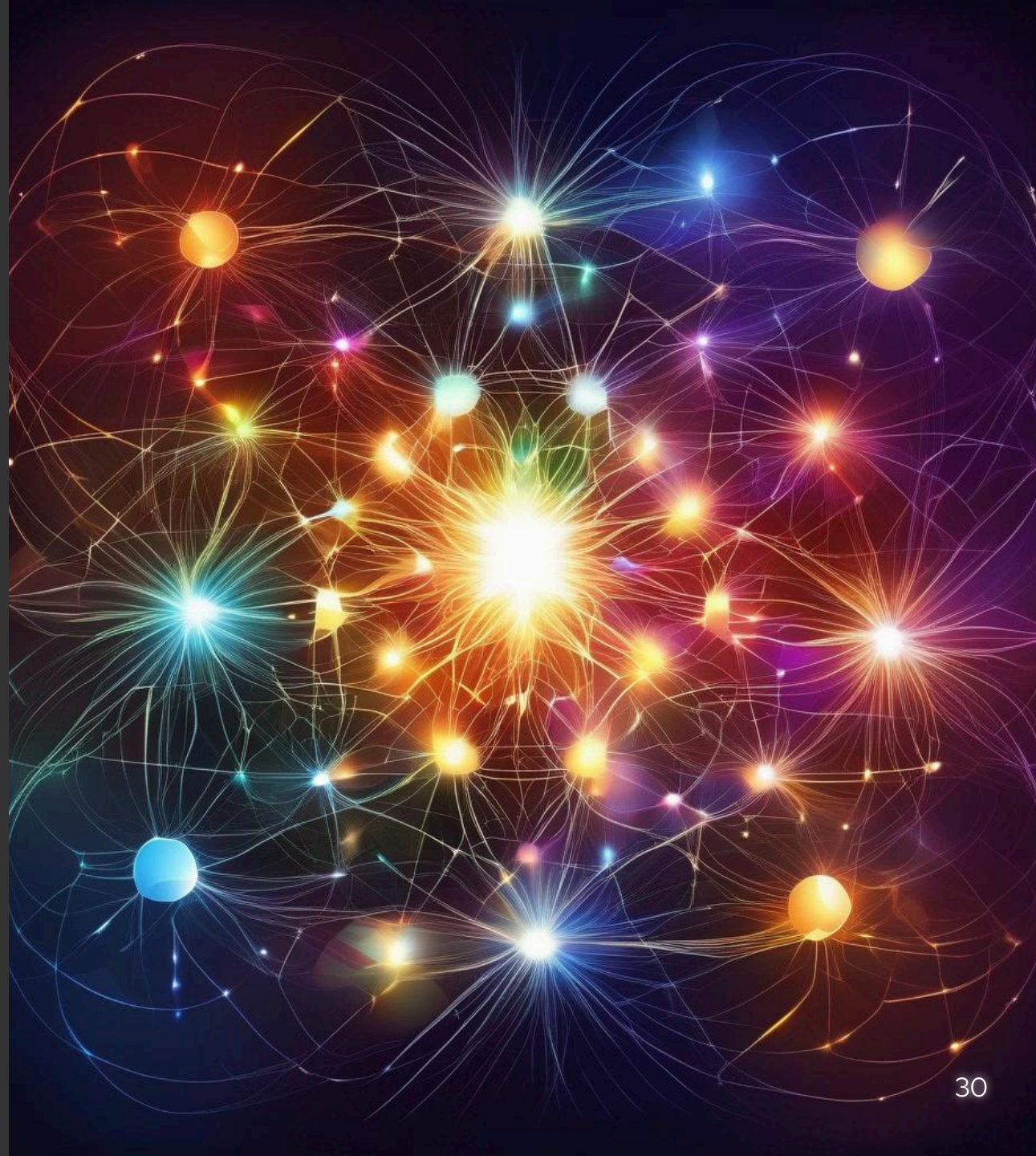
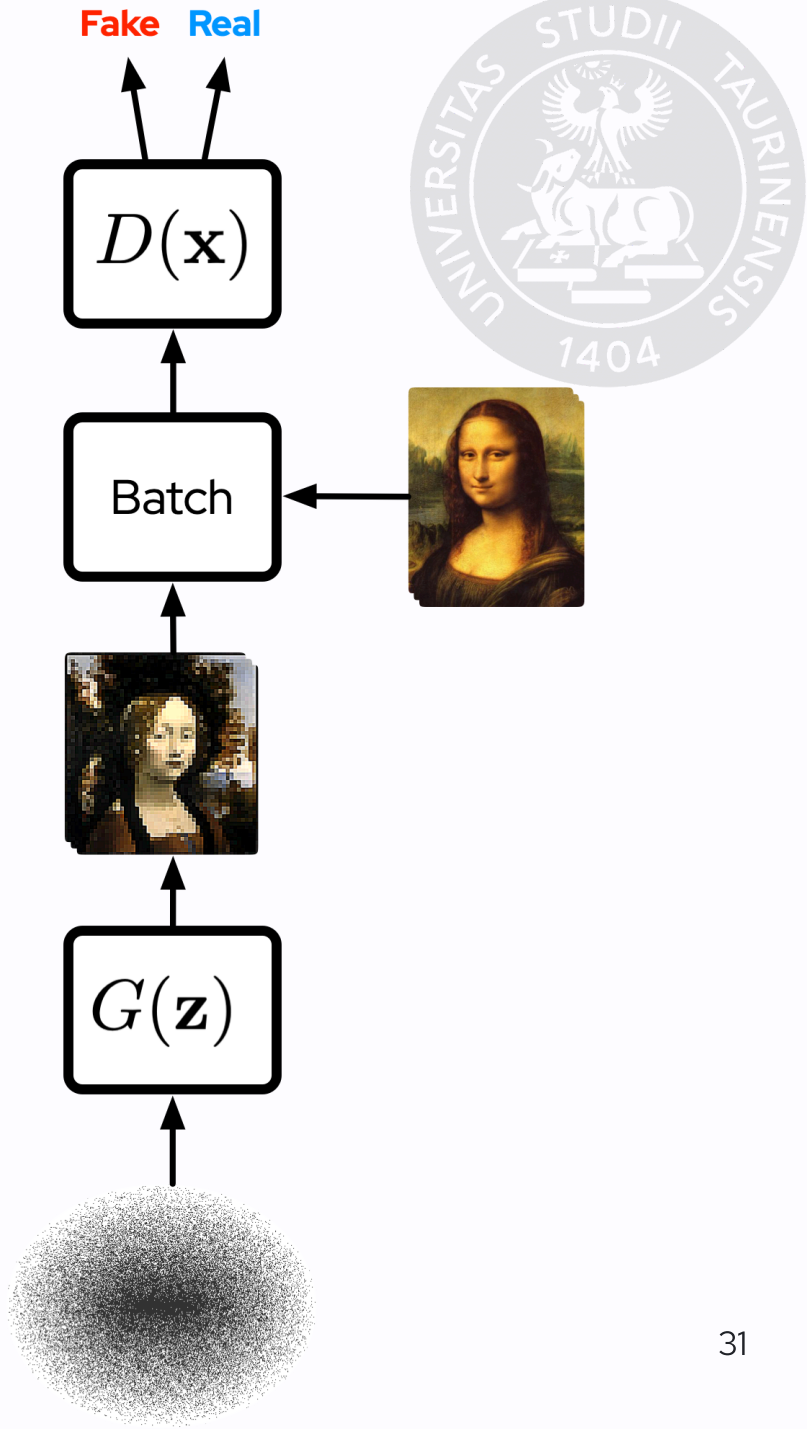


Image dreamed by stable diffusion
Prompt: "network of lights, style artistic-abstract"

How do GANs work?

The idea in GANs is to set up a game between two players:

- the **generator** G creates samples intended to come from the same distribution as the training data;
- the **discriminator** D examines samples to determine whether they are real or false



GANs

Definition of G and D

G and D are two differentiable functions (i.e., two NNs).

- *G takes an input $\mathbf{z}$, is defined in terms of parameters θ_G and outputs a value $\tilde{\mathbf{x}}$ from the same space of $\mathbf{x}$.*
- *D takes an input $\mathbf{x}$, is defined in terms of parameters θ_D and outputs a value $y \in \{\text{Fake}, \text{Real}\}$.*

In the following we will assume that the discriminator tries to predict the probability that the input is real. That is: $D(\mathbf{x}) \approx 1$ if the discriminator believes that the $P(y = \text{Real}|\mathbf{x})$ is high.

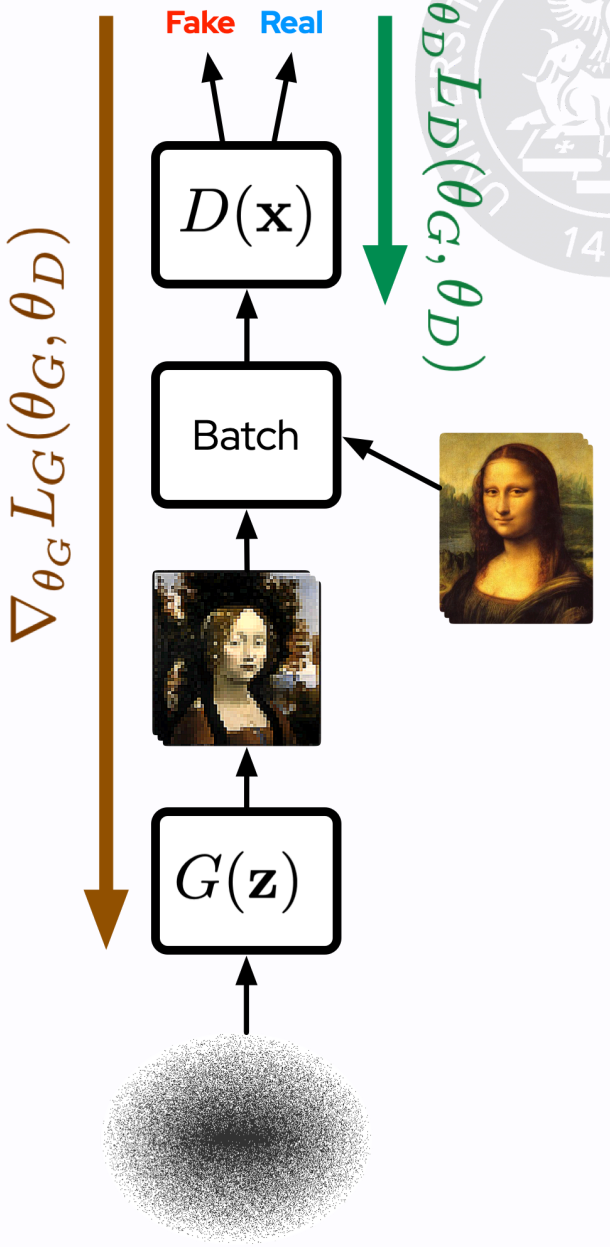
The generator and discriminator **cost functions**, denoted as $L_G(\theta_G, \theta_D)$ and $L_D(\theta_G, \theta_D)$, are defined with respect to both θ_G and θ_D.

GANs

Each player want to minimize its cost function but **can only do so by acting on its own parameters**.

The optimization can be seen as a zero-sum game between the two players (*not always true*, depending on the definition of L_G and L_D).

The solution of a zero-sum game is a *Nash equilibrium*, i.e., a point in the (θ_G, θ_D) space such that the point is local minimum of L_D w.r.t. θ_D and a local minimum of L_G w.r.t. θ_G.

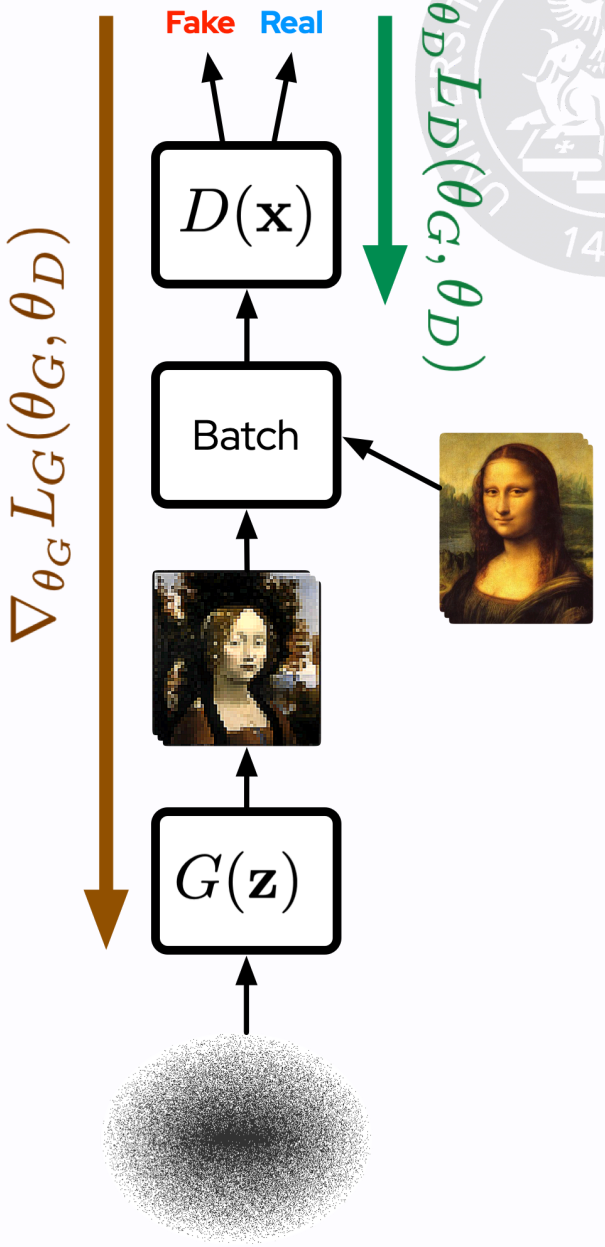


Training

The training process consists of simultaneous SGD.

- sample two minibatches (one from real data, one from generated data).
- evaluate the two losses and update θ_D using the gradients from L_D, update θ_G using the gradients from L_G.

Sometimes more steps are performed on the discriminator before going back to the generator (e.g., 1 step on the generator every 5 steps on the discriminator).



The Discriminator Cost Function

The** discriminator**'s cost function is:

$$L_D(\theta_D, \theta_G) = -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log D(\mathbf{x})] - \frac{1}{2}\mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))]$$

This is the usual cross-entropy used when minimizing a binary classifier with sigmoid output units.

Initially, all variants of GANs used this exact cost for the discriminator (costs for the generator changes from model to model).

Cross Entropy (refresher)

The cross-entropy $H(p, q)$ is a measure of how many bits of information (on average) we have to send when the code is optimized for distribution q and the data follow distribution p.

$$H(p, q) = -\mathbb{E}_p[\log q] = -\sum p(x) \log q(x)$$

Note that the optimal code would send only $H(p)$ bits of information. The relative entropy (a.k.a., KL divergence) is the difference between $H(p, q)$ and $H(p)$.

Cross Entropy (refresher)

In the binary case $x \in \{0, 1\}$, and the cross-entropy is:

$$H(p, q) = -\sum_{x' \in \{0,1\}} p(x = x') \log q(x = x')$$
$$= -p(x = 1) \log q(x = 1) - p(x = 0) \log q(x = 0)$$

In the binary classification case, p corresponds to the true distribution of the data: $p(x = 1) = y$ or $p(x = 0) = 1 - y$. Analogously, q corresponds to the prediction of the classifier: $q(x = 1) = \hat{y}$ or $q(x = 0) = 1 - \hat{y}$.

The above equation then becomes:

$$H(p, q) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y})$$

Cross Entropy and GANs

$\text{CrossEntropy}(y, \hat{y}) = -\mathbb{E}_{\mathbf{x}}[y \log \hat{y} + (1-y)\log(1-\hat{y})]$	We start from the definition of the cross-entropy for a binary classifier.
$\text{CrossEntropy}(y=1, \hat{y}=D(\mathbf{x})) = -\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log D(\mathbf{x})]$	In our case when $y=1$ we are dealing with Real examples and the prediction we are interested in is $D(\mathbf{x})$ with $\mathbf{x} \sim p_{\text{data}}(\mathbf{x})$.
$\text{CrossEntropy}(y=0, \hat{y}=D(G(\mathbf{z}))) = -\mathbb{E}_{\mathbf{z}}[\log(1-D(G(\mathbf{z})))]$	When $y=0$ we are dealing with Fake examples and the prediction is given by $D(G(\mathbf{z}))$ with $\mathbf{z}$ drawn from its own distribution.

Putting everything together we get the discriminator loss we just introduced.

$$L_D(\theta_D, \theta_G) = -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[\log D(\mathbf{x})] - \frac{1}{2}\mathbb{E}_{\mathbf{z}}[\log(1-D(G(\mathbf{z})))]$$

The Generator Cost Function

In its simplest form the generator cost function is just:

$$L_G = -L_D.$$

This is the situation that determines the **zero-sum game** situation: each time the discriminator lowers its loss, the generator is penalized by the same amount.

Minimax

Since one of the loss is the inverse of the other, we see that the game is the one where the players are just trying to minimize/maximize the same function (along different directions). If we define:

$$V(\theta_D, \theta_G) = -L_D(\theta_D, \theta_G)$$

then the whole game can be summarized as:

$$\theta_{G^\star}, \theta_{D^\star} = \arg\min_{\theta_G} \max_{\theta_D} V(\theta_D, \theta_G)$$

Minimax game optimum

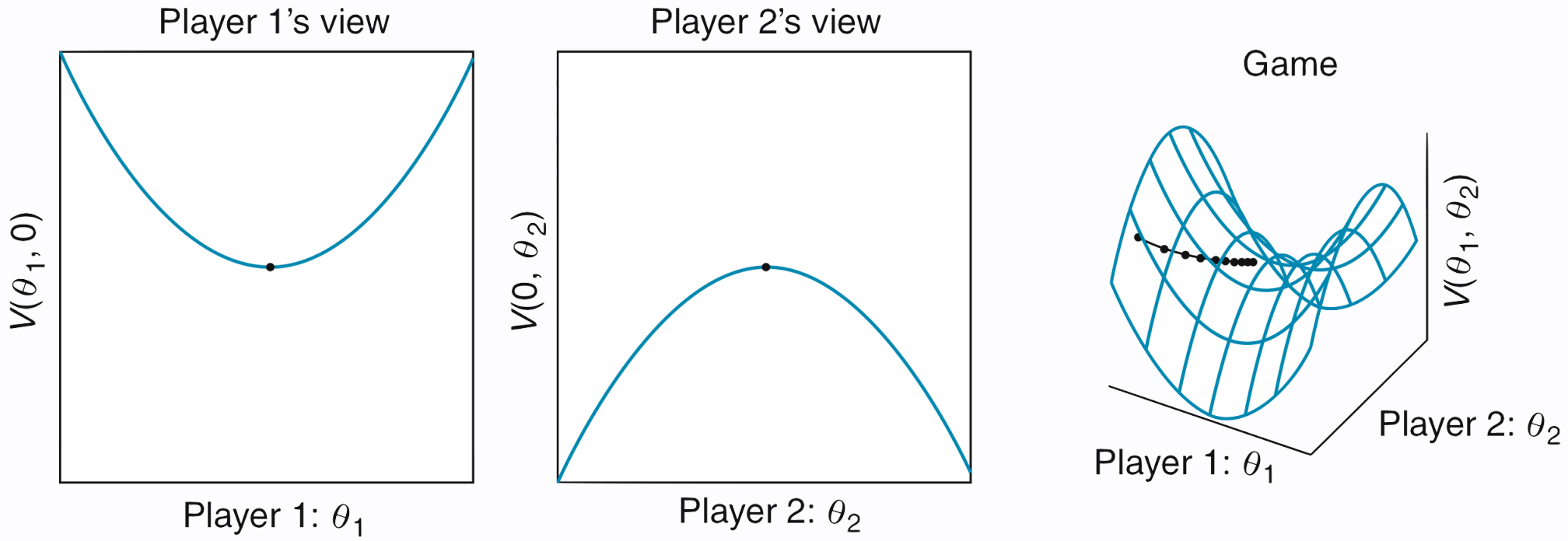


Figure from [6].

Theoretical advantages of this setup

This game is interesting from the theoretical point of view. It can be shown that this game corresponds to minimizing the Jensen-Shannon divergence between the data and the model distribution.

$$JSD(P\|Q) = \frac{1}{2}KL(P\|M) + \frac{1}{2}KL(Q\|M)$$

where $M = \frac{1}{2}(P + Q)$.

The game **converges to equilibrium** if both players' policies can be updated in **function space** (this is not possible in practice though).

Question

Assume $L_G = -L_D$ and that the discriminator succeeds in discriminating with high confidence between *real* and *fake*.

What do you expect the **gradient of the loss** used by the **generator** to be like?

Heuristic, Non-Saturating Game

Setting the cost $L_G = -L_D$ is good for the theoretical analysis, but it does not perform particularly well.

In the *minimax* game the **discriminator minimizes** the cross-entropy, but the generator **maximizes** it. This is unfortunate because the gradient of the loss vanishes when the discriminator rejects the generator examples with high confidence.

Heuristic, Non-Saturating Game

To solve the vanishing gradient problem just mentioned, one can make the generator to minimize a cross-entropy term tailored to its view of the problem:

$$L_G = -\frac{1}{2}\mathbb{E}_{\mathbf{z}} \log D(G(\mathbf{z}))$$

In this new game **the generator maximizes the probability that the discriminator is mistaken**.

Heuristic, Non-Saturating Game

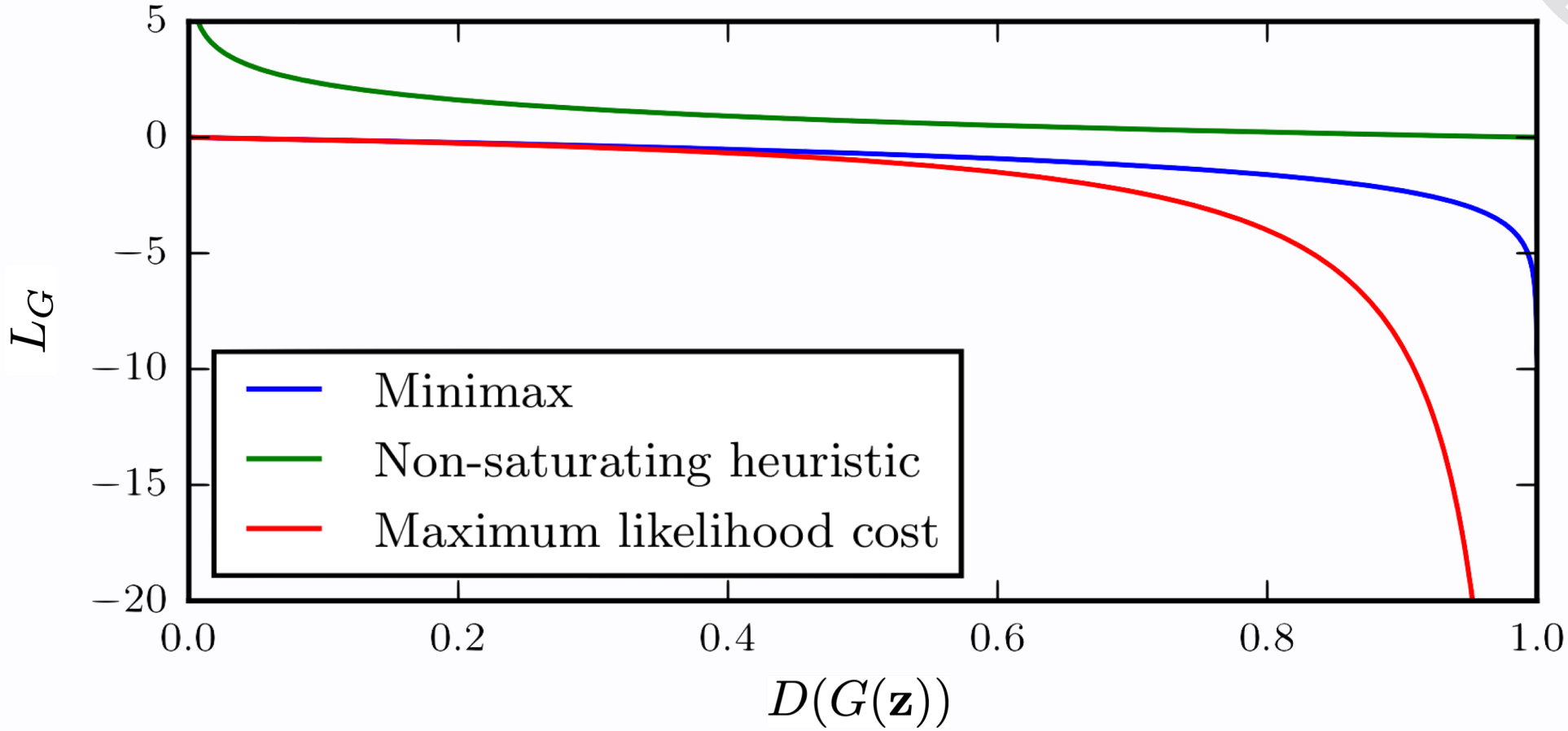


Image from [1] p. 26.

Heuristic, Non-Saturating Game

Summarizing, the new game is setup between two players G and D that minimize the following loss functions:

$$L_D(\theta_D, \theta_G) = -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}} \log D(\mathbf{x}) - \frac{1}{2}\mathbb{E}_{\mathbf{z}} \log(1 - D(G(\mathbf{z})))$$

$$L_G(\theta_D, \theta_G) = -\frac{1}{2}\mathbb{E}_{\mathbf{z}} \log D(G(\mathbf{z}))$$

This time the two losses are not the one the inverse of the other and the game is no longer a zero-sum game.

Why do GANs Work?

For a long time the reason behind the good performances of GANs was attributed to the minimization of the Jensen-Shannon divergence instead of minimizing the KL divergence.

The KL divergence is not symmetric and maximum likelihood estimation minimizes $KL(p_{\text{data}}||p_{\text{model}})$.

Minimizing the Jensen-Shannon divergence is more akin to minimizing $KL(p_{\text{model}}||p_{\text{data}})$.

Why do GANs Work?

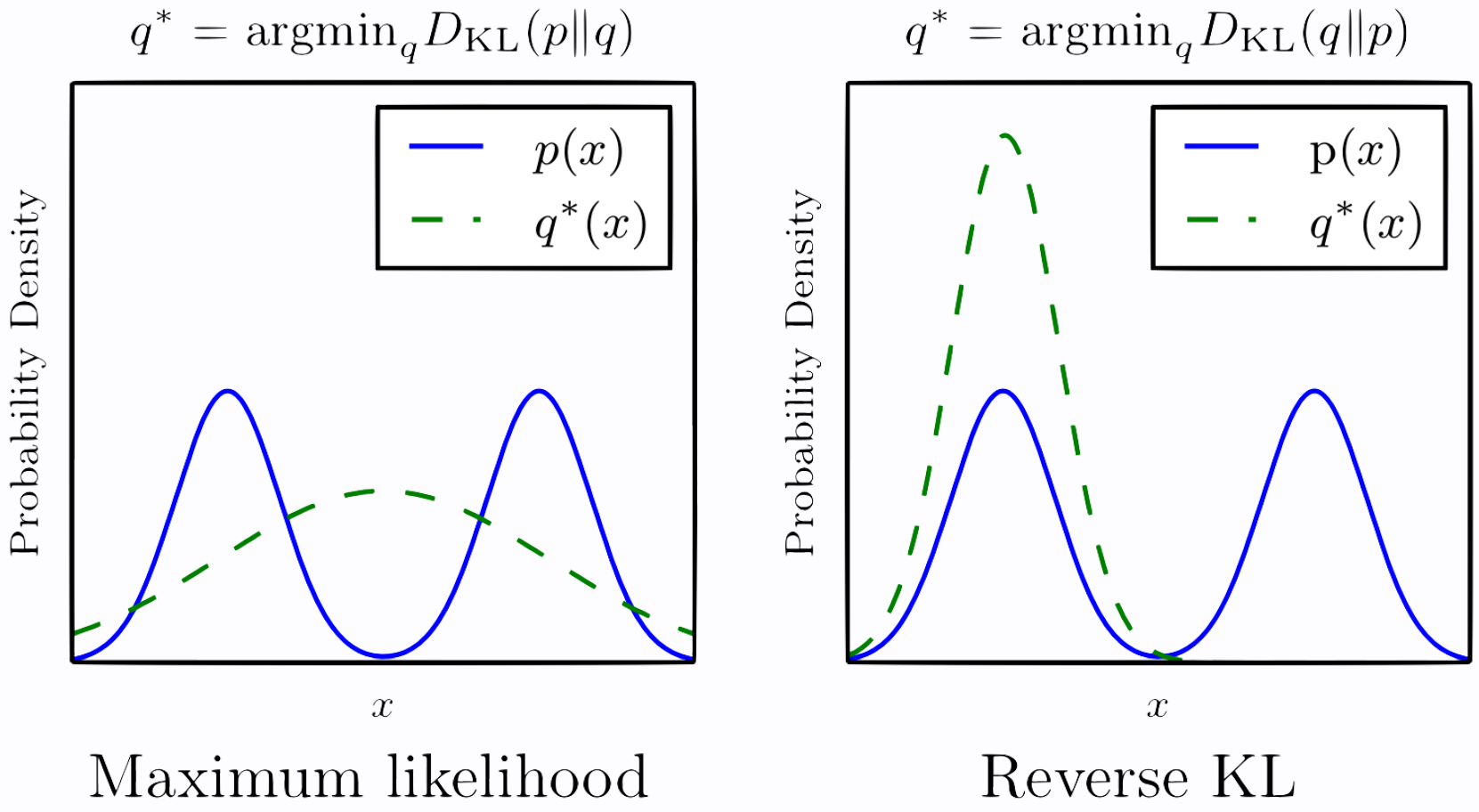


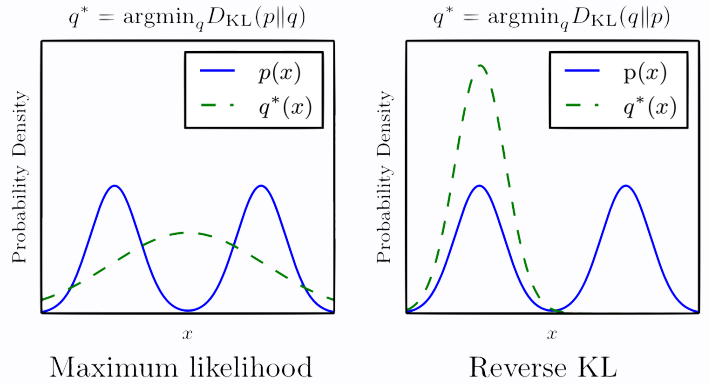
Image from [1] p. 24.

Why do GANs Work?

In both cases it is best to find a q as close as possible to p.
The difference is how the two formulae weights their errors.

$$KL(p(x)||q(x)) = \mathbb{E}_{p(x)} \log \frac{p(x)}{q(x)}$$

weights the errors using $p(x)$: if $p(x)$ is bimodal and $q(x)$ unimodal, it is best to spread $q(x)$ between the two modes, otherwise the error on the second part of the distribution would get weighted too much.

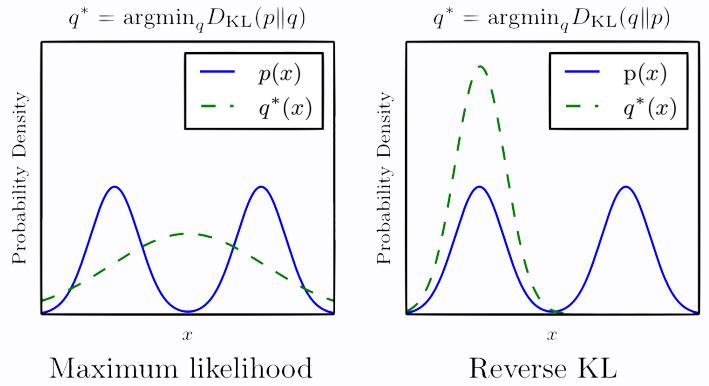


Why do GANs Work?

$$KL(q(x)||p(x)) = \mathbb{E}_{q(x)} \log \frac{q(x)}{p(x)}$$

weights the errors using $q(x)$. Then, it is important that where $q(x)$ is high also $p(x)$ is high. Concentrating all the probability mass in just one of the two modes would work nicely.

This reasoning let people think that GANs worked well because it did not average between modes.



Why do GANs Work?

Some new evidence suggests that the use of the Jensen-Shannon divergence does not explain why GANs make sharper samples:

- Using maximum likelihood to optimize GANs does not show problems in selecting a small number of modes and generating sharp images.
- GANs often choose to generate from very few modes (fewer than the number allowed by the model capacity). Reverse KL would select as many modes as allowed by the model.

Why do GANs Work?

This suggests that GANs **choose to generate a small number of modes due to a defect in the training procedure**, rather than due to the divergence they aim to minimize.

The reason why this happens is not clear. Maybe it **makes different approximations** than other models, or maybe it **optimizes a different family of functions**.

DCGANs

Image dreamed by stable diffusion
Prompt: "network of lights, style abstract expressionism"

Most GANs today are loosely based on the *Deep Convolutional GAN* architecture (DCGAN).

Main insights for this architecture are:

- batch normalization in most layers for both D and G, batches for G and D are normalized separately. The last layer of the generator and the first layer of the discriminator are not batch normalized.

DCGANs (cont.)

- Architecture contains neither pooling nor "unpooling" layers. When the generator needs to increase the spatial dimension of the representation it uses *transposed convolution* (a.k.a., deconvolution) with a stride greater than 1.
- The optimizer of choice is Adam instead of SGD with momentum.

DCGANs

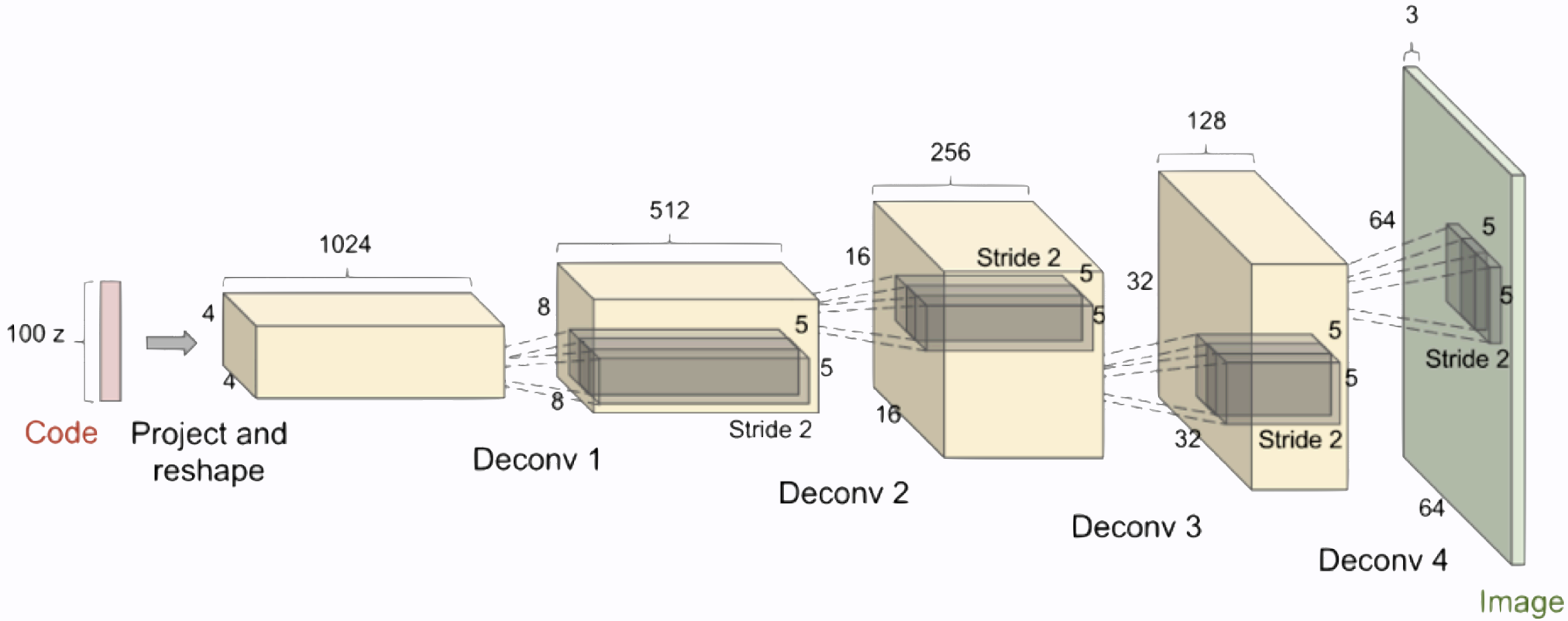


Image from [1] p. 27.

Wasserstein GANs

This part of the lecture is based on [7].

All images in this section can be found therein.

Wasserstein distance (a.k.a., Earth-Mover distance)

*The **Wasserstein distance** between two probability distributions p and q is defined as:*

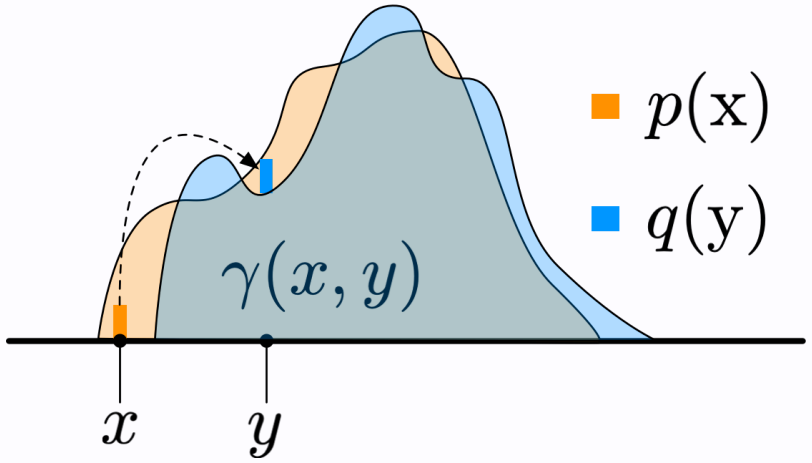
$$W(p,q) = \inf_{\gamma \in \Pi(p,q)} \mathbb{E}_{(x,y)\in\gamma} \|x - y\|$$

where $\Pi(P,Q)$ denotes the set of all joint distributions $\gamma(x,y)$ whose marginals are respectively p and q.

Interpretation of the Wasserstein distance

$$W(p,q) = \inf_{\gamma \in \Pi(p,q)} \overbrace{\mathbb{E}_{(x,y)\in\gamma}\|x-y\|}^{\text{Total work needed by policy } \gamma} = \inf_{\gamma \in \Pi(p,q)} \int_{\mathbb{R}^d \times \mathbb{R}^d} \overbrace{\|x-y\|}^{\text{distance}} \underbrace{\gamma(x,y)}_{\text{qty moved}} dxdy$$

- $\gamma(x,y)$: a policy to move "mass" from x to y;
- $\int \gamma(x,y)dy = p(x)$: total mass moved from x is $p(x)$.
- $\int \gamma(x,y)dx = q(y)$: total mass moved to y is $q(y)$;



Then, Wasserstein distance can be interpreted as **the minimum** amount of "work" required to transform the probability mass p to q.

WGANs

It can be shown that the earth-mover distance has better properties than other distances and divergences used for GANs (e.g., Kullback-Leibler, Jensen-Shannon). In particular, **it allows learning to convergence in many situations where other measures fail**.

WGANs

Unfortunately the infimum in the definition of W is intractable, however it can be shown that **under the assumption** that the family of functions considered $\{f_{\theta_D}(x)\}$ is **Lipschitz continuous**, one can **train the discriminator** to maximize the W distance by maximising:

$$\mathbb{E}_{\mathbf{x}\sim p_{\text{data}}}[D_{\theta_D}(\mathbf{x})] - \mathbb{E}_{\mathbf{z}}[D_{\theta_D}(G_{\theta_G}(\mathbf{z}))]$$

The generator is instead trained to minimize:

$$L_G(\theta_G, \theta_D) = -\mathbb{E}_{\mathbf{z}}[D_{\theta_D}(G_{\theta_G}(\mathbf{z}))]$$

Lipshitiz continuity

Without going into the details, a function f is Lipshitz continuous if there exists a constant c such that:

$$\|f(x) - f(y)\| \leq c\|x - y\|$$

for all x and y in the domain of f.

In other words, the function cannot change too much in a small region of the domain.

WGANs implementation details

Everything can be trained by backpropagation with only a small trick to ensure that the the learned function is Lipshitz continuous. **Every time the θ_D parameters are updated, they are clipped in $[-c, c]$**, where c is a user-defined constant.

Another difference with respect to the original GANs is that **the activation function of the last layer of the discriminator is linear** instead of sigmoid. This is because the discriminator is no longer meant to model a probability distribution (for this reason is often referred to as the Critic instead of the Discriminator).

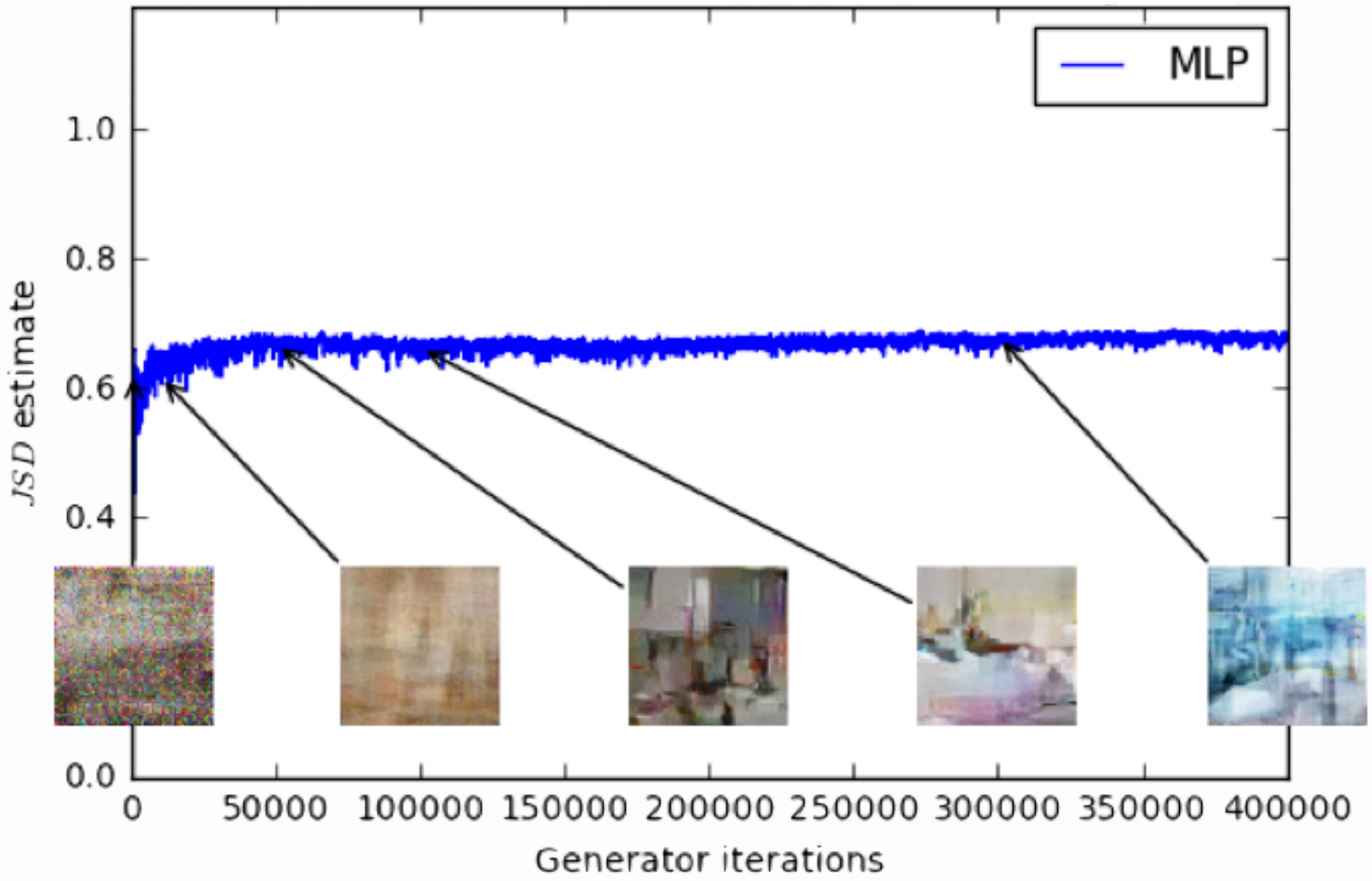
WGANs advantages

The new loss metric is **meaningful** and **correlates** with the **generator's convergence** and sample quality.

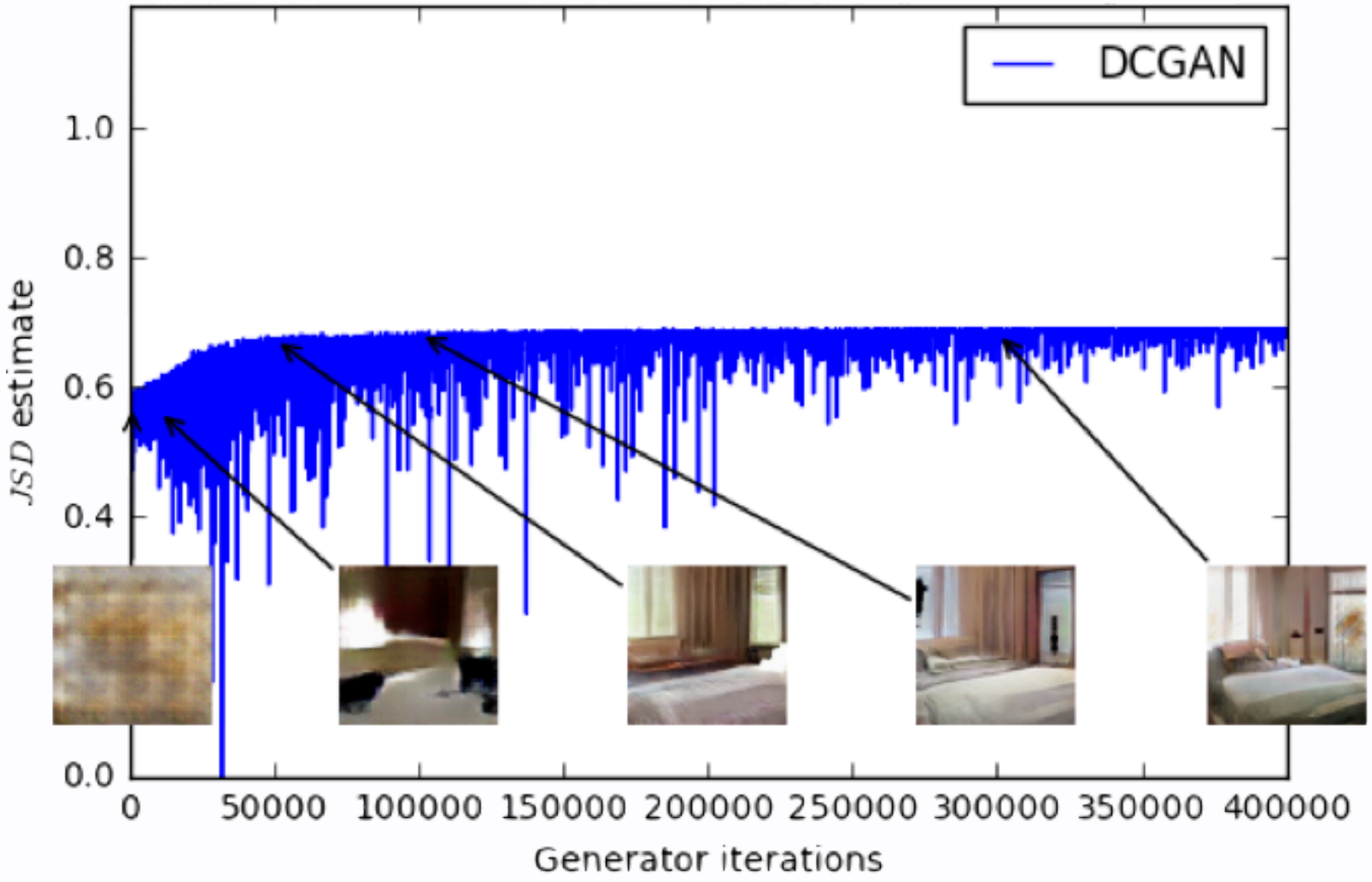
In the WGAN algorithm the discriminator f is usually trained near optimality. In this situation it can be shown that the **loss is an estimate of the earth-mover distance up to a given factor** (determined by the constant c).

Empirical evidence shows that this **correlates with the quality of the generated samples**.

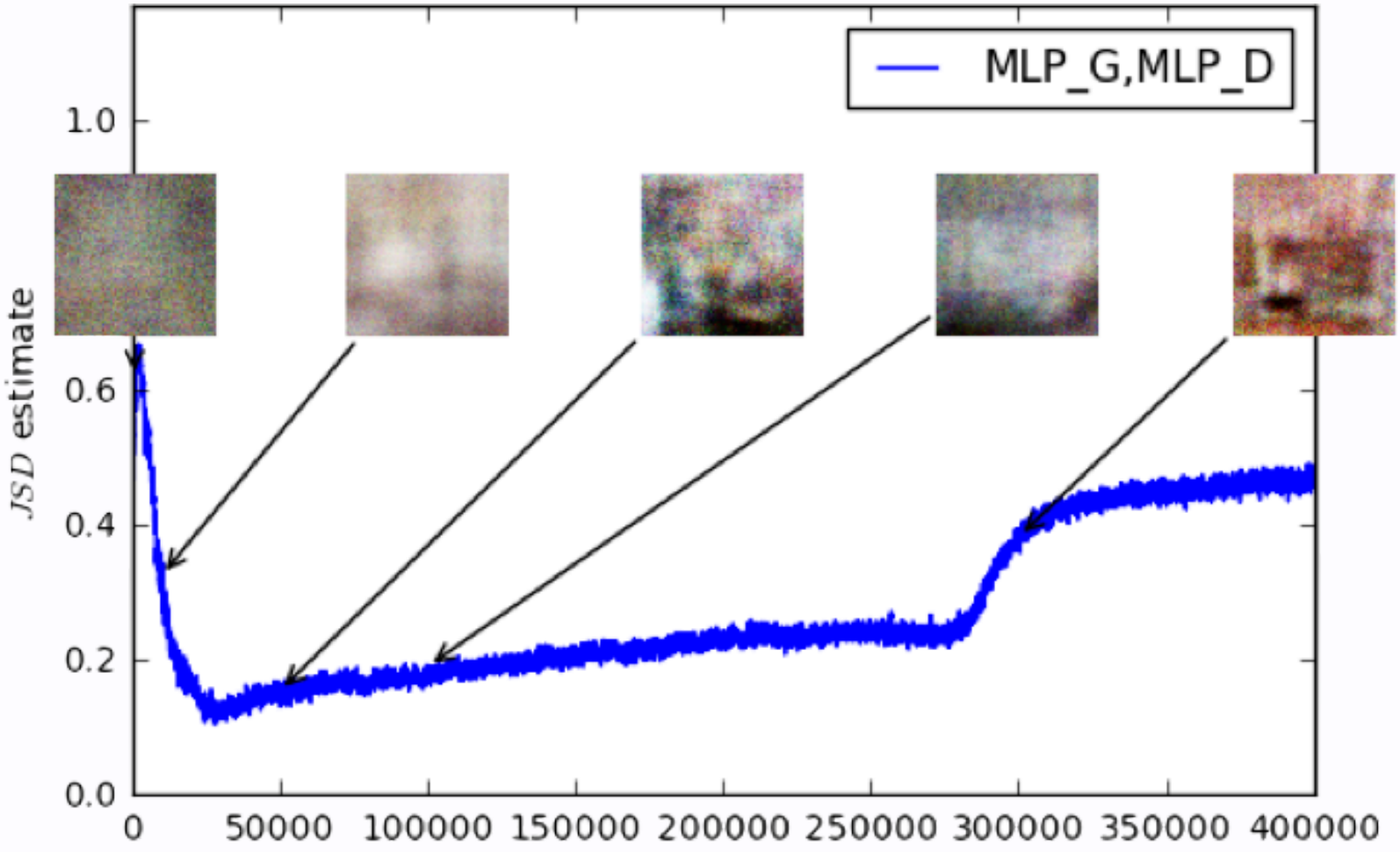
Other losses do not correlate with sample quality



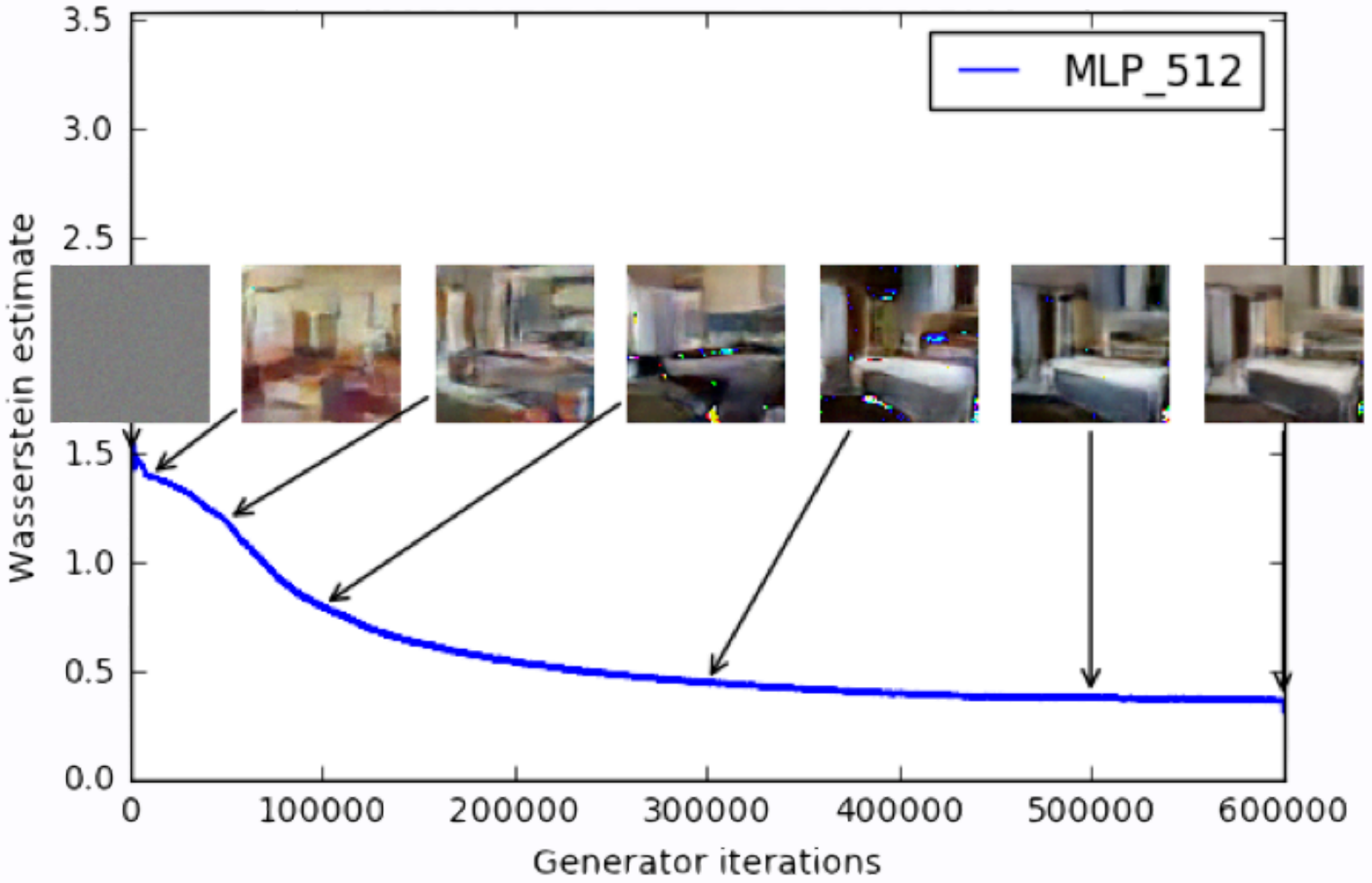
Other losses do not correlate with sample quality



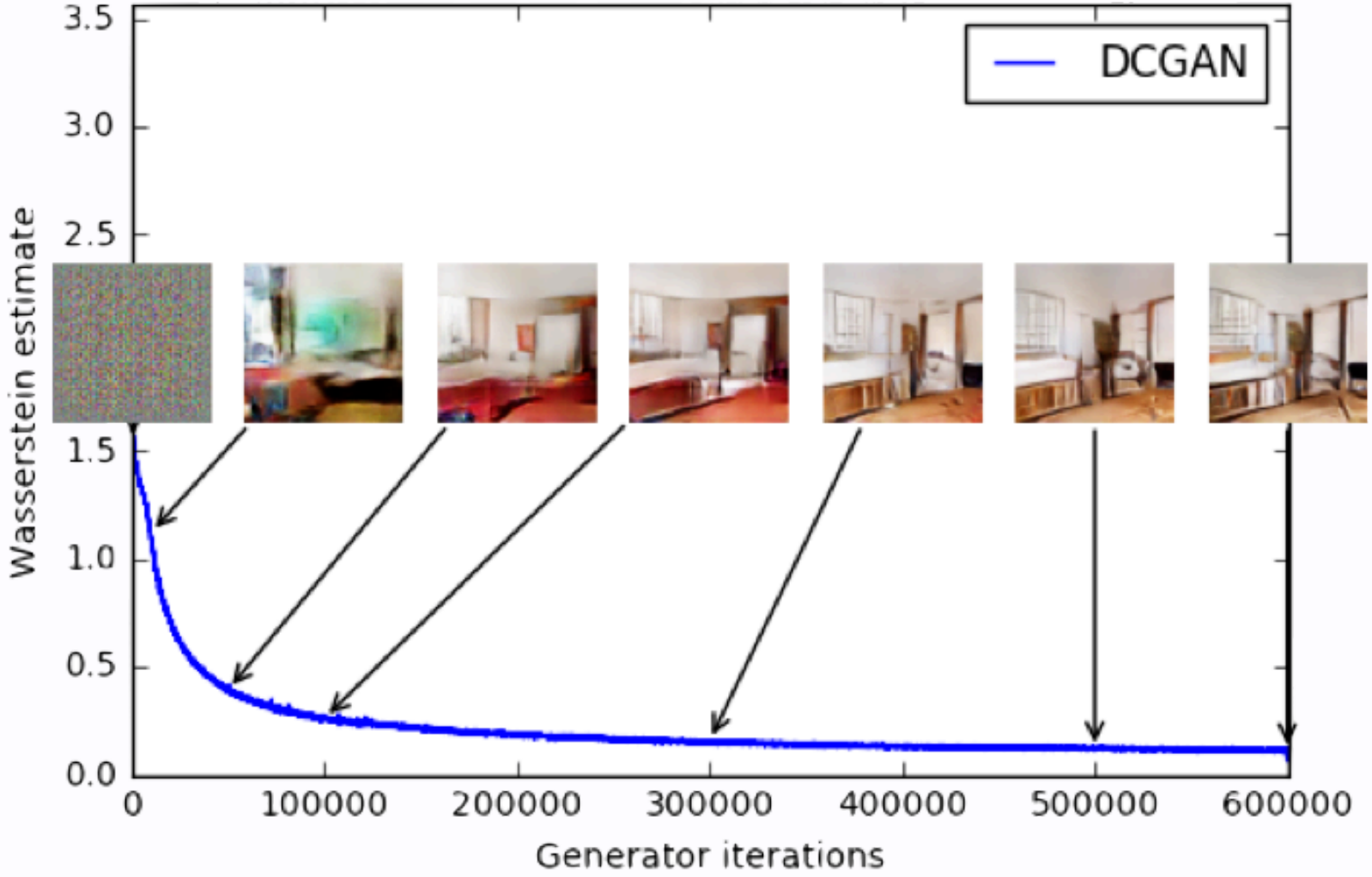
Other losses do not correlate with sample quality



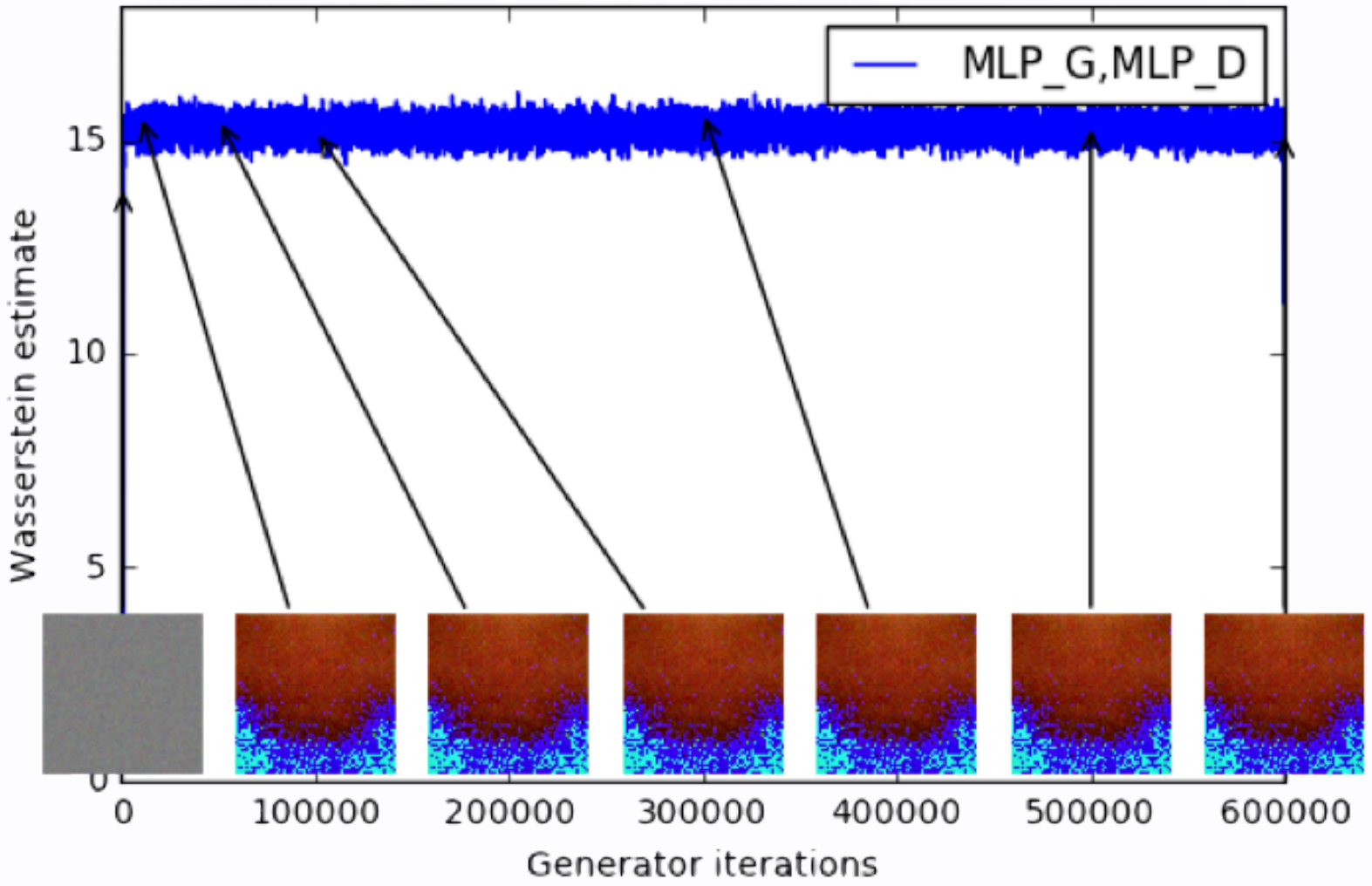
WGAN loss correlates with sample quality



WGAN loss correlates with sample quality



WGAN loss correlates with sample quality



WGANs advantages

One of the advantages of WGANs is that the discriminator can (and should) be trained till optimality. When the critic is trained to completion, it simply provides a loss to the generator that we can train as any other neural network.

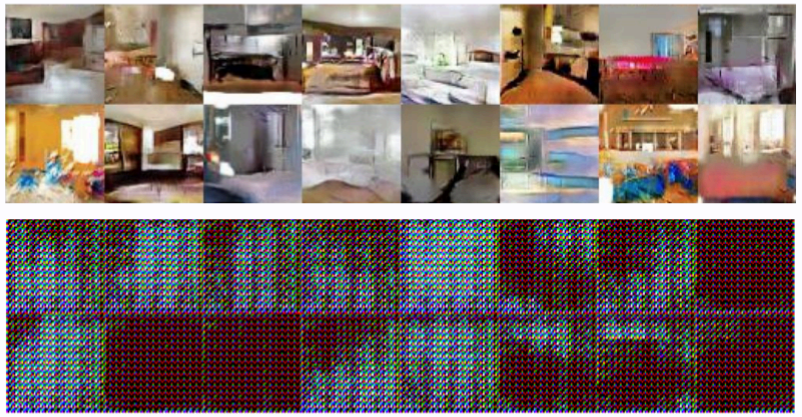
Empirical evidence shows that **WGANs are much more robust than GANs when one varies the generator**.

WGANs advantages

- (a) WGAN vs DCGAN (both good)
- (b) removed the batch normalization layer
- (c) no convolutional architectures (significant degree of model collapse)

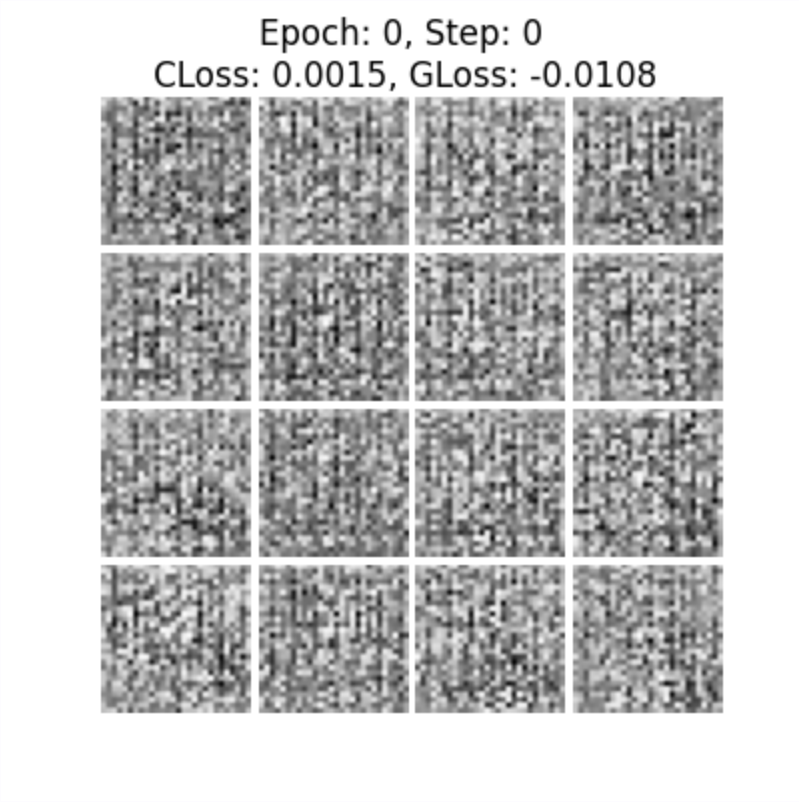
(a)

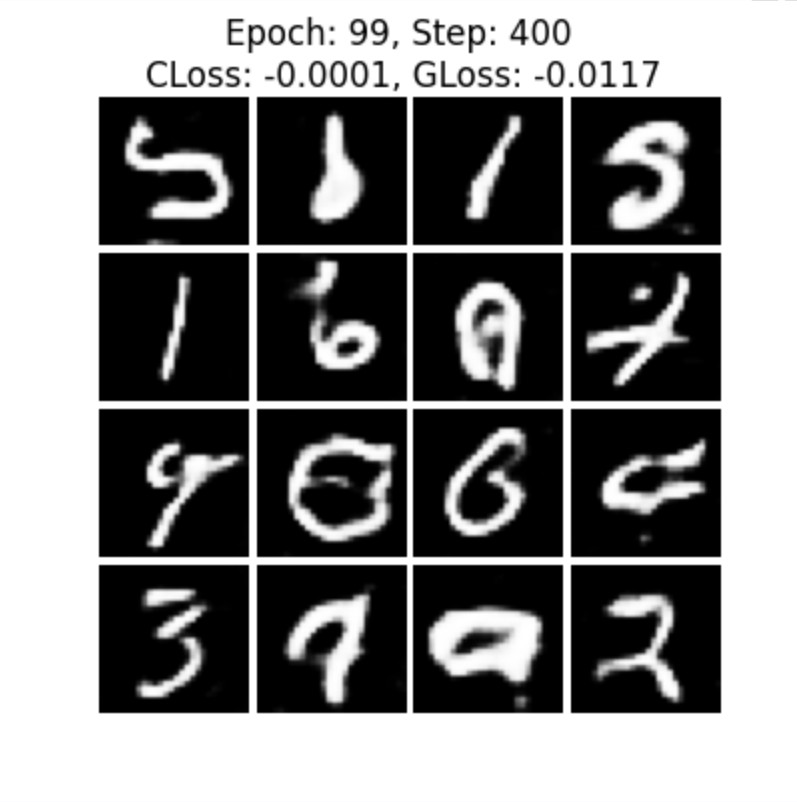
(b)



(c)

WGANs learning from MNIST





Video

Bibliography

- [1] Goodfellow, I. (2016). *NIPS 2016 tutorial: Generative adversarial networks*. arXiv preprint arXiv:1701.00160.
- [2] Lotter, W., Kreiman, G., & Cox, D. (2015). *Unsupervised learning of visual structure using predictive generative networks*. arXiv preprint arXiv:1511.06380.
- [3] Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., ... & Shi, W. (2017). *Photo-realistic single image super-resolution using a generative adversarial network*. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 4681-4690).

- [4] Zhu, J. Y., Krähenbühl, P., Shechtman, E., & Efros, A. A. (2016, October). *Generative visual manipulation on the natural image manifold*. In European conference on computer vision (pp. 597-613). Springer, Cham.
- [5] Brock, A., Lim, T., Ritchie, J. M., & Weston, N. (2016). *Neural photo editing with introspective adversarial networks*. arXiv preprint arXiv:1609.07093.
- [6] Ian Goodfellow (2019) Slides of *Adversarial Machine Learning*. ICLR 2019. https://www.iangoodfellow.com/slides/2019-05-07.pdf
- [7] Arjovsky, M., Chintala, S., & Bottou, L. (2017). *Wasserstein GAN*. arXiv preprint arXiv:1701.07875.