Deep Learning

Lecture 6: Generative adversarial networks

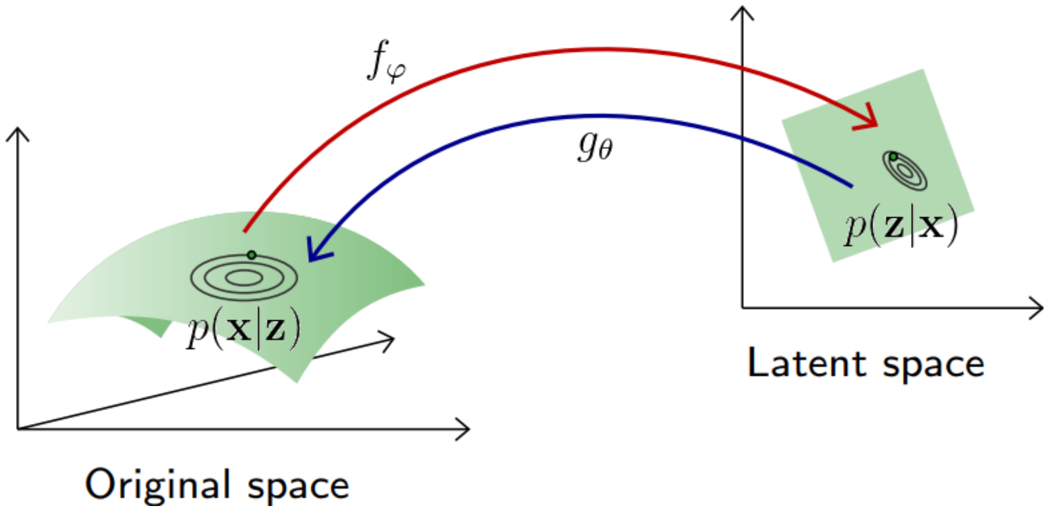
Gilles Louppe
g.louppe@uliege.be

LIÈGE université

Outline

Goals: Learn models of the data itself.

- Generative models (lecture 5)
- Variational inference (lecture 5)
- Variational auto-encoders (lecture 5)
- Generative adversarial networks



Generative adversarial networks

A STEVEN SPIELBERG FILM
catch me
if you can

Generative adversarial networks

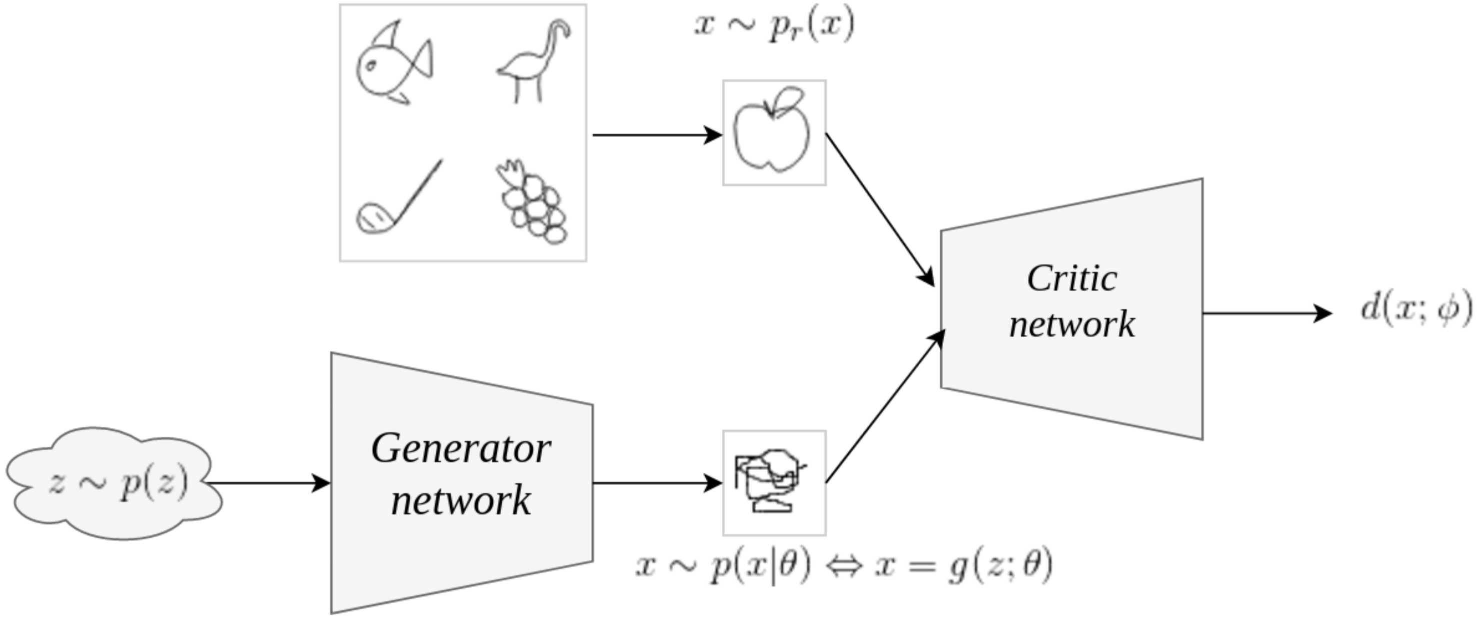
The main idea of generative adversarial networks (GANs) is to express the task of learning a generative model as a two-player zero-sum game between two networks.

- The first network is a generator $g(\cdot;\theta) : \mathcal{Z} \to \mathcal{X}$, mapping a latent space equipped with a prior distribution $p(\mathbf{z})$ to the data space, thereby inducing a distribution

$$\mathbf{x} \sim p(\mathbf{x};\theta) \Leftrightarrow \mathbf{z} \sim p(\mathbf{z}), \mathbf{x} = g(\mathbf{z};\theta).$$

- The second network $d(\cdot;\phi) : \mathcal{X} \to [0,1]$ is a classifier trained to distinguish between true samples $\mathbf{x} \sim p_r(\mathbf{x})$ and generated samples $\mathbf{x} \sim p(\mathbf{x};\theta)$.

The central mechanism will be to use supervised learning to guide the learning of the generative model.


$x \sim p_r(x)$
Critic network
$d(x;\phi)$
$z \sim p(z)$
Generator network
$x \sim p(x|\theta) \Leftrightarrow x = g(z;\theta)$

Game analysis

Consider a generator g fixed at θ. Given a set of observations

$$\mathbf{x}_i \sim p_r(\mathbf{x}), i = 1, ..., N,$$

we can generate a two-class dataset

$$\mathbf{d} = \{(\mathbf{x}_1, 1), ..., (\mathbf{x}_N, 1), (g(\mathbf{z}_1; \theta), 0), ..., (g(\mathbf{z}_N; \theta), 0))\}.$$

The best classifier d is obtained by minimizing the cross-entropy

$$\begin{aligned}\mathcal{L}(\phi) &= -\frac{1}{2N}\left(\sum_{i=1}^{N}\left[\log d(\mathbf{x}_i; \phi)\right] + \sum_{i=1}^{N}\left[\log(1 - d(g(\mathbf{z}_i; \theta); \phi))\right]\right) \\ &\approx -\frac{1}{2}\left(\mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})}\left[\log d(\mathbf{x}; \phi)\right] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\log(1 - d(g(\mathbf{z}; \theta); \phi))\right]\right)\end{aligned}$$

with respect to ϕ.

Following Goodfellow et al (2014), let us define the value function

$$V(\phi, \theta) = \mathbb{E}_{\mathbf{x} \sim p_r(\mathbf{x})} \left[\log d(\mathbf{x}; \phi) \right] + \mathbb{E}_{\mathbf{z} \sim p(\mathbf{z})} \left[\log(1 - d(g(\mathbf{z}; \theta); \phi)) \right].$$

Then,

- $V(\phi, \theta)$ is high if d is good at recognizing true from generated samples.
- If d is the best classifier given g, and if V is high, then this implies that the generator is bad at reproducing the data distribution.
- Conversely, g will be a good generative model if V is low when d is a perfect opponent.

Therefore, the ultimate goal is

$$\theta^* = \arg \min_\theta \max_\phi V(\phi, \theta).$$

For a generator g fixed at θ, the classifier d with parameters ϕ^*_θ is optimal if and only if

$$\forall \mathbf{x}, d(\mathbf{x}; \phi^*_\theta) = \frac{p_r(\mathbf{x})}{p(\mathbf{x};\theta) + p_r(\mathbf{x})}.$$

Therefore,

$$\begin{aligned}
&\min_\theta \max_\phi V(\phi, \theta) = \min_\theta V(\phi^*_\theta, \theta) \\
&= \min_\theta \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})}\left[\log \frac{p_r(\mathbf{x})}{p(\mathbf{x};\theta)+p_r(\mathbf{x})}\right] + \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x};\theta)}\left[\log \frac{p(\mathbf{x};\theta)}{p(\mathbf{x};\theta)+p_r(\mathbf{x})}\right] \\
&= \min_\theta \text{KL}\left(p_r(\mathbf{x}) || \frac{p_r(\mathbf{x}) + p(\mathbf{x};\theta)}{2}\right) \\
&\qquad\qquad + \text{KL}\left(p(\mathbf{x};\theta) || \frac{p_r(\mathbf{x}) + p(\mathbf{x};\theta)}{2}\right) - \log 4 \\
&= \min_\theta 2\,\text{JSD}(p_r(\mathbf{x}) || p(\mathbf{x};\theta)) - \log 4
\end{aligned}$$

where JSD is the Jensen-Shannon divergence.

In summary, solving the minimax problem

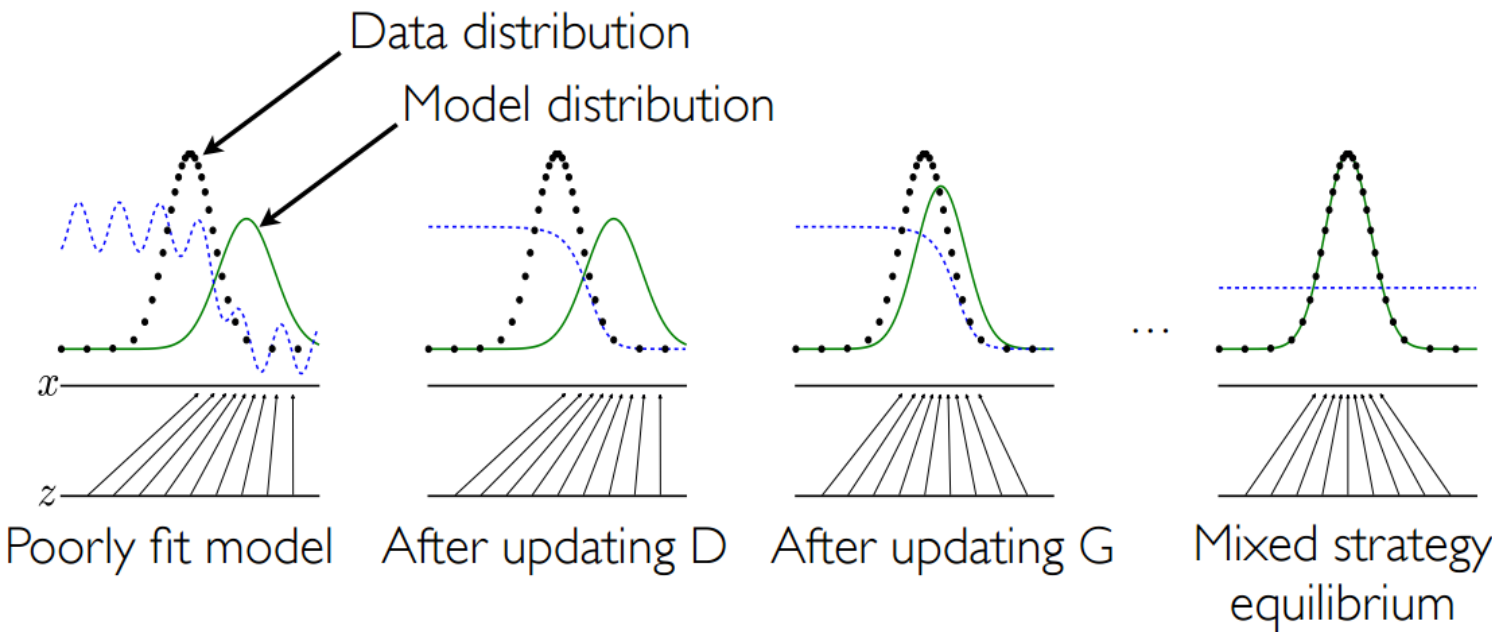
$$\theta^* = \arg\min_\theta \max_\phi V(\phi, \theta)$$

is equivalent to

$$\theta^* = \arg\min_\theta \text{JSD}(p_r(\mathbf{x})||p(\mathbf{x};\theta)).$$

Since $\text{JSD}(p_r(\mathbf{x})||p(\mathbf{x};\theta))$ is minimum if and only if $p_r(\mathbf{x}) = p(\mathbf{x};\theta)$, this proves that the minimax solution corresponds to a generative model that perfectly reproduces the true data distribution.

Learning process



(Goodfellow et al, 2014)

Alternating SGD

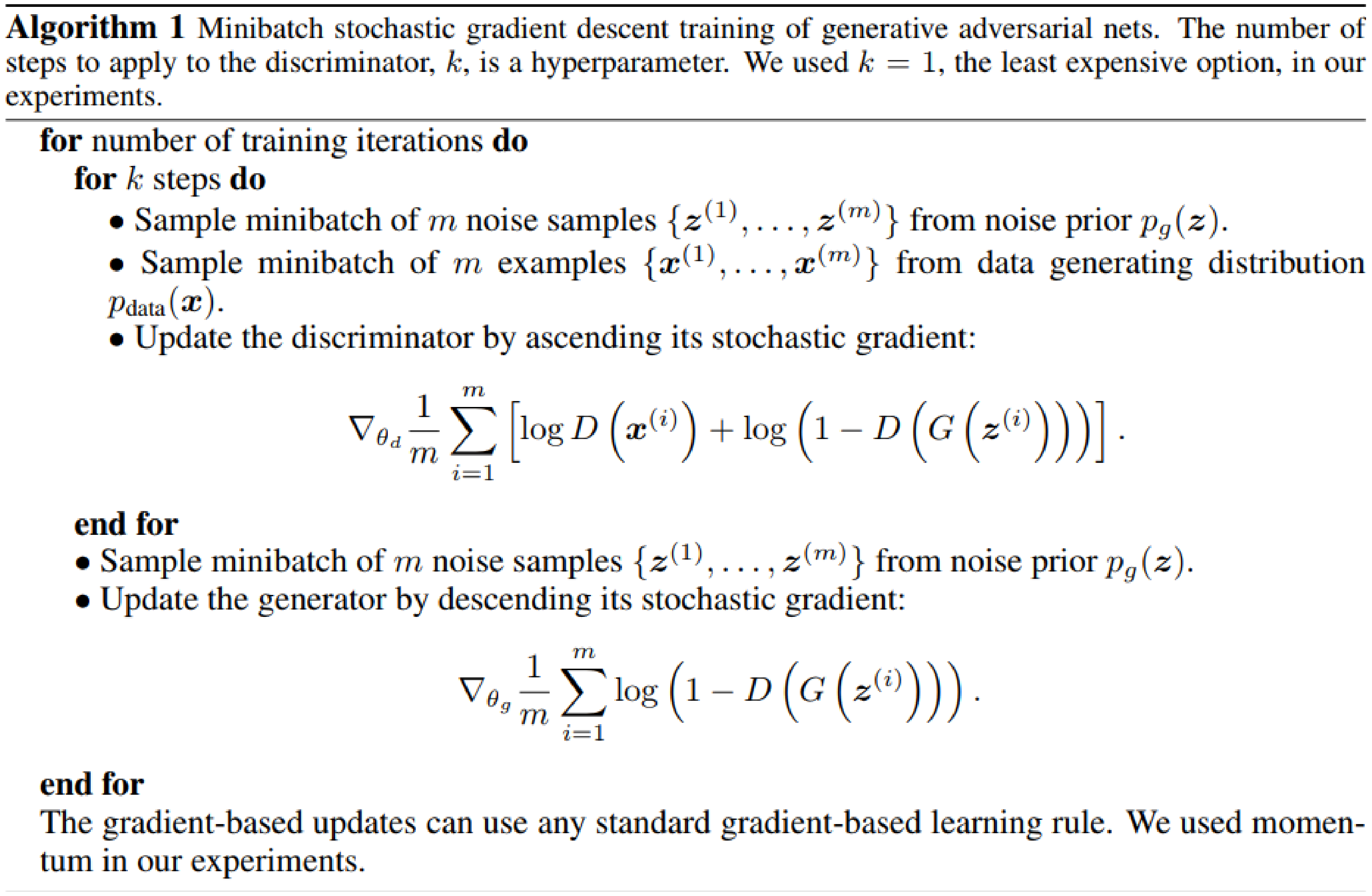
In practice, the minimax solution is approximated using alternating stochastic gradient descent, for which gradients

$$\nabla_\phi V(\phi,\theta) = \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})}\left[\nabla_\phi \log d(\mathbf{x};\phi)\right] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\nabla_\phi \log(1-d(g(\mathbf{z};\theta);\phi))\right],$$
$$\nabla_\theta V(\phi,\theta) = \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\nabla_\theta \log(1-d(g(\mathbf{z};\theta);\phi))\right],$$

are approximated using Monte Carlo integration.

These noisy estimates can in turn be used alternatively to do gradient descent on θ and gradient ascent on ϕ.

- For one step on θ, we can optionally take k steps on ϕ, since we need the classifier to remain near optimal.
- Note that to compute $\nabla_\theta V(\phi,\theta)$, it is necessary to backprop all the way through d before computing the partial derivatives with respect to g's internals.



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 $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \dots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:
        $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$
  end for
  • Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:
        $\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$
end for
```

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

(Goodfellow et al, 2014)

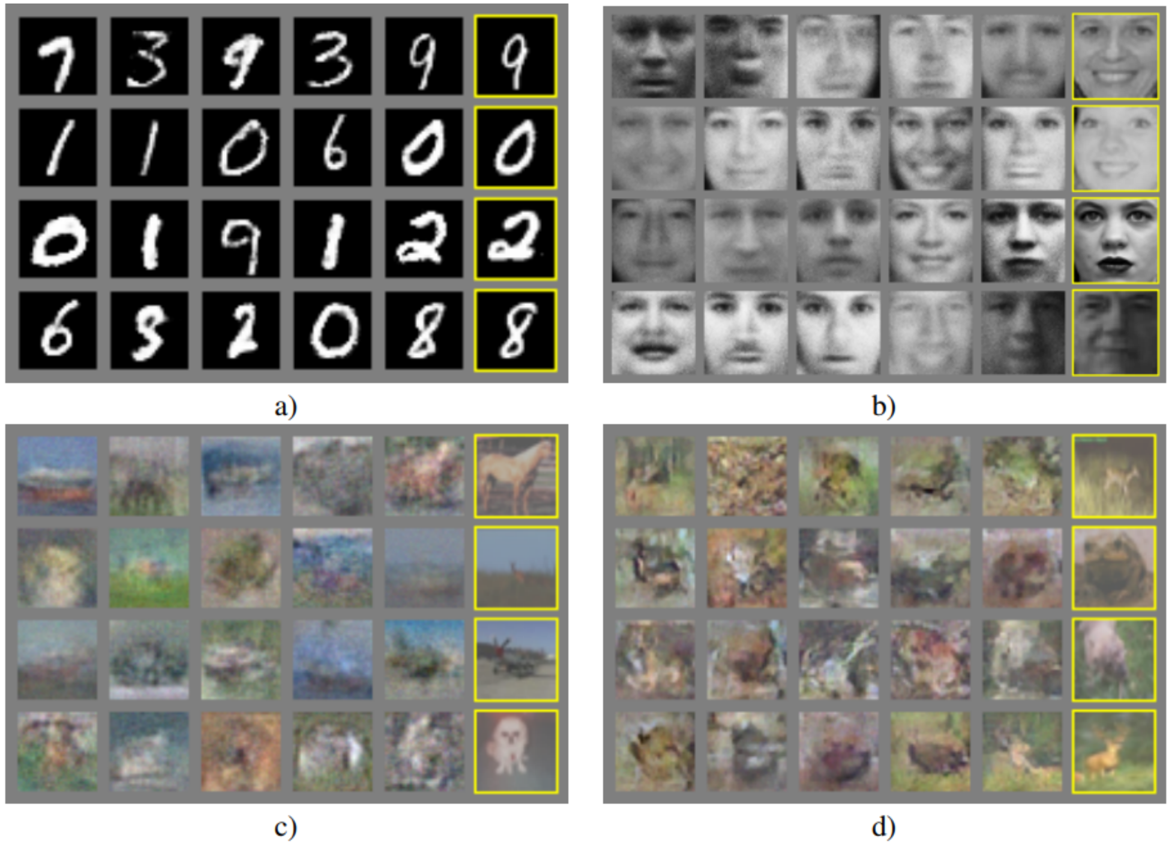


Figure 2: Visualization of samples from the model. Rightmost column shows the nearest training example of the neighboring sample, in order to demonstrate that the model has not memorized the training set. Samples are fair random draws, not cherry-picked. Unlike most other visualizations of deep generative 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)

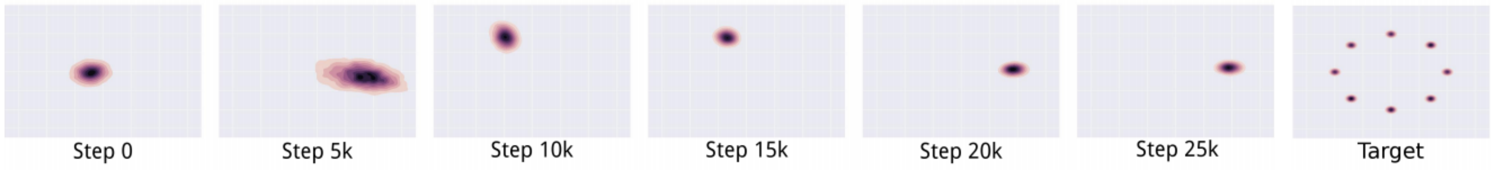
(Goodfellow et al, 2014)

Open problems

Training a standard GAN often results in pathological behaviors:

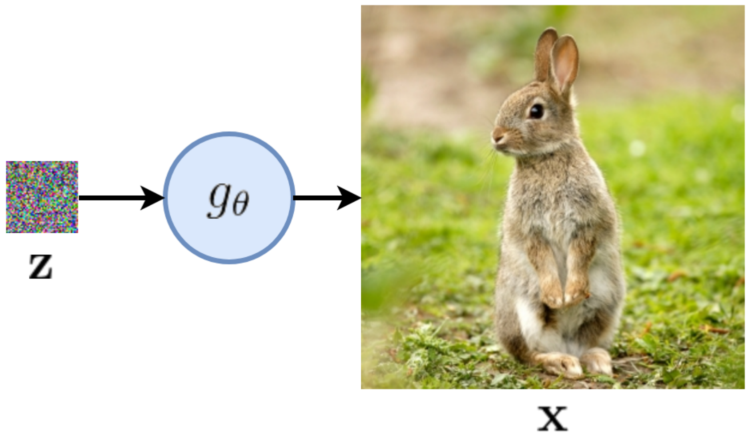
- Oscillations without convergence: contrary to standard loss minimization, alternating stochastic gradient descent has no guarantee of convergence.
- Vanishing gradient: when the classifier d is too good, the value function saturates and we end up with no gradient to update the generator (more on this later).
- Mode collapse: the generator g models very well a small sub-population, concentrating on a few modes of the data distribution.

Performance is also difficult to assess in practice.



Mode collapse (Metz et al, 2016)

Deep convolutional GAN



Deep generative architectures require layers that increase the input dimension, i.e., that go from $\mathbf{z} \in \mathbb{R}^q$ to $\mathbf{x} = g(\mathbf{z}) \in \mathbb{R}^p$, with $p \gg q$.

- This is the opposite of what we did so far with feedforward networks, in which we reduced the dimension of the input to a few values.
- Fully connected layers could be used for that purpose but would face the same limitations as before (spatial specialization, too many parameters).
- Ideally, we would like layers that implement the inverse of convolutional and pooling layers.

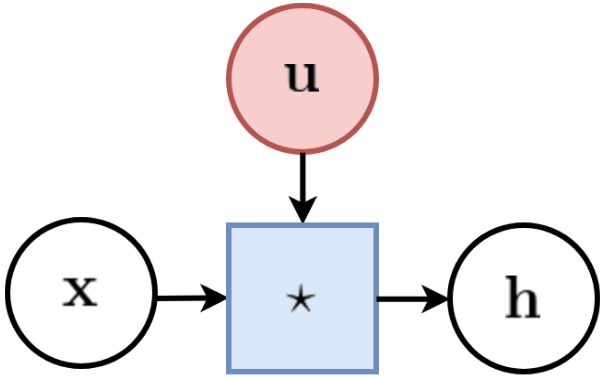
Convolution

For $\mathbf{x} \in \mathbb{R}^{H \times W}$ and convolutional kernel $\mathbf{u} \in \mathbb{R}^{h \times w}$, we defined the discrete convolution $\mathbf{x} \star \mathbf{u}$ as a 2D tensor of size $(H - h + 1) \times (W - w + 1)$ such that

$$(\mathbf{x} \star \mathbf{u})_{j,i} = \sum_{n=0}^{h-1} \sum_{m=0}^{w-1} \mathbf{x}_{j+n,i+m} \mathbf{u}_{n,m}.$$

For example,

$$\begin{pmatrix} 4 & 5 & 8 & 7 \\ 1 & 8 & 8 & 8 \\ 3 & 6 & 6 & 4 \\ 6 & 5 & 7 & 8 \end{pmatrix} \star \begin{pmatrix} 1 & 4 & 1 \\ 1 & 4 & 3 \\ 3 & 3 & 1 \end{pmatrix} = \begin{pmatrix} 122 & 148 \\ 126 & 134 \end{pmatrix}$$


u
x
⋆
h

The convolution operation can be equivalently re-expressed as a single matrix multiplication.

Following the previous example,

- the convolutional kernel $\mathbf{u}$ is rearranged as a sparse Toeplitz circulant matrix, called the convolution matrix:

$$\mathbf{U} = \begin{pmatrix} 1 & 4 & 1 & 0 & 1 & 4 & 3 & 0 & 3 & 3 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 4 & 1 & 0 & 1 & 4 & 3 & 0 & 3 & 3 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 4 & 1 & 0 & 1 & 4 & 3 & 0 & 3 & 3 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 4 & 1 & 0 & 1 & 4 & 3 & 0 & 3 & 3 & 1 \end{pmatrix}$$

- the input $\mathbf{x}$ is flattened row by row, from top to bottom:

$$v(\mathbf{x}) = \begin{pmatrix} 4 & 5 & 8 & 7 & 1 & 8 & 8 & 8 & 3 & 6 & 6 & 4 & 6 & 5 & 7 & 8 \end{pmatrix}^T$$

Then,

$$\mathbf{U}v(\mathbf{x}) = \begin{pmatrix} 122 & 148 & 126 & 134 \end{pmatrix}^T$$

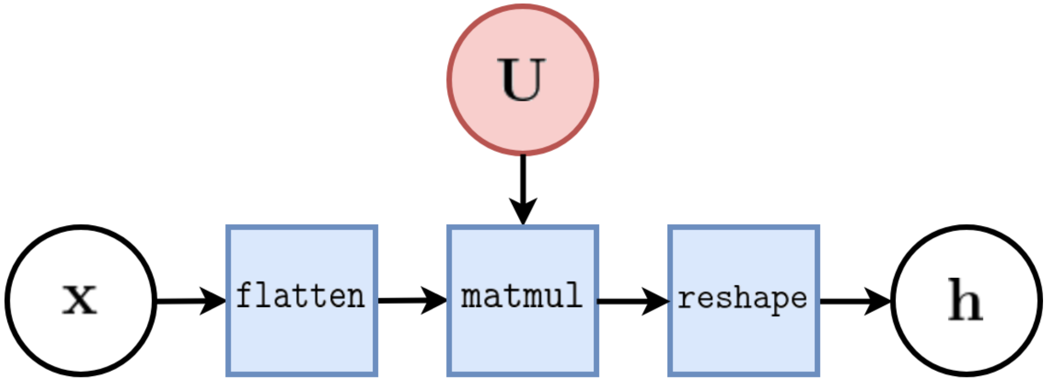
which we can reshape to a 2×2 matrix to obtain $\mathbf{x} \star \mathbf{u}$.

The same procedure generalizes to $\mathbf{x} \in \mathbb{R}^{H\times W}$ and convolutional kernel $\mathbf{u} \in \mathbf{R}^{h\times w}$, such that:

- the convolutional kernel is rearranged as a sparse Toeplitz circulant matrix $\mathbf{U}$ of shape $(H-h+1)(W-w+1) \times HW$ where
  - each row i identifies an element of the output feature map,
  - each column j identifies an element of the input feature map,
  - the value $\mathbf{U}_{i,j}$ corresponds to the kernel value the element j is multiplied with in output i;
- the input $\mathbf{x}$ is flattened into a column vector $v(\mathbf{x})$ of shape $HW \times 1$;
- the output feature map $\mathbf{x} \star \mathbf{u}$ is obtained by reshaping the $(H-h+1)(W-w+1) \times 1$ column vector $\mathbf{U}v(\mathbf{x})$ as a $(H-h+1) \times (W-w+1)$ matrix.

Therefore, a convolutional layer is a special case of a fully connected layer:

$$\mathbf{h} = \mathbf{x} \star \mathbf{u} \Leftrightarrow v(\mathbf{h}) = \mathbf{U}v(\mathbf{x}) \Leftrightarrow v(\mathbf{h}) = \mathbf{W}^T v(\mathbf{x})$$


U
x
flatten
matmul
reshape
h

In a fully connected layer $\mathbf{h} = \mathbf{W}^T\mathbf{x}$, the partial derivatives with respect to the layer inputs are

$$\frac{\partial \mathbf{h}}{\partial \mathbf{x}} = \mathbf{W}.$$

Since a convolutional layer $\mathbf{h} = \mathbf{x} \star \mathbf{u}$ can be expressed as a fully connected layer $v(\mathbf{h}) = \mathbf{U}v(\mathbf{x})$, the partial derivatives with respect to its inputs are

$$\frac{\partial v(\mathbf{h})}{\partial v(\mathbf{x})} = \mathbf{U}^T.$$

The backward pass of convolutional layer therefore amounts to multiplying the loss with $\mathbf{U}^T$ and reshaping appropriately.

- The backward pass takes some q-dimensional vector as input and produces some p-dimensional vector as output, with $q < p$.
- It does so while keeping a connectivity pattern that is compatible with $\mathbf{U}$, by construction.

Transposed convolution

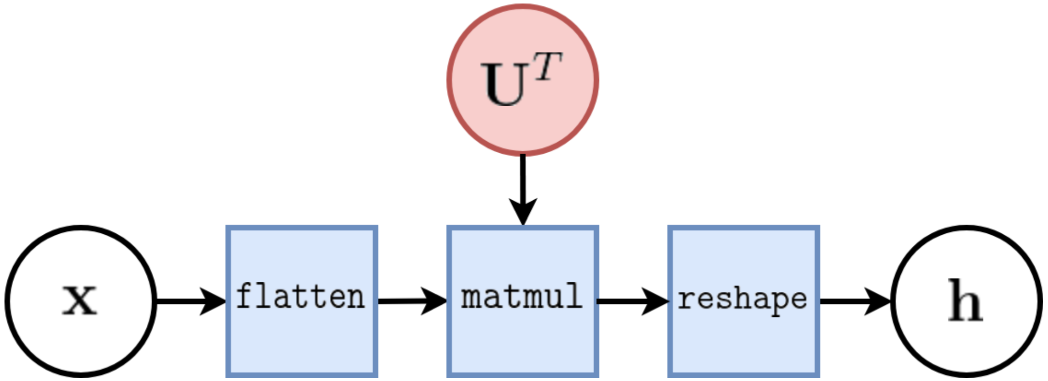
A transposed convolution is a convolution where the implementation of the forward and backward passes are swapped.

Therefore, a transposed convolution can be seen as the gradient of some convolution with respect to its input.

Given a convolutional kernel $\mathbf{u}$,

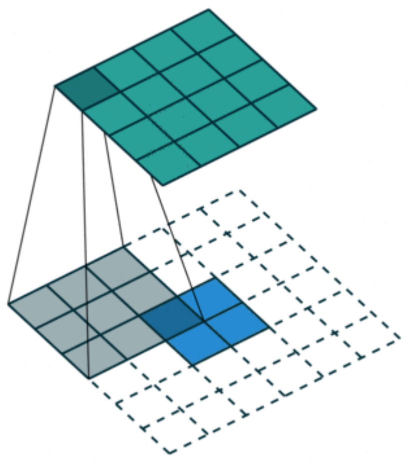
- the forward pass is implemented as $v(\mathbf{h}) = \mathbf{U}^T v(\mathbf{x})$ with appropriate reshaping, thereby effectively up-sampling an input $v(\mathbf{x})$ into a larger one;
- the backward pass is computed by multiplying the loss by $\mathbf{U}$ instead of $\mathbf{U}^T$.

Transposed convolutions are also referred to as fractionally-stride convolutions or deconvolutions (mistakenly).


$\mathbf{U}^T$
$\mathbf{x}$
flatten
matmul
reshape
$\mathbf{h}$

$$\mathbf{U}^T v(\mathbf{x}) = v(\mathbf{h})$$

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 4 & 1 & 0 & 0 \\ 1 & 4 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 \\ 4 & 1 & 4 & 1 \\ 3 & 4 & 1 & 4 \\ 0 & 3 & 0 & 1 \\ 3 & 0 & 1 & 0 \\ 3 & 3 & 4 & 1 \\ 1 & 3 & 3 & 4 \\ 0 & 1 & 0 & 3 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 3 & 3 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 \\ 1 \\ 4 \\ 4 \end{pmatrix} = \begin{pmatrix} 2 \\ 9 \\ 6 \\ 1 \\ 6 \\ 29 \\ 30 \\ 7 \\ 10 \\ 29 \\ 33 \\ 13 \\ 12 \\ 24 \\ 16 \\ 4 \end{pmatrix}$$

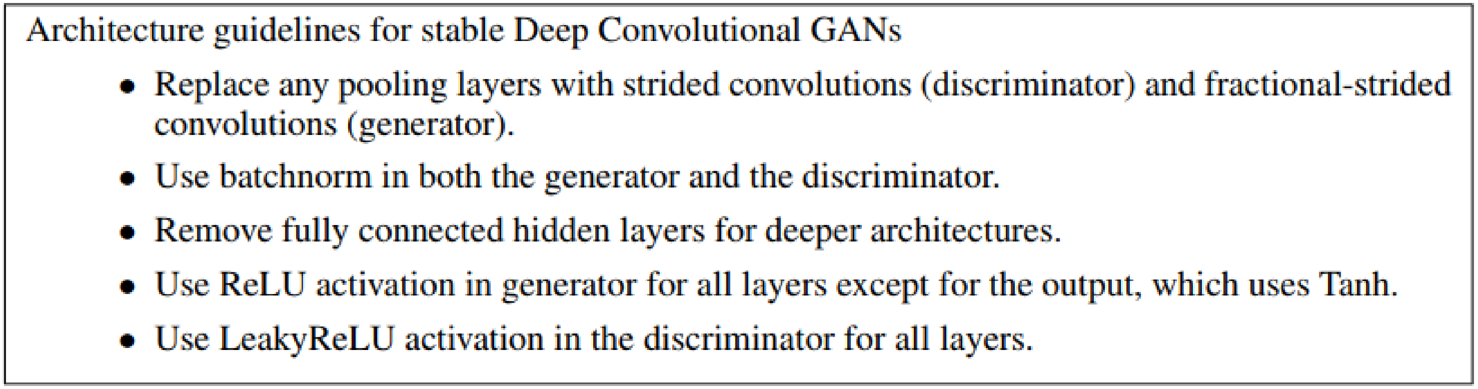


Transposed convolution (no padding, no stride)

Deep convolutional GAN

Given transposed convolutional layers, we are now equipped for building deep convolutional generative models.

Radford et al (2015) identify the following guidelines to ensure stable training:



Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

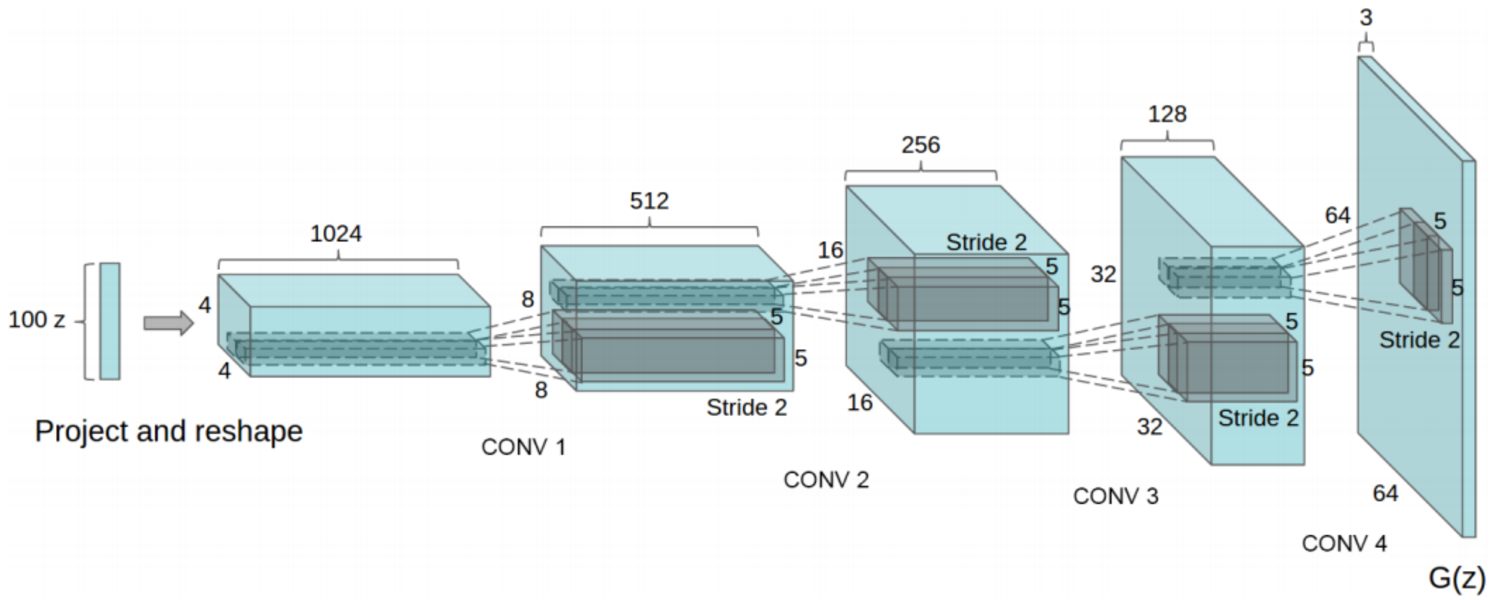


Figure 1: DCGAN generator used for LSUN scene modeling. A 100 dimensional uniform distribution Z is projected to a small spatial extent convolutional representation with many feature maps. A series of four fractionally-strided convolutions (in some recent papers, these are wrongly called deconvolutions)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

The DCGAN generator architecture (Radford et al, 2015)

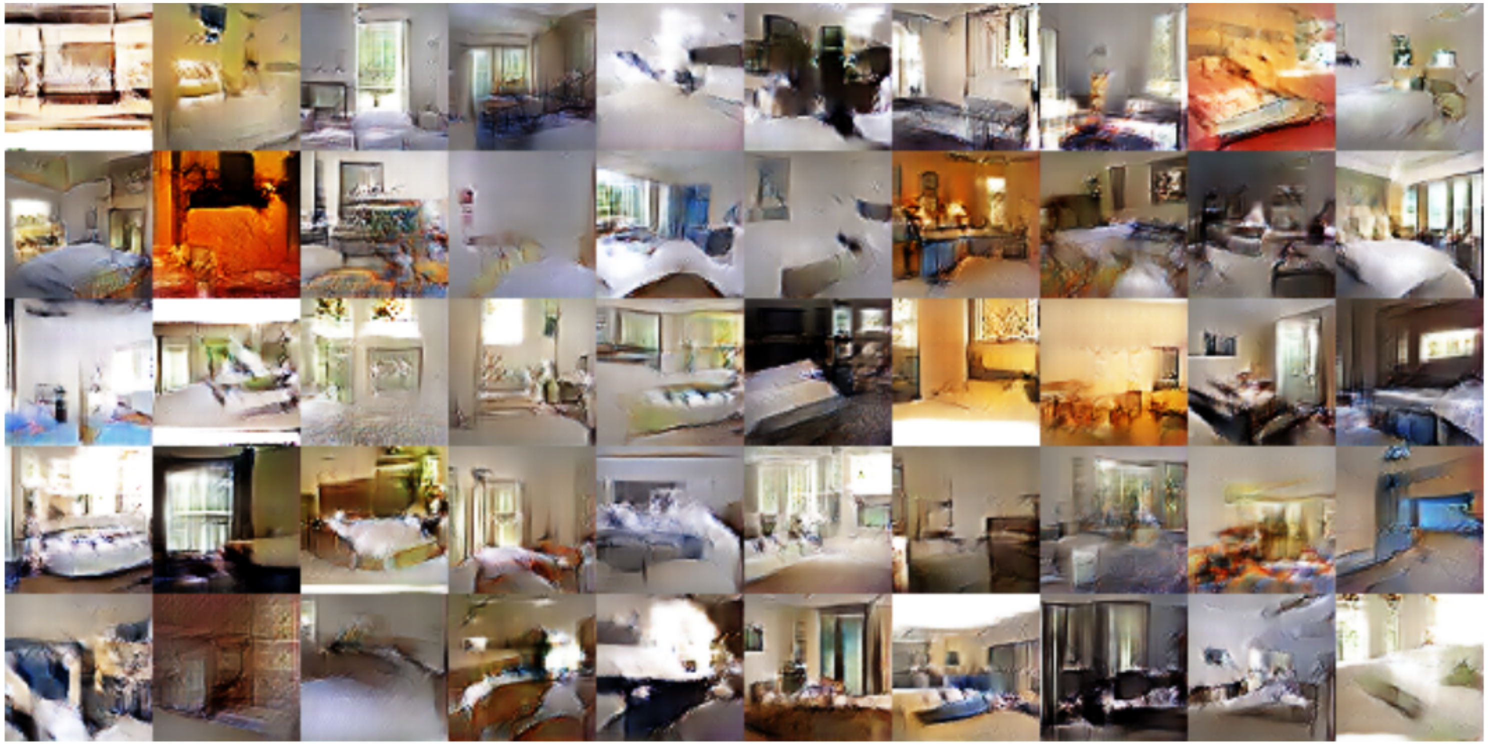


Figure 2: Generated bedrooms after one training pass through the dataset. Theoretically, the model could learn to memorize training examples, but this is experimentally unlikely as we train with a small learning rate and minibatch SGD. We are aware of no prior empirical evidence demonstrating memorization with SGD and a small learning rate.

(Radford et al, 2015)

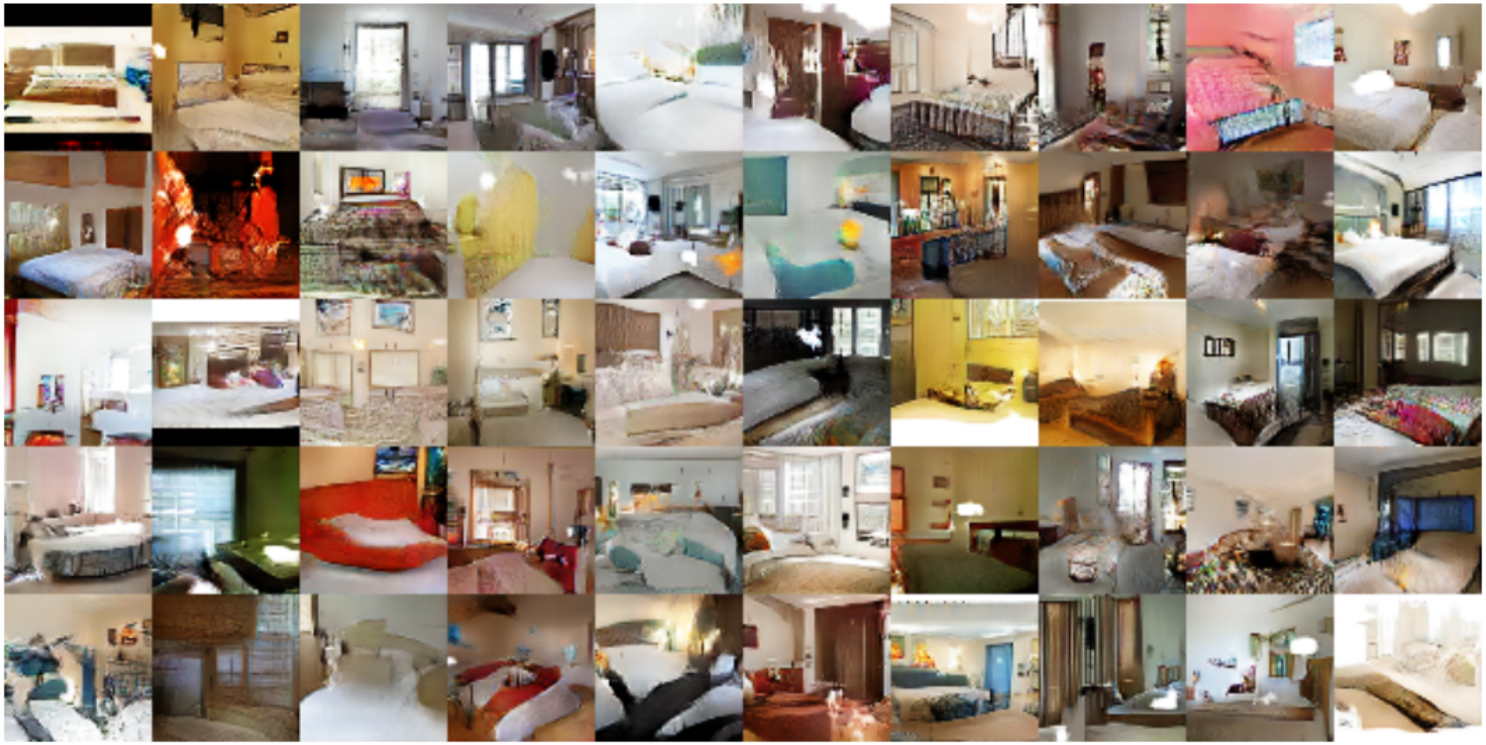
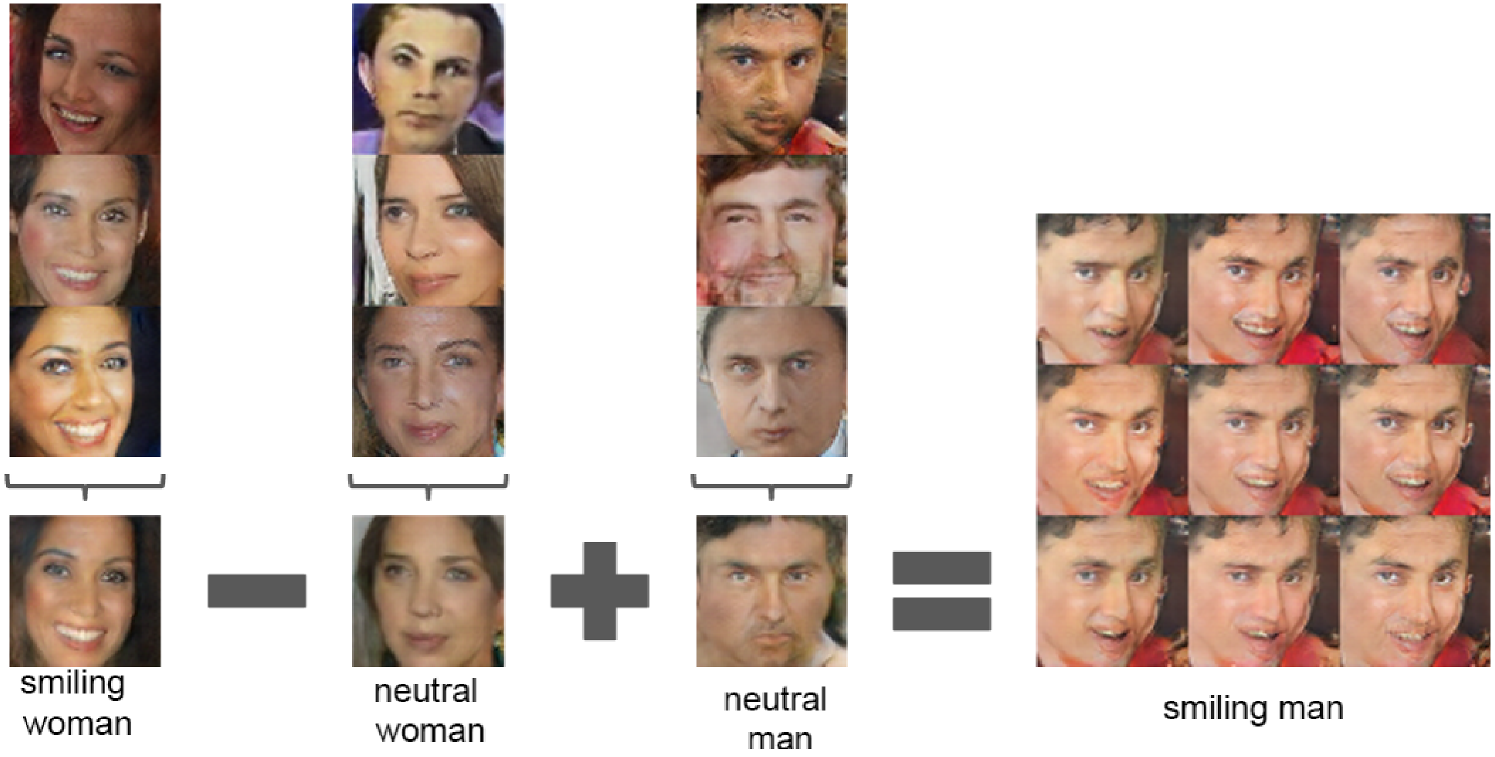


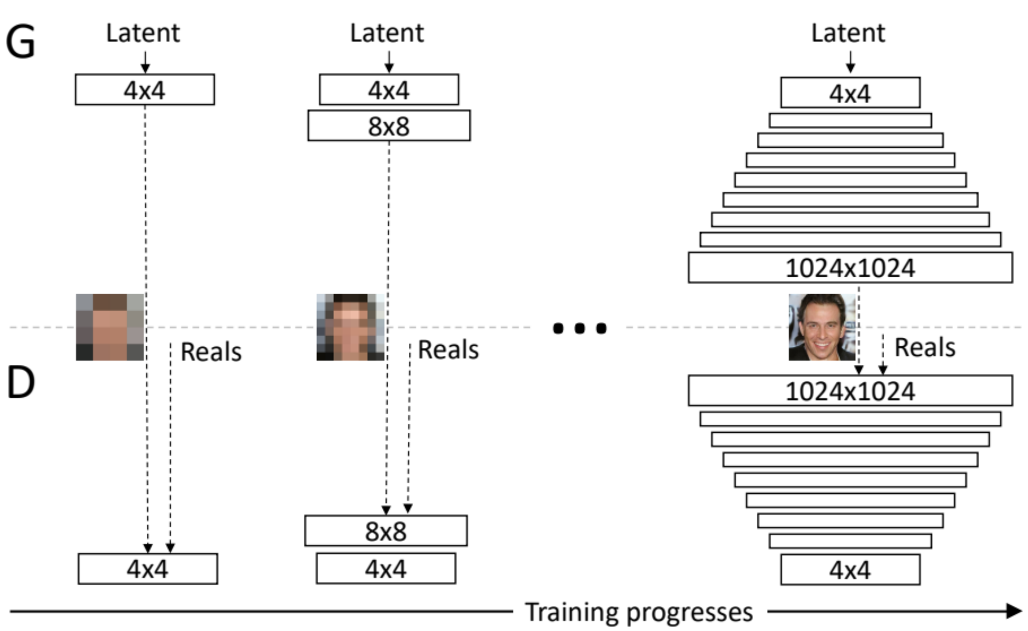
Figure 3: Generated bedrooms after five epochs of training. There appears to be evidence of visual under-fitting via repeated noise textures across multiple samples such as the base boards of some of the beds.

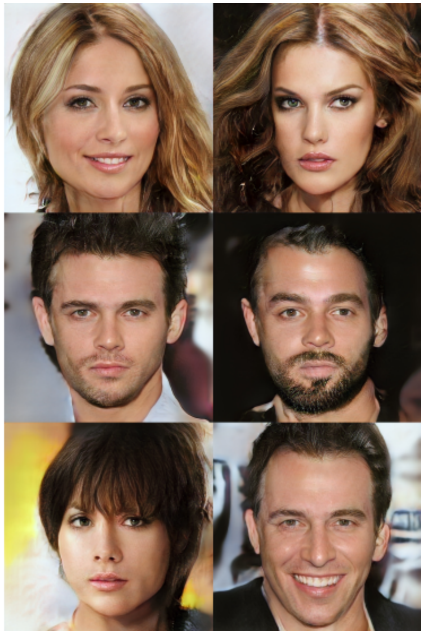
(Radford et al, 2015)



Vector arithmetic in $\mathcal{Z}$-space (Radford et al, 2015)

Progressive growing of GANs





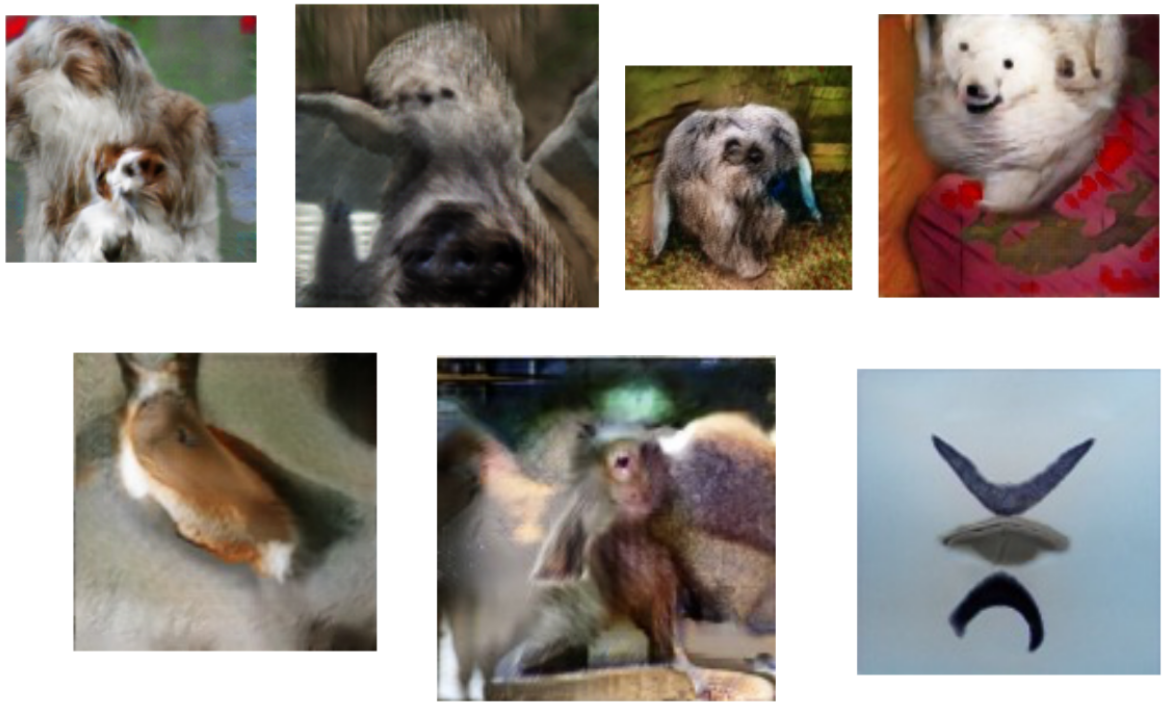
(Karras et al, 2017)

(Karras et al, 2017)

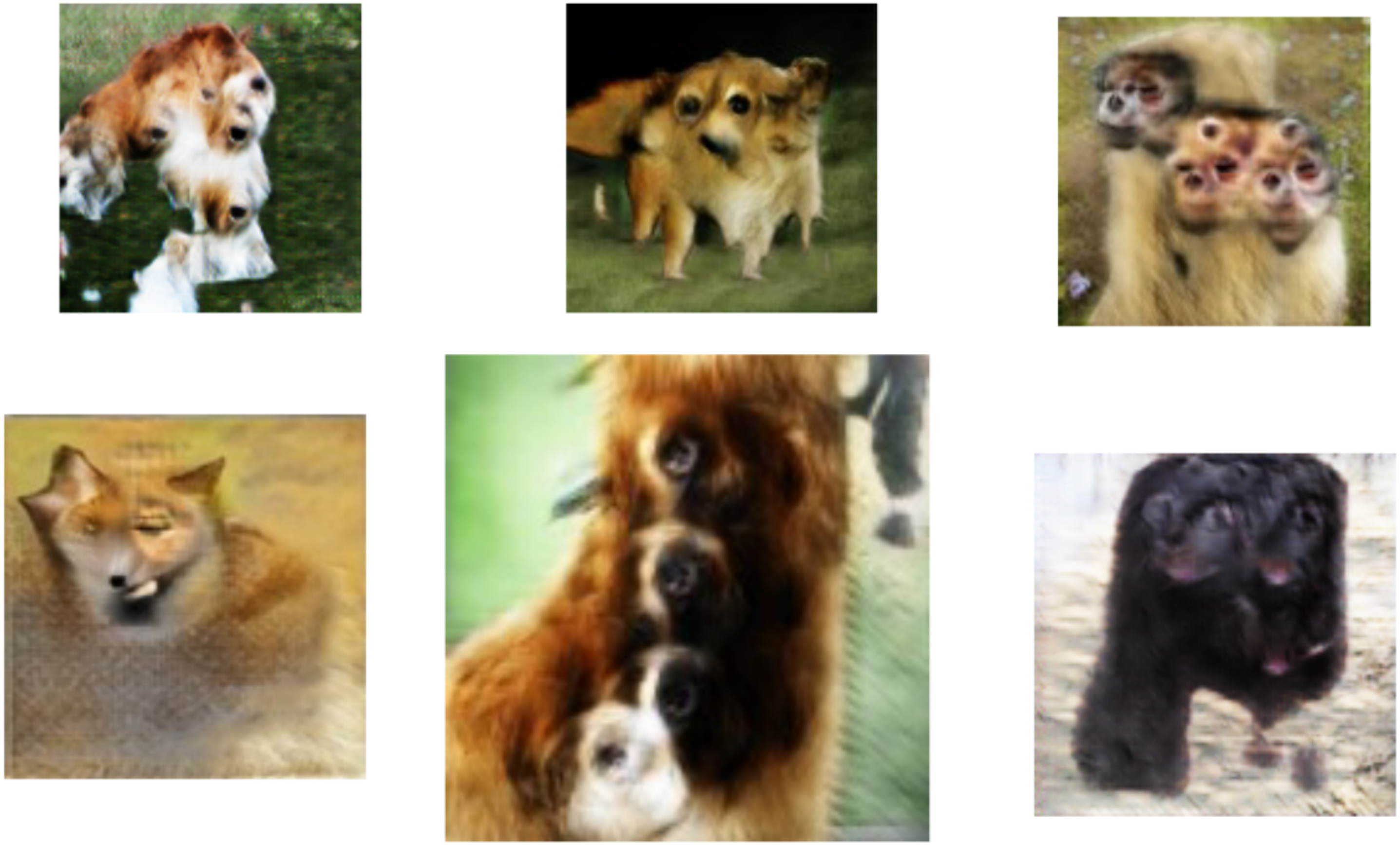
Cabinet of curiosities

While state-of-the-art results are impressive, a close inspection of the fake samples distribution $p(\mathbf{x};\theta)$ often reveals fundamental issues highlighting architectural limitations.

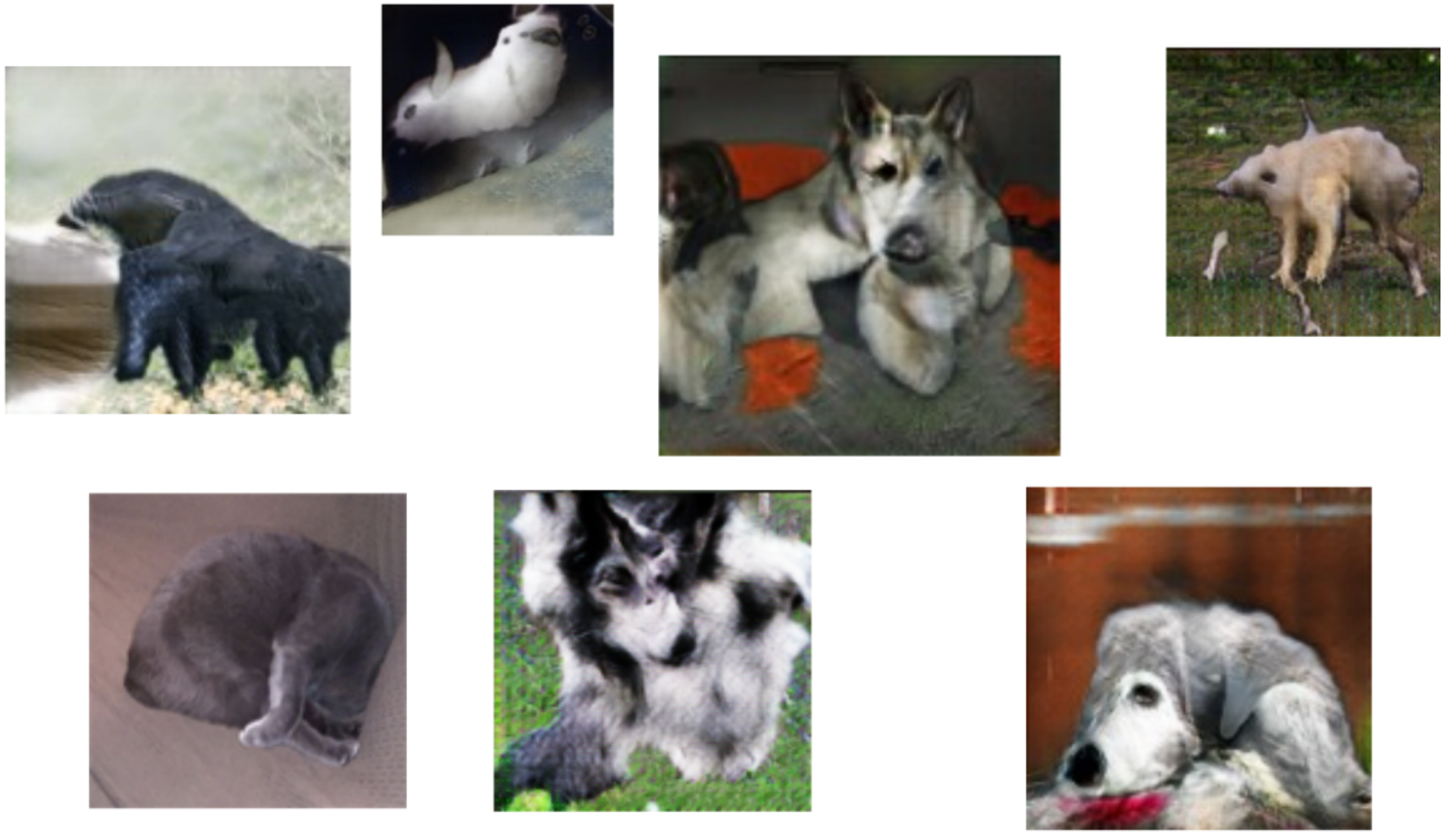
These issues remain an open research problem.



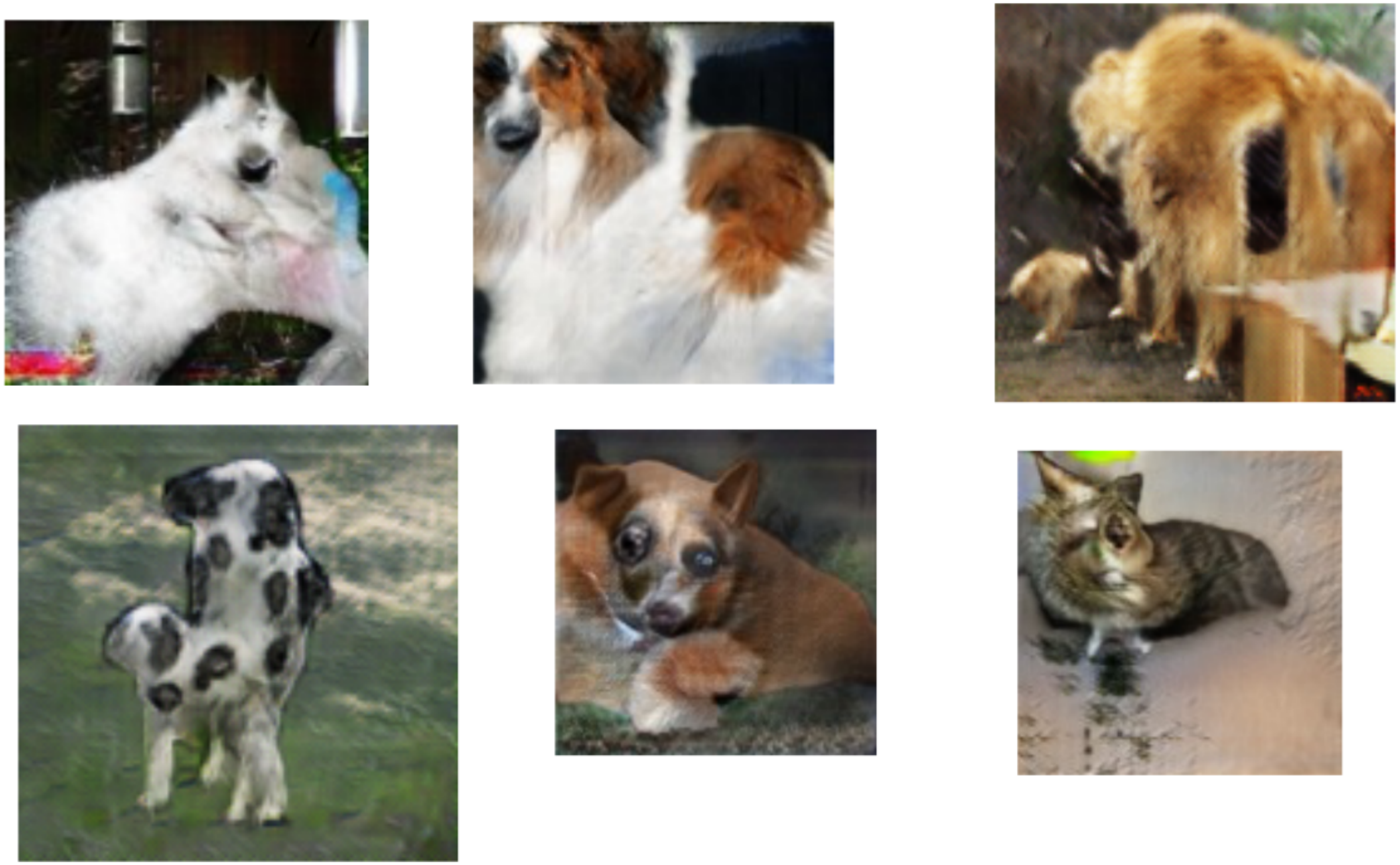
Cherry-picks (Goodfellow, 2016)



Problems with counting (Goodfellow, 2016)



Problems with perspective (Goodfellow, 2016)



Problems with global structures (Goodfellow, 2016)

Wasserstein GAN

Vanishing gradients

For most non-toy data distributions, the fake samples $\mathbf{x} \sim p(\mathbf{x};\theta)$ may be so bad initially that the response of d saturates. At the limit, when d is perfect given the current generator g,

$$\begin{aligned} d(\mathbf{x};\phi) &= 1, \forall \mathbf{x} \sim p_r(\mathbf{x}), \\ d(\mathbf{x};\phi) &= 0, \forall \mathbf{x} \sim p(\mathbf{x};\theta). \end{aligned}$$

Therefore,

$$V(\phi,\theta) = \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})}\left[\log d(\mathbf{x};\phi)\right] + \mathbb{E}_{\mathbf{z}\sim p(\mathbf{z})}\left[\log(1 - d(g(\mathbf{z};\theta);\phi))\right] = 0$$

and $\nabla_\theta V(\phi,\theta) = 0$, thereby halting gradient descent.

Dilemma:

- If d is bad, then g does not have accurate feedback and the loss function cannot represent the reality.
- If d is too good, the gradients drop to 0, thereby slowing down or even halting the optimization.

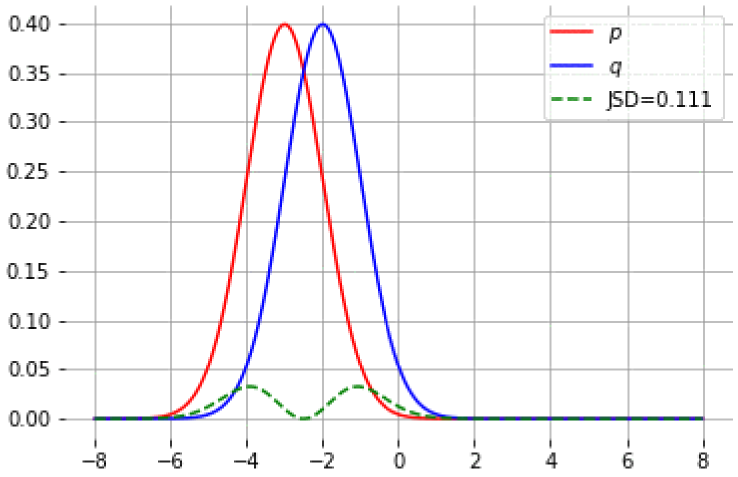
Jensen-Shannon divergence

For any two distributions p and q,

$$0 \leq JSD(p||q) \leq \log 2,$$

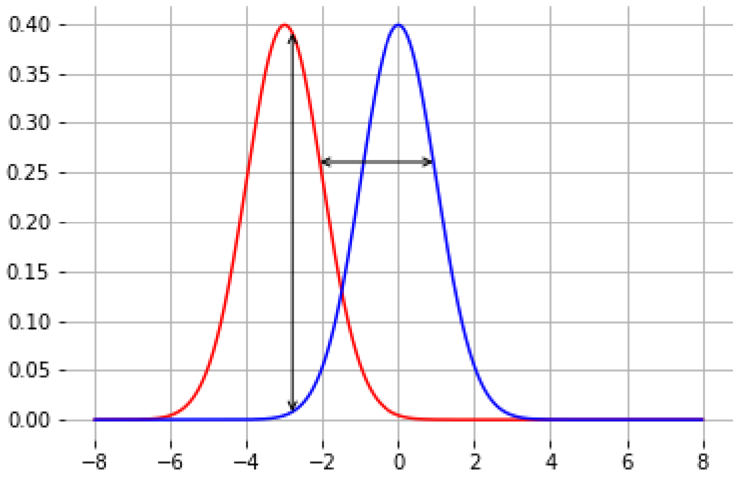
where

- $JSD(p||q) = 0$ if and only if $p = q$,
- $JSD(p||q) = \log 2$ if and only if p and q have disjoint supports.



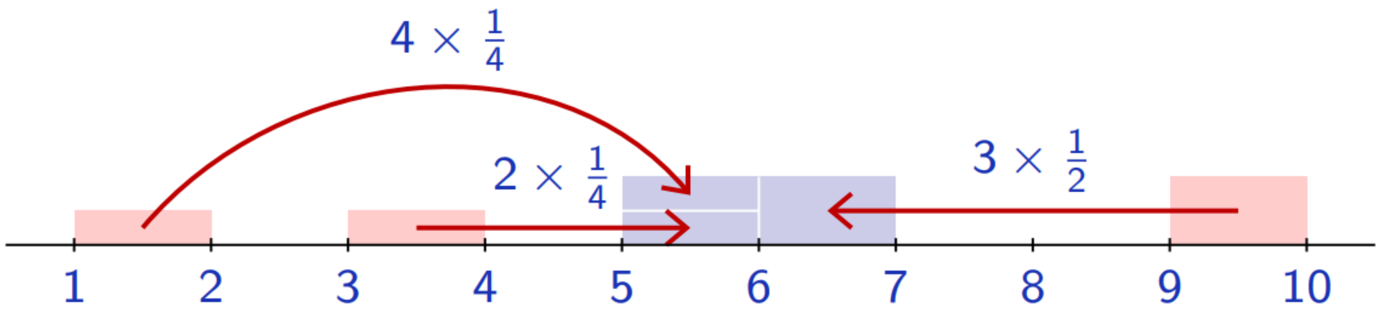
Notice how the Jensen-Shannon divergence poorly accounts for the metric structure of the space.

Intuitively, instead of comparing distributions "vertically", we would like to compare them "horizontally".



Wasserstein distance

An alternative choice is the Earth mover's distance, which intuitively corresponds to the minimum mass displacement to transform one distribution into the other.



- $p = \frac{1}{4}\mathbf{1}_{[1,2]} + \frac{1}{4}\mathbf{1}_{[3,4]} + \frac{1}{2}\mathbf{1}_{[9,10]}$
- $q = \mathbf{1}_{[5,7]}$

Then,

$$\mathrm{W}_1(p,q) = 4 \times \frac{1}{4} + 2 \times \frac{1}{4} + 3 \times \frac{1}{2} = 3$$

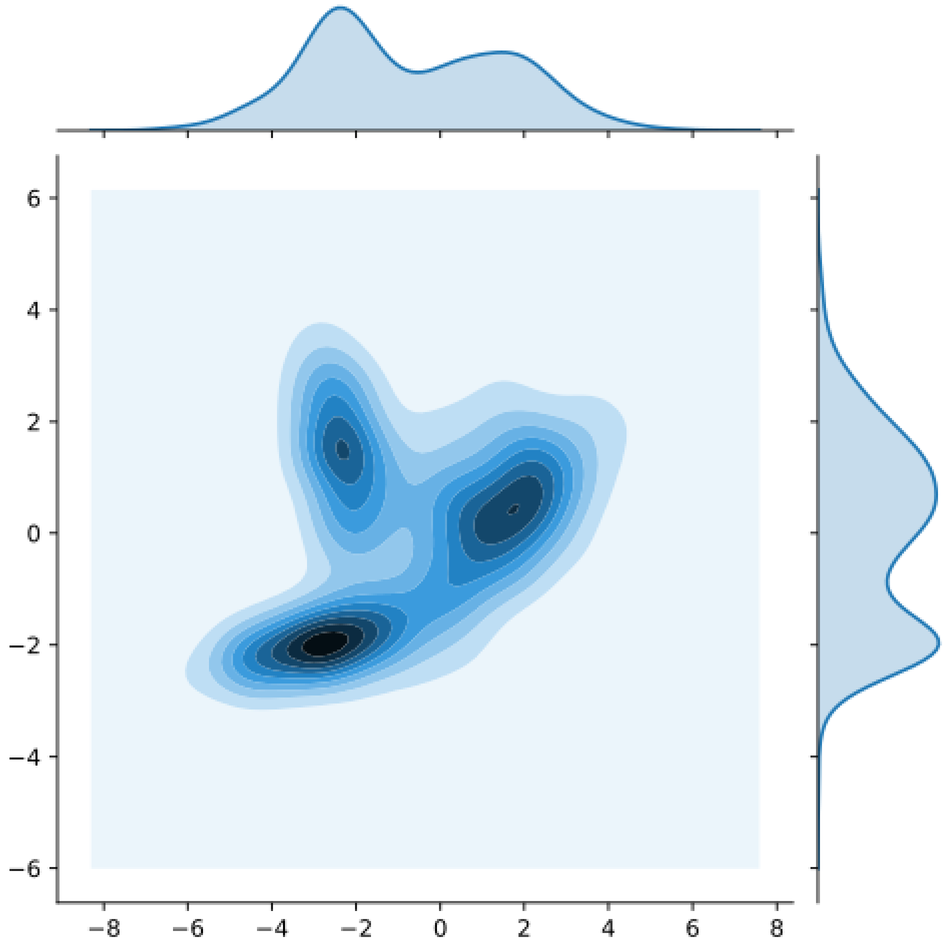
Credits: EE559 Deep Learning (Fleuret, 2018)

The Earth mover's distance is also known as the Wasserstein-1 distance and is defined as:

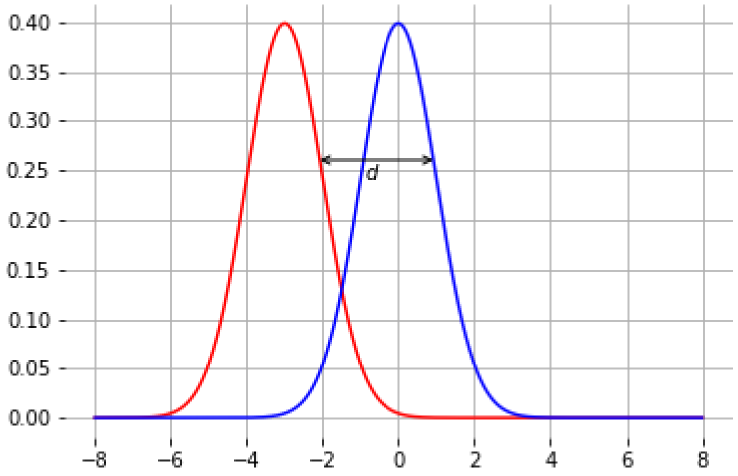
$$\mathrm{W}_1(p,q) = \inf_{\gamma \in \Pi(p,q)} \mathbb{E}_{(x,y)\sim\gamma}\left[||x-y||\right]$$

where:

- $\Pi(p,q)$ denotes the set of all joint distributions $\gamma(x,y)$ whose marginals are respectively p and q;
- $\gamma(x,y)$ indicates how much mass must be transported from x to y in order to transform the distribution p into q.
- $||\cdot||$ is the L1 norm and $||x-y||$ represents the cost of moving a unit of mass from x to y.


6
4
2
0
−2
−4
−6
−8
−6
−4
−2
0
2
4
6
8

Notice how the W_1 distance does not saturate. Instead, it increases monotonically with the distance between modes:



$$\mathrm{W}_1(p,q) = d$$

For any two distributions p and q,

- $W_1(p,q) \in \mathbb{R}^+$,
- $W_1(p,q) = 0$ if and only if $p = q$.

Wasserstein GAN

Given the attractive properties of the Wasserstein-1 distance, Arjovsky et al (2017) propose to learn a generative model by solving instead:

$$\theta^* = \arg\min_\theta \mathrm{W}_1(p_r(\mathbf{x})||p(\mathbf{x};\theta))$$

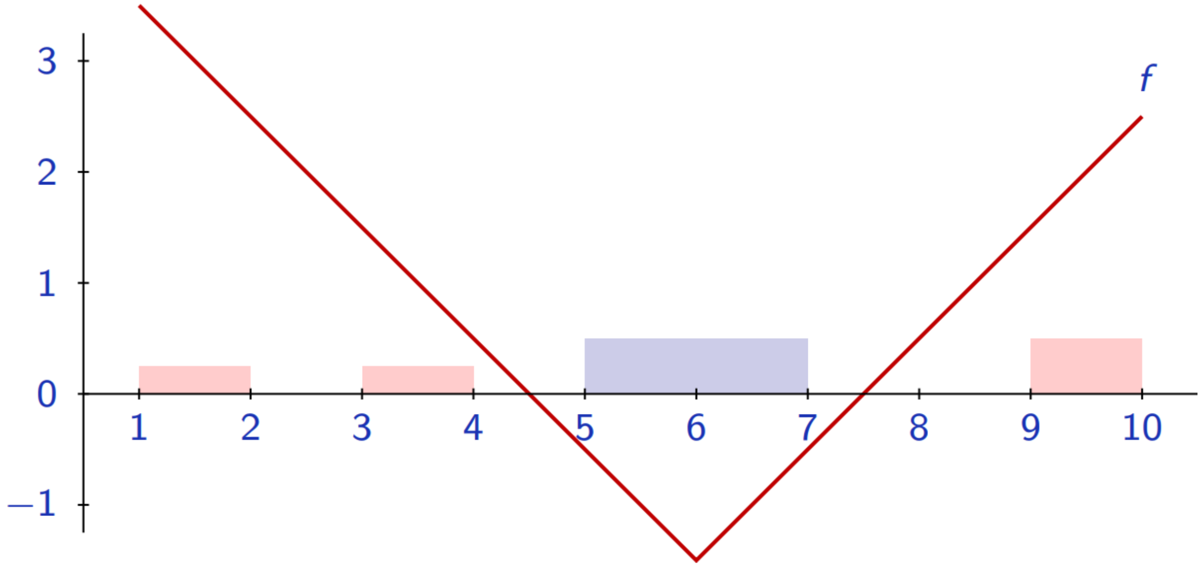
Unfortunately, the definition of W_1 does not provide with an operational way of estimating it because of the intractable $\inf$.

On the other hand, the Kantorovich-Rubinstein duality tells us that

$$\mathrm{W}_1(p_r(\mathbf{x})||p(\mathbf{x};\theta)) = \sup_{||f||_L \leq 1} \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})}\left[f(\mathbf{x})\right] - \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x};\theta)}\left[f(\mathbf{x})\right]$$

where the supremum is over all the 1-Lipschitz functions $f : \mathcal{X} \to \mathbb{R}$. That is, functions f such that

$$||f||_L = \max_{\mathbf{x},\mathbf{x}'} \frac{||f(\mathbf{x}) - f(\mathbf{x}')||}{||\mathbf{x} - \mathbf{x}'||} \leq 1.$$



For $p = \frac{1}{4}\mathbf{1}_{[1,2]} + \frac{1}{4}\mathbf{1}_{[3,4]} + \frac{1}{2}\mathbf{1}_{[9,10]}$ and $q = \mathbf{1}_{[5,7]}$,

$$\begin{aligned}
\mathrm{W}_1(p,q) &= 4 \times \frac{1}{4} + 2 \times \frac{1}{4} + 3 \times \frac{1}{2} = 3 \\
&= \underbrace{\left(3 \times \frac{1}{4} + 1 \times \frac{1}{4} + 2 \times \frac{1}{2}\right)}_{\mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})}[f(\mathbf{x})]} - \underbrace{\left(-1 \times \frac{1}{2} - 1 \times \frac{1}{2}\right)}_{\mathbb{E}_{\mathbf{x}\sim p(\mathbf{x};\theta)}[f(\mathbf{x})]} = 3
\end{aligned}$$

Credits: EE559 Deep Learning (Fleuret, 2018)

Using this result, the Wasserstein GAN algorithm consists in solving the minimax problem:

$$\theta^* = \arg\min_\theta \max_{\phi:||d(\cdot;\phi)||_L \leq 1} \mathbb{E}_{\mathbf{x}\sim p_r(\mathbf{x})}\left[d(\mathbf{x};\phi)\right] - \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x};\theta)}\left[d(\mathbf{x};\phi)\right]$$

Note that this formulation is very close to the original GAN, except that:

- The classifier $d : \mathcal{X} \to [0,1]$ is replaced by a critic function $d : \mathcal{X} \to \mathbb{R}$ and its output is not interpreted through the cross-entropy loss;
- There is a strong regularization on the form of d. In practice, to ensure 1-Lipschitzness,
  - Arjovsky et al (2017) propose to clip the weights of the critic at each iteration;
  - Gulrajani et al (2017) add a regularization term to the loss.
- As a result, Wasserstein GANs benefit from:
  - a meaningful loss metric,
  - improved stability (no mode collapse is observed).

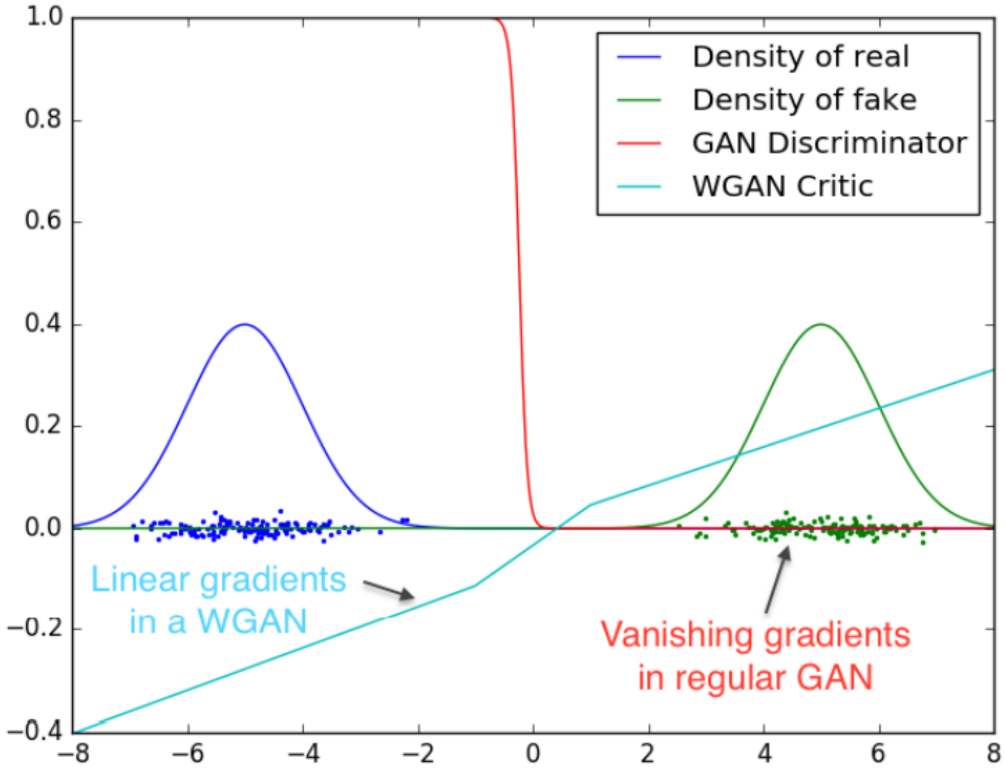
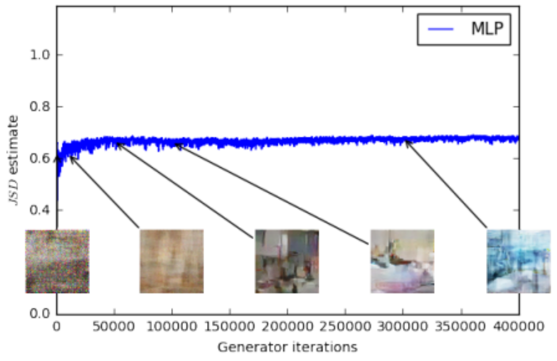
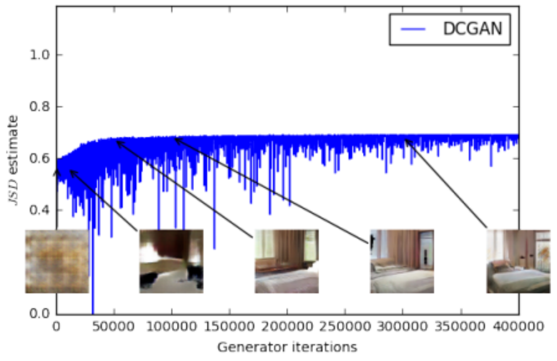


Figure 2: Optimal discriminator and critic when learning to differentiate two Gaussians. As we can see, the discriminator of a minimax GAN saturates and results in vanishing gradients. Our WGAN critic provides very clean gradients on all parts of the space.

(Arjovsky et al, 2017)





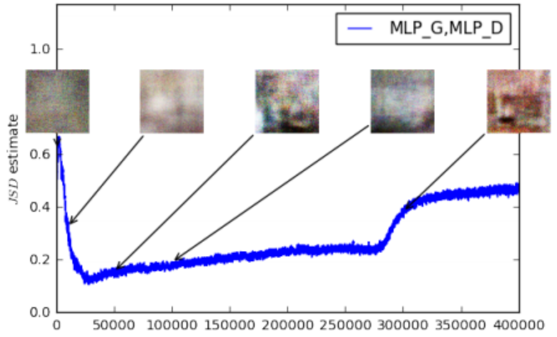
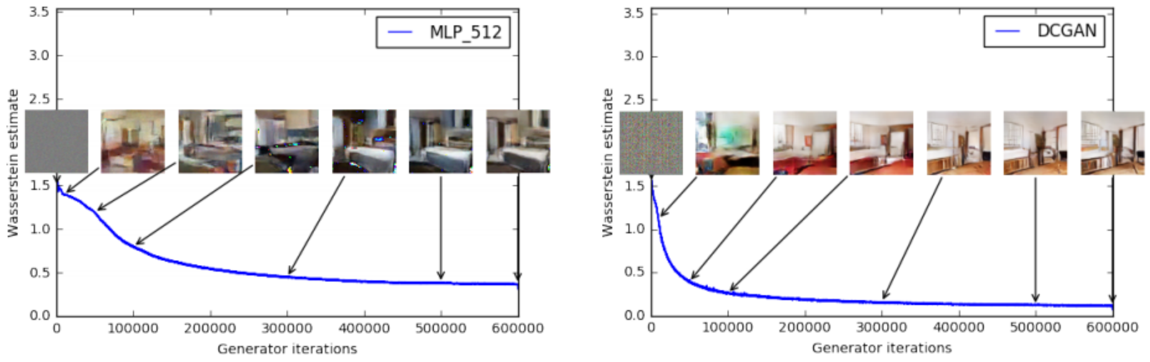


Figure 4: JS estimates for an MLP generator (upper left) and a DCGAN generator (upper right) trained with the standard GAN procedure. Both had a DCGAN discriminator. Both curves have increasing error. *Samples get better for the DCGAN but the JS estimate increases or stays constant, pointing towards 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.*

(Arjovsky et al, 2017)



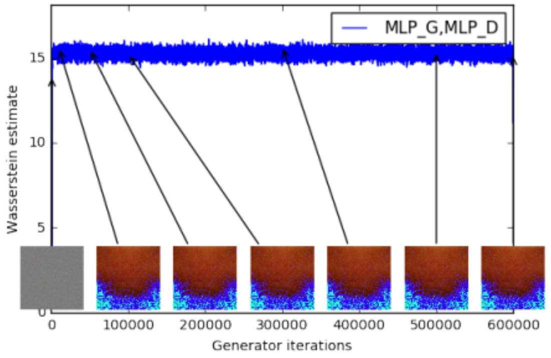
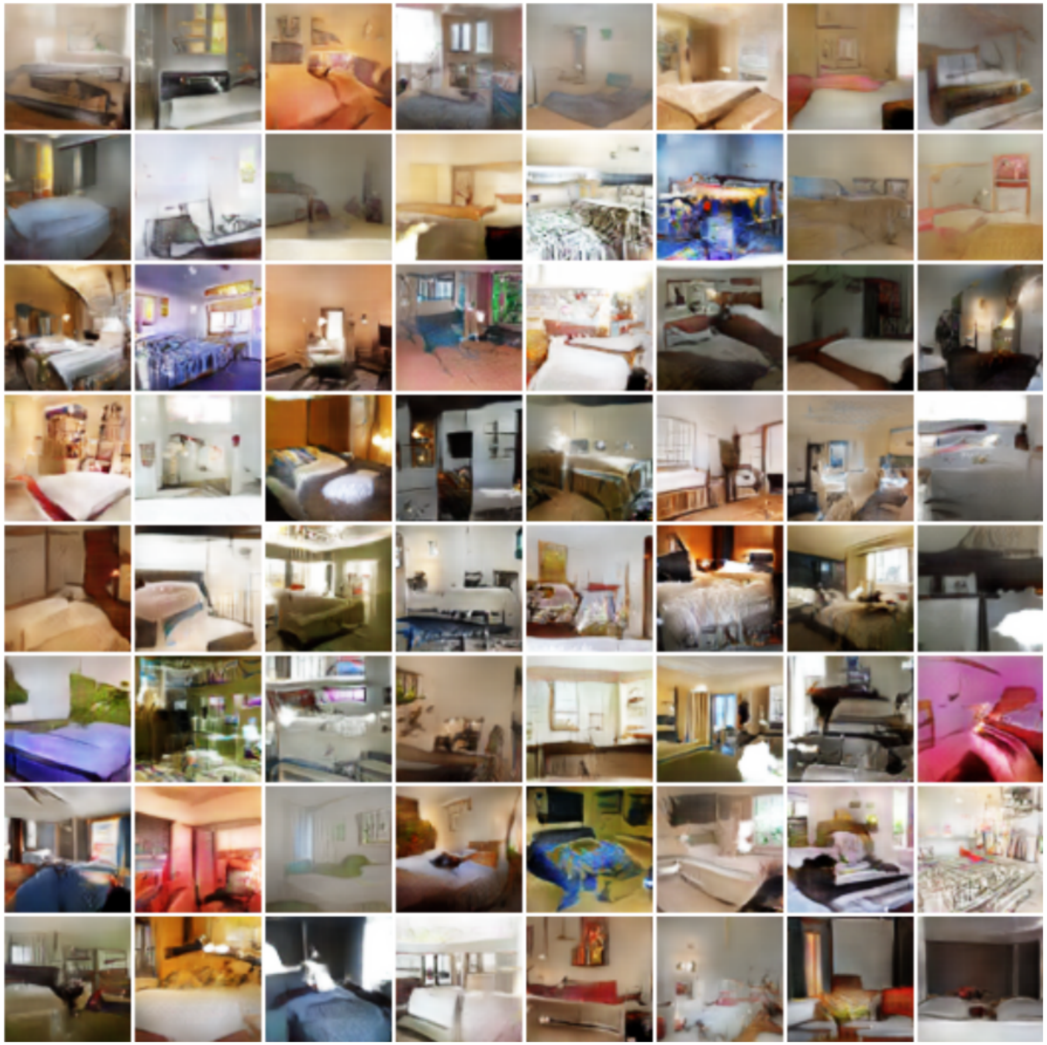


Figure 3: Training curves and samples at different stages of training. We can see a clear correlation between lower error and better sample quality. Upper left: the generator is an MLP with 4 hidden layers and 512 units at each layer. The loss decreases constistently as training progresses and sample 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.

(Arjovsky et al, 2017)

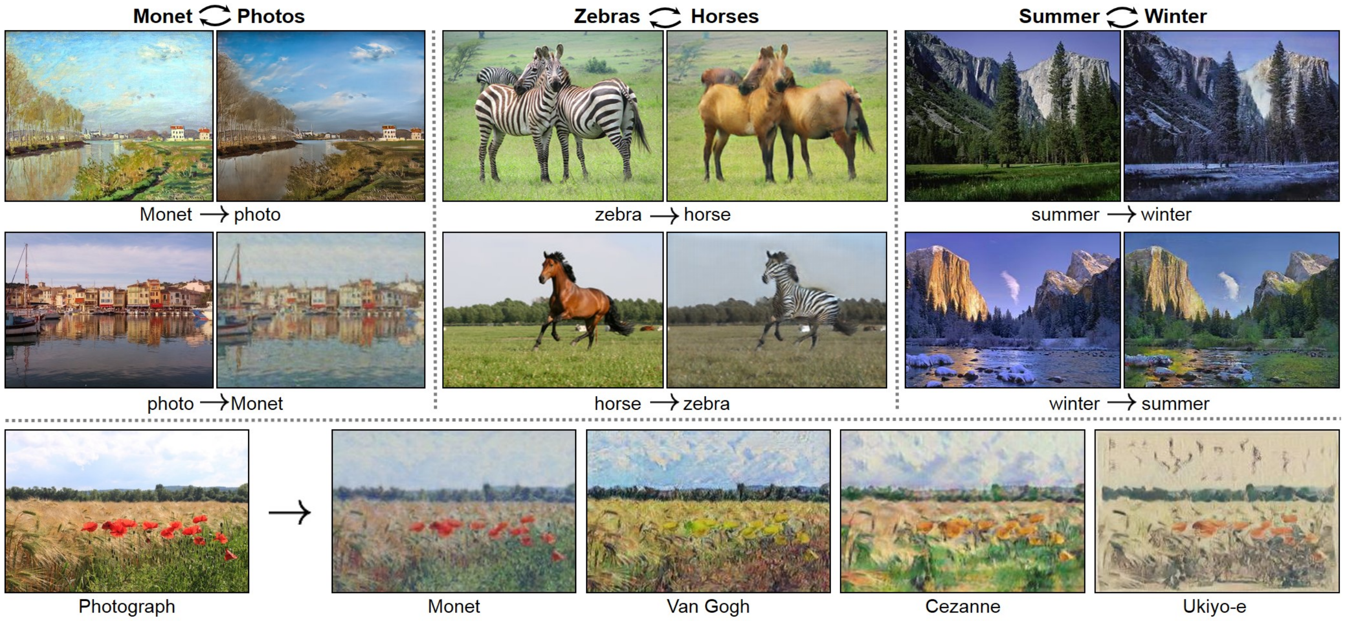


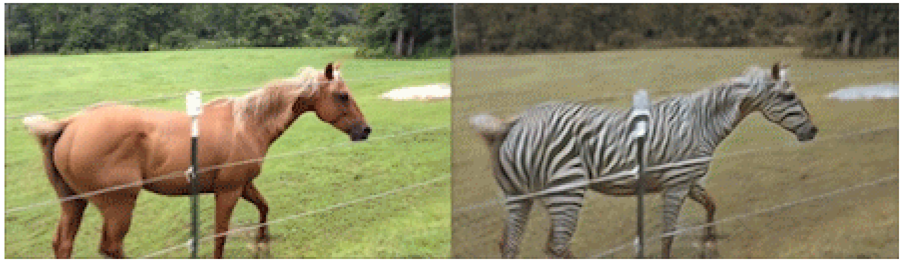
(Arjovsky et al, 2017)

Some applications

$p(\mathbf{z})$ need not be a random noise distribution.

Image-to-image translation

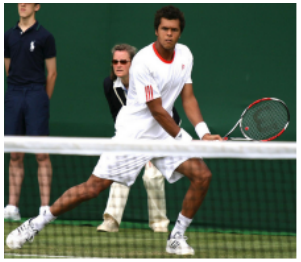


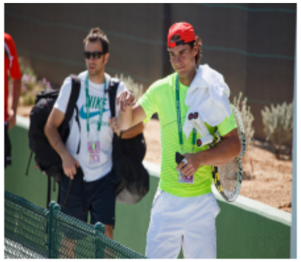


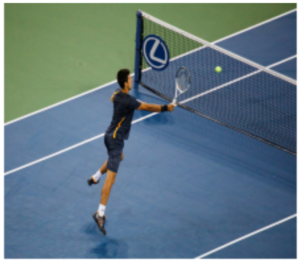
(Zhu et al, 2017)

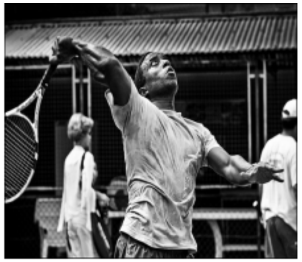
(Wang et al, 2017)

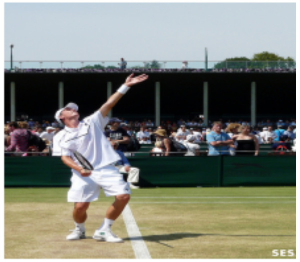
Captioning

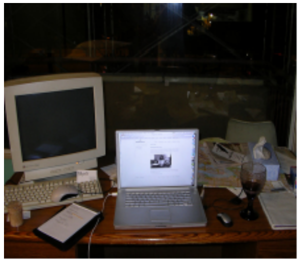

a tennis player gets ready to return a serve

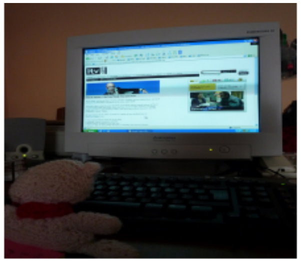

two men dressed in costumes and holding tennis rackets

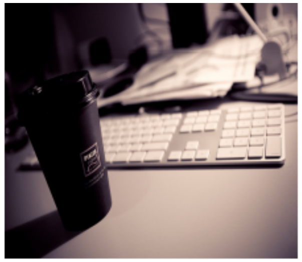

a tennis player hits the ball during a match

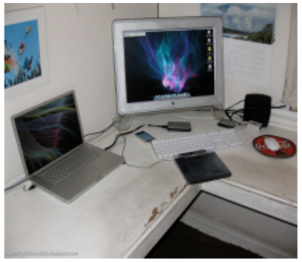

a male tennis player in action on the court

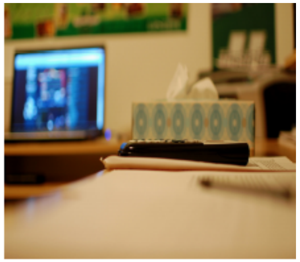

a man in white is about to serve a tennis ball


a laptop and a desktop computer sit on a desk


a person is working on a computer screen


a cup of coffee sitting next to a laptop


a laptop computer sitting on top of a desk next to a


a picture of a computer on a desk

(Shetty et al, 2017)

Text-to-image synthesis

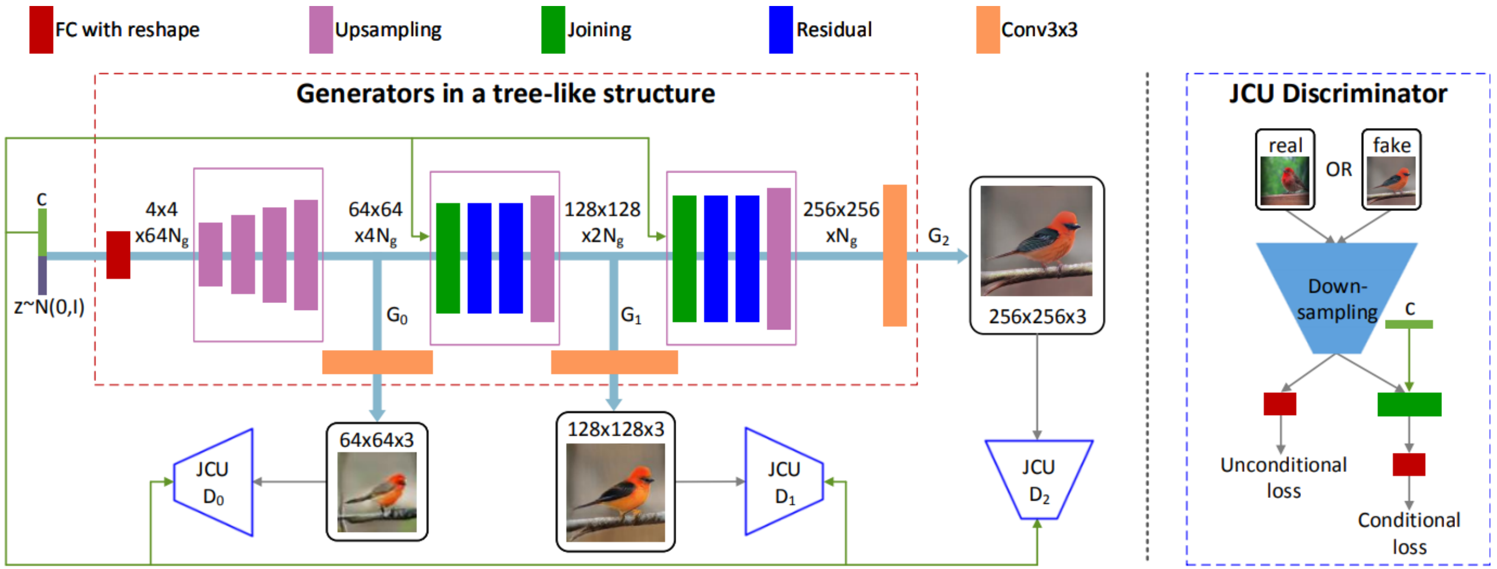
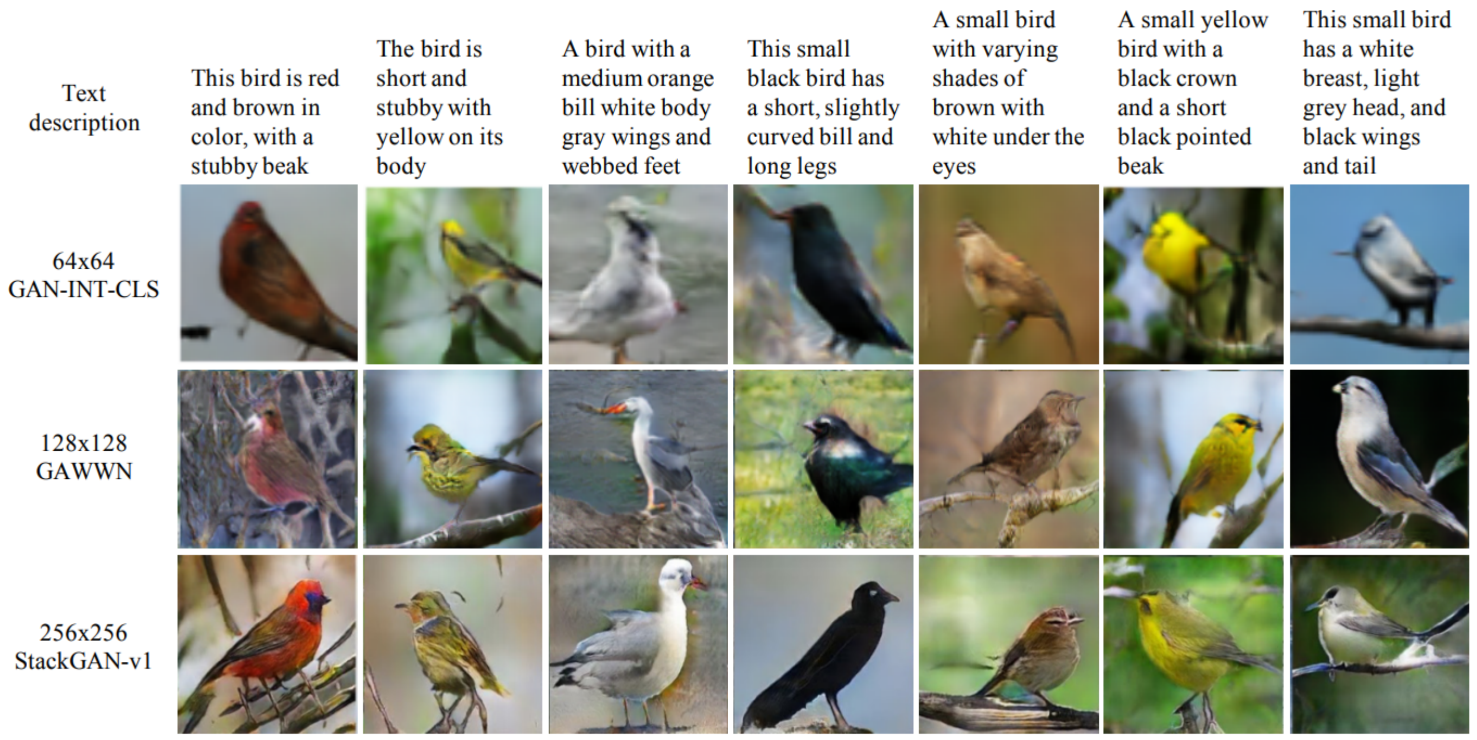


Fig. 2: The overall framework of our proposed StackGAN-v2 for the conditional image synthesis task. c is the vector of conditioning variables which can be computed from the class label, the text description, *etc.*. N_g and N_d are the numbers of channels of a tensor.

(Zhang et al, 2017)


Text description

This bird is red and brown in color, with a stubby beak

The bird is short and stubby with yellow on its body

A bird with a medium orange bill white body gray wings and webbed feet

This small black bird has a short, slightly curved bill and long legs

A small bird with varying shades of brown with white under the eyes

A small yellow bird with a black crown and a short black pointed beak

This small bird has a white breast, light grey head, and black wings and tail

Fig. 3: Example results by our StackGAN-v1, GAWWN [29], and GAN-INT-CLS [31] conditioned on text descriptions from CUB test set.

(Zhang et al, 2017)

Unsupervised machine translation

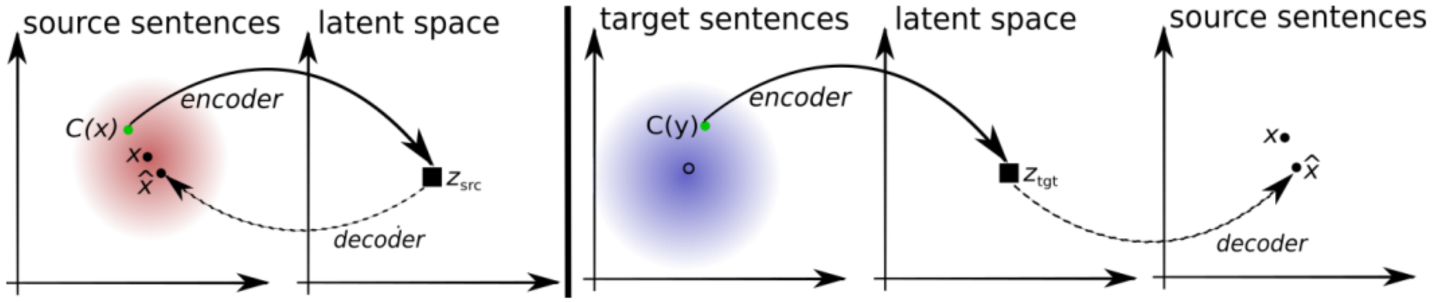


Figure 1: Toy illustration of the principles guiding the design of our objective function. Left (auto-encoding): the model is trained to reconstruct a sentence from a noisy version of it. x is the target, $C(x)$ is the noisy input, $\hat{x}$ is the reconstruction. Right (translation): the model is trained to translate a sentence in the other domain. The input is a noisy translation (in this case, from source-to-target) produced by the model itself, M, at the previous iteration (t), $y = M^{(t)}(x)$. The model is symmetric, and we repeat the same process in the other language. See text for more details.

(Lample et al, 2018)

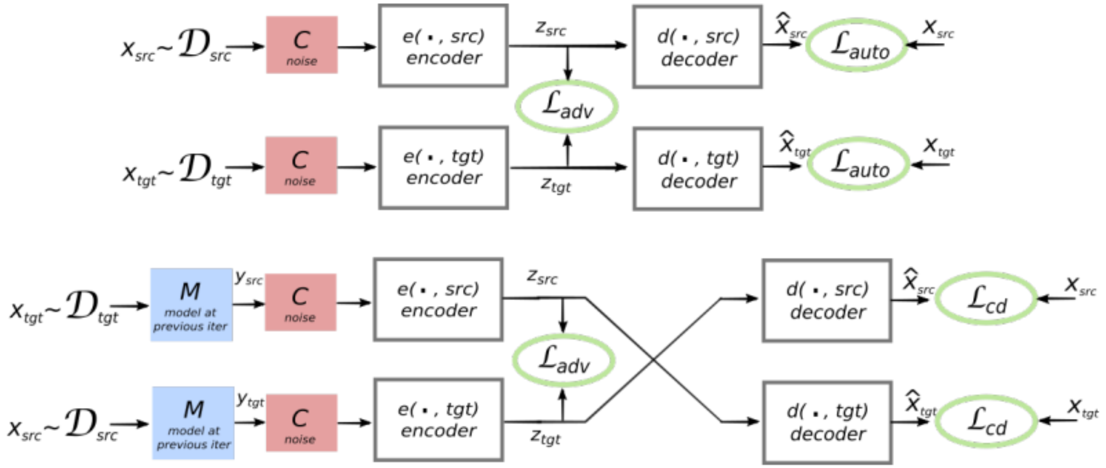
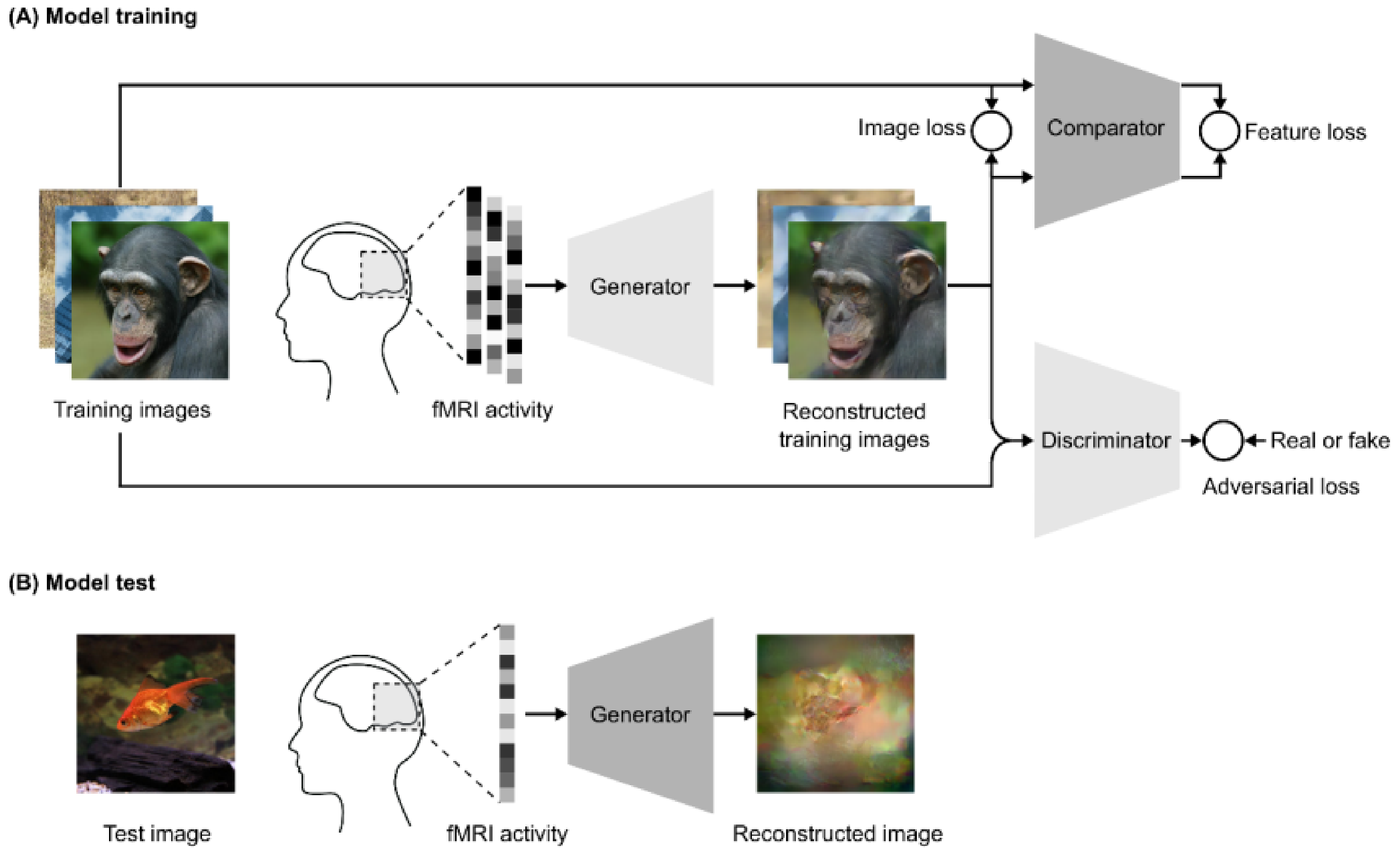


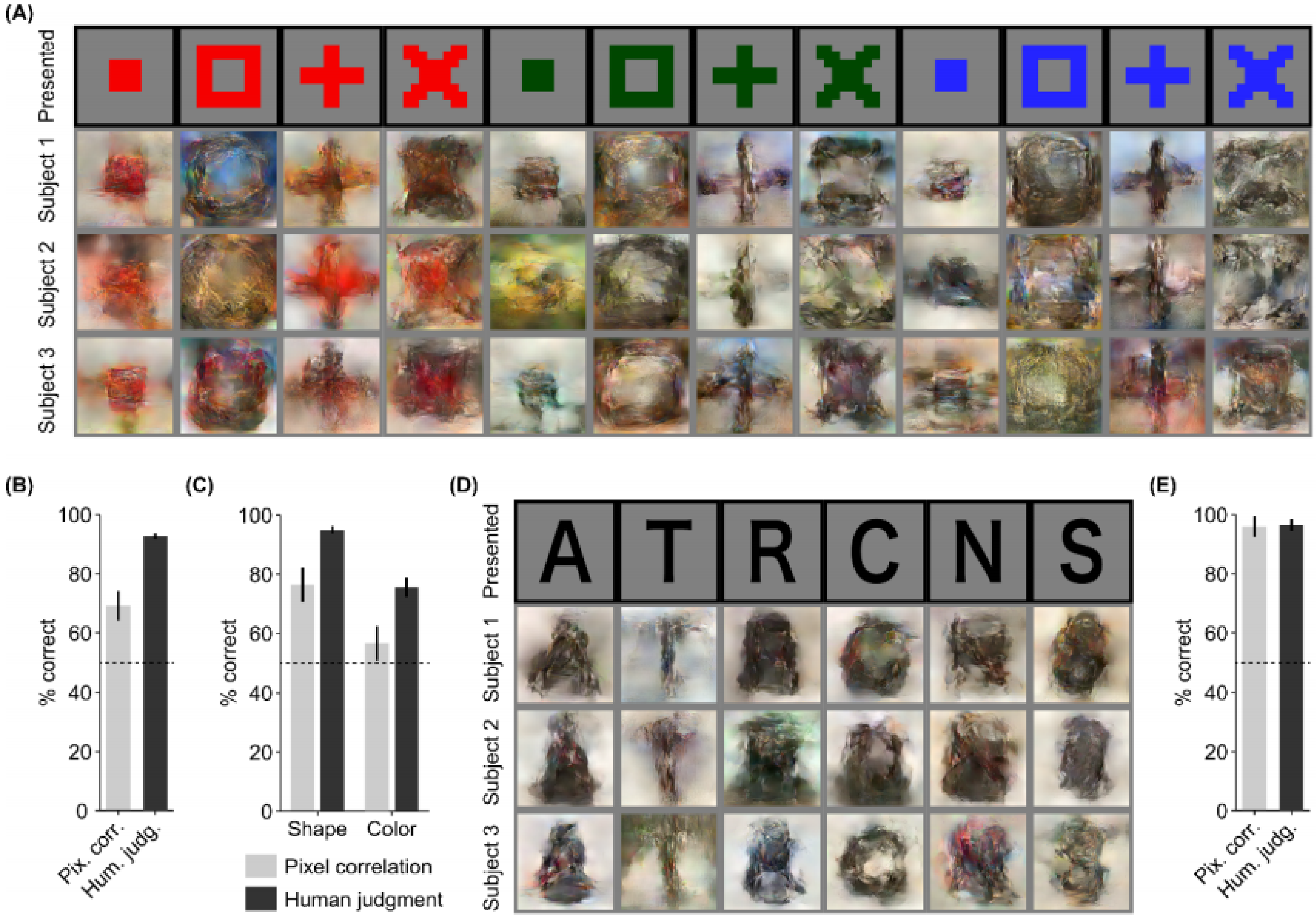
Figure 2: Illustration of the proposed architecture and training objectives. The architecture is a sequence to sequence model, with both encoder and decoder operating on two languages depending on an input language identifier that swaps lookup tables. Top (auto-encoding): the model learns to denoise sentences in each domain. Bottom (translation): like before, except that we encode from another language, using as input the translation produced by the model at the previous iteration (light blue box). The green ellipses indicate terms in the loss function.

(Lample et al, 2018)

Brain reading



(Shen et al, 2018)



(Shen et al, 2018)

(Shen et al, 2018)

The end.

References

- EE-559 Deep learning (Fleuret, 2018)
- Tutorial: Generative adversarial networks (Goodfellow, 2016)
- From GAN to WGAN (Weng, 2017)
- Wasserstein GAN and the Kantorovich-Rubinstein Duality (Herrmann, 2017)