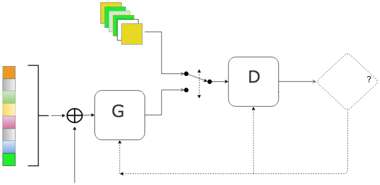


Generative Adversarial Networks

Vincent Barra
LIMOS, UMR 6158 CNRS, Université Clermont Auvergne

Latent models

A latent variable model relates a set of observable variables $\boldsymbol{x} \in X$ to a set of latent variables $\boldsymbol{h} \in H$

$$\mathbf{p}(\boldsymbol{x}, \boldsymbol{h}) = \mathbf{p}(\boldsymbol{x}|\boldsymbol{h})\mathbf{p}(\boldsymbol{h})$$

if $\boldsymbol{h}$ are causal factors for $\boldsymbol{x} \Rightarrow$ sampling from $\mathbf{p}(\boldsymbol{x}|\boldsymbol{h})$ = generative process from H to X.

Inference

Inference: given $\mathbf{p}(\boldsymbol{x}, \boldsymbol{h})$, compute

$$\mathbf{p}(\boldsymbol{h}|\boldsymbol{x}) = \frac{\mathbf{p}(\boldsymbol{x}|\boldsymbol{h})\mathbf{p}(\boldsymbol{h})}{\mathbf{p}(\boldsymbol{x})}$$

Intractable

GAN

GAN

- Generative Adversarial Network
- Two-player game between a discriminator D and a Generator G
- G and D: neural networks

LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

GAN

GAN

- Generative Adversarial Network
- Two-player game between a discriminator D and a Generator G
- G and D: neural networks

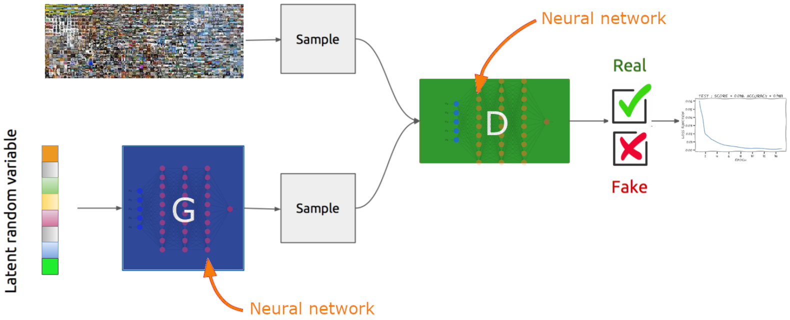
Game

- G tries to generate synthetic data close to real ones, and aims at fooling D
- D tries to discriminate between real and fake images.
- Adversarial: G and D have antagonistic objectives.

LIMOS Laboratoire d'informatique, de modélisation et d'optimisation des systèmes

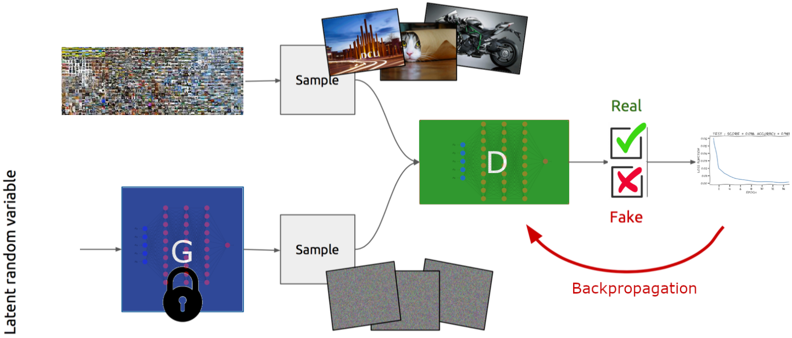
Université Clermont Auvergne

Overview

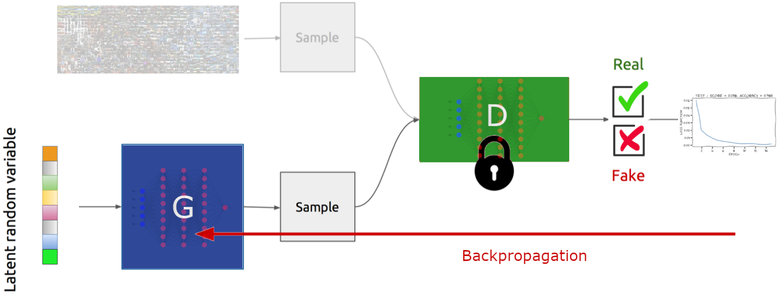


Sample
Neural network
Real
Latent random variable
G
D
Sample
Fake
Neural network

Overview


Sample
Sample
Latent random variable
G
D
Real
Fake
Backpropagation

Overview



LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

cnrs

UCA UNIVERSITÉ Clermont Auvergne

Generator and Discriminator

$G : \mathbb{R}^d \to \mathbb{R}^n$

- Function from the latent space to the data space
- MLP, CNN, RNN...
- trained so that for $\boldsymbol{h} \in \mathbb{R}^d,\ G(h) \sim \mathbf{p}_{data}$

LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

UNIVERSITÉ Clermont Auvergne

Generator and Discriminator

$G : \mathbb{R}^d \to \mathbb{R}^n$

- Function from the latent space to the data space
- MLP, CNN, RNN...
- trained so that for $\boldsymbol{h} \in \mathbb{R}^d,\ G(h) \sim \mathbf{p}_{data}$

$D : \mathbb{R}^n \to [0, 1]$

- Function from the data space producing a probability
- MLP, CNN, RNN...
- trained so that for $\boldsymbol{x} \in \mathbb{R}^n$, predicts if $\boldsymbol{x} = G(\boldsymbol{h})$ or real data

LIMOS Laboratoire d'Informatique, de Modélisation et d'Optimisation des Systèmes

Training

If G is fixed, training D is easy

1. pick up p real data $\boldsymbol{x}_1 \cdots \boldsymbol{x}_p \in \mathbb{R}^n$
2. generate p fakes $G(\boldsymbol{h}_i), i \in [\![1,p]\!]$, $h_i \sim \mathbf{p}_G$
3. build a training set = $Z = \{(G(\boldsymbol{h}_i), 0), (\boldsymbol{x}_i, 1), i \in [\![1,p]\!]\}$
4. train D by minimizing the binary cross-entropy

$$\begin{aligned}\mathcal{L}(Z) &= -\frac{1}{2p}\left(\sum_{i=1}^{p}[log D(\boldsymbol{x}_i) + log(1 - D(G(\boldsymbol{h}_i)))]\right)\\ &= -\frac{1}{2}\left(\mathbb{E}_{X\sim \mathbf{p}_{data}} log(D(X)) + \mathbb{E}_{X\sim \mathbf{p}_G} log(1 - D(X))\right)\end{aligned}$$

LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES | cnrs | UCA Université Clermont Auvergne

TRAINING

If G is fixed, training D is easy

1. pick up p real data $\boldsymbol{x}_1 \cdots \boldsymbol{x}_p \in \mathbb{R}^n$
2. generate p fakes $G(\boldsymbol{h}_i), i \in [\![1,p]\!]$, $h_i \sim \mathbf{p}_G$
3. build a training set = $Z = \{(G(\boldsymbol{h}_i), 0), (\boldsymbol{x}_i, 1), i \in [\![1,p]\!]\}$
4. train D by minimizing the binary cross-entropy

$$\begin{aligned}\mathcal{L}(Z) &= -\frac{1}{2p}\left(\sum_{i=1}^{p}[log D(\boldsymbol{x}_i) + log(1 - D(G(\boldsymbol{h}_i)))]\right) \\ &= -\frac{1}{2}\left(\mathbb{E}_{X\sim \mathbf{p}_{data}} log(D(X)) + \mathbb{E}_{X\sim \mathbf{p}_G} log(1 - D(X))\right)\end{aligned}$$

- $Min \; -log D(\boldsymbol{x}_i)$: maximizing the recognition of true data
- $Min \; -log(1 - D(G(\boldsymbol{h}_i)))$: maximizing the recognition of fake data

LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

UNIVERSITÉ Clermont Auvergne

TRAINING

But... G wants to fool D and has to be optimized to maximize D's loss.

Loss of G

$$\mathcal{L}_G(D,G) = \mathbb{E}_{X\sim\mathbf{p}_{data}} log(D(X)) + \mathbb{E}_{X\sim\mathbf{p}_G} log(1 - D(X))$$

- high if D is right
- low if G fools D often

Training

But... G wants to fool D and has to be optimized to maximize D's loss.

Loss of G

$$\mathcal{L}_G(D,G) = \mathbb{E}_{X\sim \mathbf{p}_{data}} log(D(X)) + \mathbb{E}_{X\sim \mathbf{p}_G} log(1-D(X))$$

- high if D is right
- low if G fools D often

Find an optimal generator G^* fooling any D

$$G^* \quad = \quad arg\min_G \max_D \mathcal{L}_G(D,G) = arg\min_G \mathcal{L}_G(D^*_G, G)$$

where $D^*_G = arg\max_D \mathcal{L}_G(D,G)$

$\Rightarrow$ Find a generator G whose loss against the best discriminator is low.

LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

Université Clermont Auvergne

Training

Nash equilibrium

$$(\forall \boldsymbol{x})\ D^*_G(\boldsymbol{x}) = \frac{\mathsf{p}_{data}(\boldsymbol{x})}{\mathsf{p}_{data}(\boldsymbol{x}) + \mathsf{p}_G(\boldsymbol{x})}$$

and thus

$$\begin{aligned}
\mathcal{L}_G(D^*_G, G) &= \mathbb{E}_{X\sim \mathsf{p}_{data}} log(D^*_G(X)) + \mathbb{E}_{X\sim \mathsf{p}_G} log(1 - D^*_G(X)) \\
&= \mathbb{E}_{X\sim \mathsf{p}_{data}} log\left(\frac{\mathsf{p}_{data}(\boldsymbol{x})}{\mathsf{p}_{data}(\boldsymbol{x}) + \mathsf{p}_G(\boldsymbol{x})}\right) + \mathbb{E}_{X\sim \mathsf{p}_G} log\left(\frac{\mathsf{p}_{latent}(\boldsymbol{x})}{\mathsf{p}_{data}(\boldsymbol{x}) + \mathsf{p}_G(\boldsymbol{x})}\right) \\
&= KL\left(\mathsf{p}_{data} || \frac{\mathsf{p}_{data} + \mathsf{p}_G}{2}\right) + KL\left(\mathsf{p}_G || \frac{\mathsf{p}_{data} + \mathsf{p}_G}{2}\right) - log(4) \\
&= 2JS(\mathsf{p}_{data}, \mathsf{p}_G) - log(4) \quad \text{(JS: Jensen-Shannon divergence)}
\end{aligned}$$

Training

In practice D is not fully optimized when optimizing G
Alternating gradient step for G and D

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:

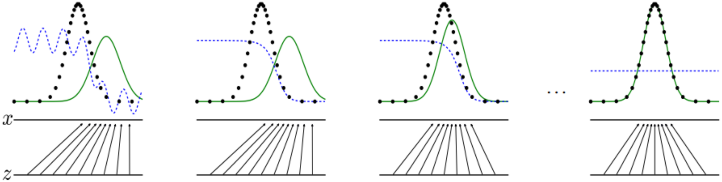
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

Source: Goodfellow et al., 2014

LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

Illustration



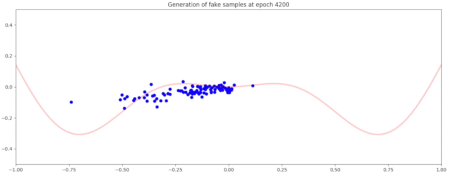
Source: Goodfellow et al., 2014

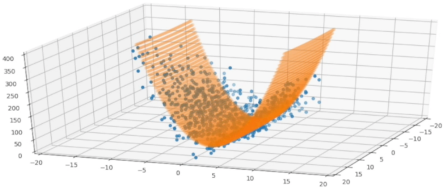
- black, dotted line : the data generating distribution
- green solid line : the generator distribution
- blue, dashed line : the discriminator
- z is sampled uniformly over the domain described by the lower lines

1. initial values of the data, G and D distributions.
2. Convergence $D \to D^*$
3. G updating: gradient of D guides $G(h)$ to high probability regions for the original data
4. After several epochs

LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES cnrs UCA UNIVERSITÉ Clermont Auvergne

Examples


Generation of fake samples at epoch 4200
0.4
0.2
0.0
−0.2
−0.4
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


400
350
300
250
200
150
100
50
0
−20
−15
−10
−5
0
5
10
15
20

Examples

Examples

Training problems

1. Oscillation between generator and discriminator loss

- competitive loss between G and D
- no guarantee that the loss will decrease

LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

UNIVERSITÉ Clermont Auvergne

Training problems

1. Oscillation between generator and discriminator loss
2. Mode collapse

- G tries to fool D.
- When G is trained without updating D, G produces mode x^* that fools D the most
- When D is training, the most effective way to detect generated images is to detect this single mode

$\Rightarrow$ generator produces examples of a particular kind only

- hard problem to solve
- one possible solution: Wasserstein loss (see next)

TRAINING PROBLEMS

1. Oscillation between generator and discriminator loss
2. Mode collapse
3. Discriminator is too strong, such that the gradient for the generator vanishes and the generator can't keep up

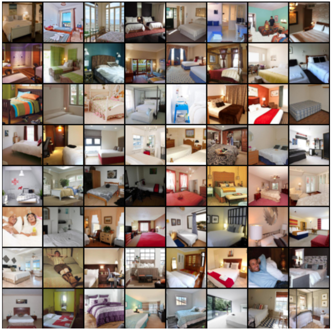
- fake samples can be initially so "fake" that the response of D saturates
- $log(1 - D(X))$ far in the exponential tail of the sigmoïd of $D \Rightarrow$ null gradient
- $\mathbb{E}_{X \sim \mathbf{p}_G} log(1 - D(X)) \longrightarrow -\mathbb{E}_{X \sim \mathbf{p}_G} log(D(X))$

A COOKING RECIPE FOR IMAGE GENERATION

"After extensive model exploration we identified a family of architectures that resulted in stable training across a range of datasets and allowed for training higher resolution and deeper generative models"
Radford et al., 2015

DC-GAN

- pooling layers in $D \rightarrow$ strided convolutions
- pooling layers in $G \rightarrow$ strided transposed convolutions in G
- use batchnorm in both D and G
- remove fully connected hidden layers
- use ReLU in G except for the output ($tanh$)
- use LeakyReLU activation in D for all layers.



Real bedrooms

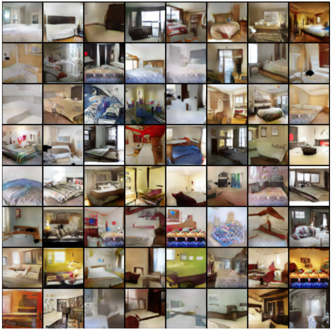
LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

A cooking recipe for image generation

"After extensive model exploration we identified a family of architectures that resulted in stable training across a range of datasets and allowed for training higher resolution and deeper generative models"
Radford et al., 2015

DC-GAN

- pooling layers in $D \rightarrow$ strided convolutions
- pooling layers in $G \rightarrow$ strided transposed convolutions in G
- use batchnorm in both D and G
- remove fully connected hidden layers
- use ReLU in G except for the output ($tanh$)
- use LeakyReLU activation in D for all layers.



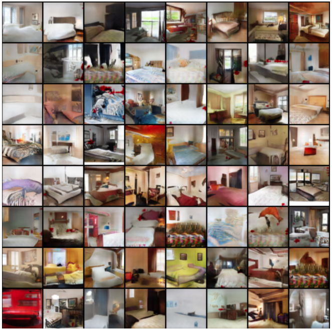
epoch 1

A cooking recipe for image generation

> "After extensive model exploration we identified a family of architectures that resulted in stable training across a range of datasets and allowed for training higher resolution and deeper generative models"
> Radford et al., 2015

DC-GAN

- pooling layers in D $\rightarrow$ strided convolutions
- pooling layers in G $\rightarrow$ strided transposed convolutions in G
- use batchnorm in both D and G
- remove fully connected hidden layers
- use ReLU in G except for the output ($tanh$)
- use LeakyReLU activation in D for all layers.



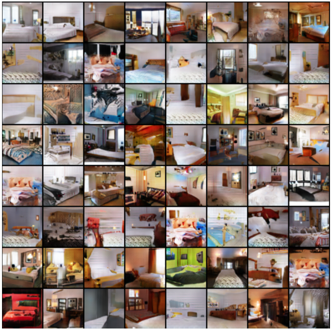
epoch 5

A cooking recipe for image generation

> "After extensive model exploration we identified a family of architectures that resulted in stable training across a range of datasets and allowed for training higher resolution and deeper generative models"
> Radford et al., 2015

DC-GAN

- pooling layers in D $\rightarrow$ strided convolutions
- pooling layers in G $\rightarrow$ strided transposed convolutions in G
- use batchnorm in both D and G
- remove fully connected hidden layers
- use ReLU in G except for the output ($tanh$)
- use LeakyReLU activation in D for all layers.



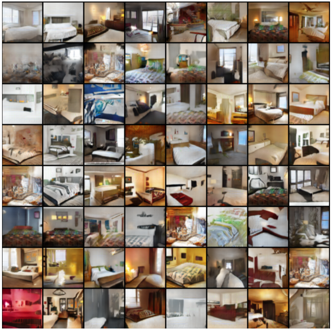
epoch 10

A cooking recipe for image generation

> "After extensive model exploration we identified a family of architectures that resulted in stable training across a range of datasets and allowed for training higher resolution and deeper generative models"
> Radford et al., 2015

DC-GAN

- pooling layers in $D \rightarrow$ strided convolutions
- pooling layers in $G \rightarrow$ strided transposed convolutions in G
- use batchnorm in both D and G
- remove fully connected hidden layers
- use ReLU in G except for the output ($tanh$)
- use LeakyReLU activation in D for all layers.



epoch 20

LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

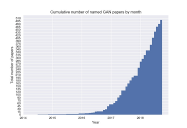
cnrs

UCA UNIVERSITÉ Clermont Auvergne

Since 2014, A LOT of models have been developed.

The GAN Zoo

Every week, new GAN papers are coming out and it's hard to keep track of them all, not to mention the incredibly creative ways in which researchers are naming these GANs! So, here's a list of what started as a fun activity compiling all named GANs!



You can also check out the same data in a tabular format with functionality to filter by year or do a quick search by title here.

Contributions are welcome. Add links through pull requests in gans.tsv file in the same format or create an issue to lemme know something I missed or to start a discussion.

Check out Deep Hunt - my weekly AI newsletter for this repo as blogpost and follow me on Twitter.

- 3D-ED-GAN - Shape Inpainting using 3D Generative Adversarial Network and Recurrent Convolutional Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling (github)
- 3D-IWGAN - Improved Adversarial Systems for 3D Object Generation and Reconstruction (github)
- 3D-PhysNet - 3D-PhysNet: Learning the Intuitive Physics of Non-Rigid Object Deformations

https://github.com/hindupuravinash/the-gan-zoo

LIMOS Laboratoire d'informatique, de modélisation et d'optimisation des systèmes

cnrs

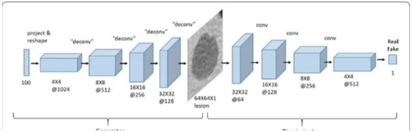
UCA Université Clermont Auvergne

Some models

Some models

A (non exhaustive) selection:

- DCGAN: from MLP to CNN (Radford).



Some models

Some models

A (non exhaustive) selection:

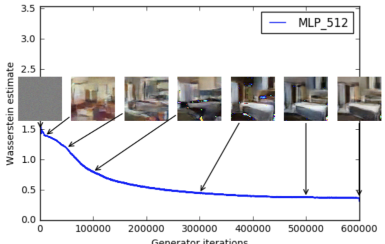
- WGAN: Wasserstein GAN .
  - JS divergence → Earth-Mover's distance:

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

$\Pi(\mathbf{p}_{data}, \mathbf{p}_G)$: set of all joint distributions $\gamma(x, y)$ whose marginals are $\mathbf{p}_{data}$ and $\mathbf{p}_G$

$\Rightarrow$ $\gamma(x, y)$: how much "mass" must be transported from x to y in order to transform $\mathbf{p}_{data}$ to $\mathbf{p}_G$.

  - More stable training
  - less mode collapsing



LIMOS LABORATOIRE D'INFORMA DE MODÉLISATION ET D'OI

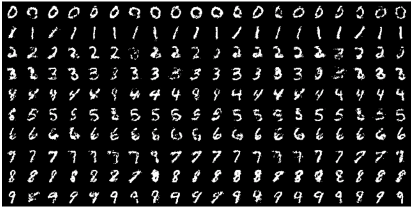
Some models

Some models

A (non exhaustive) selection:

- CGAN: Conditional GAN

$$\mathcal{L}_G(D,G) = \mathbb{E}_{x\sim\mathbf{p}_{data}} log(D(x|y)) + \mathbb{E}_{h\sim\mathbf{p}_G} log(1 - D(G(h|y)), y)$$



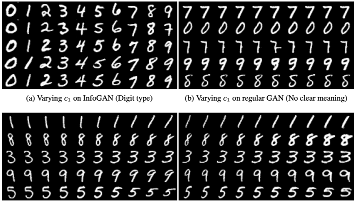
SOME MODELS

Some models

A (non exhaustive) selection:

- InfoGAN: CGAN trained in an unsupervised way = GAN + maximization of the mutual information between a small subset of the latent variables and the observation

$$\min_g \max_D \mathcal{L}_G(D, G) - \lambda I(y|G(h, y), \quad I(X, Y) = H(X) - H(X|Y)$$



(a) Varying c_1 on InfoGAN (Digit type)
(b) Varying c_1 on regular GAN (No clear meaning)
(c) Varying c_2 from -2 to 2 on InfoGAN (Rotation)
(d) Varying c_3 from -2 to 2 on InfoGAN (Width)

$y = (c_1, c_2, c_3)$, c_1 : categorical, $c_2, c_3 \sim Uniform(-1, 1)$

source: Chen et al, 2017

Some models

Some models

A (non exhaustive) selection:

- BiGAN: G maps from H to X and from X to H $\Rightarrow$ Adversarial Feature Learning

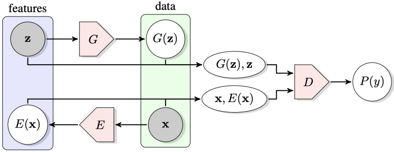


Figure 1: The structure of Bidirectional Generative Adversarial Networks (BiGAN).

Source: Donahue et al., 2017

LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

cnrs

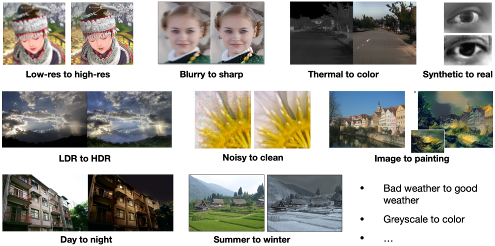
UCA UNIVERSITÉ Clermont Auvergne

SOME MODELS

Some models

A (non exhaustive) selection:

- Paired or unpaired image to image translation: h is replaced by an image (SRGAN, Pix2Pix, SimGAN,CycleGAN, DiscoGAN, CoVAE-GAN...)



Source: CVPR 2017 Tutorial

```
#Define generator
def G():
    g = Sequential()
    ...

    return g

#Define discriminator
def D():
    d = Sequential()
    ...

    return d


#Define loss function

g.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])
d.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])


#Define GAN
inputs = Input(shape=(z_dim, ))
h = G(inputs)
output = D(h)
gan = Model(inputs, output)
gan.compile(loss='binary_crossentropy', optimizer=adam, metrics=['accuracy'])
```

LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

cnrs

UCA UNIVERSITÉ Clermont Auvergne

```python
for e in range(1, epochs+1):

    for _ in range(batchCount):
        # batch creation
        image_batch = x_train[np.random.randint(0, x_train.shape[0], size=BATCH_SIZE)]
        # Latent variables
        noise = np.random.normal(0, 1, size=(BATCH_SIZE, z_dim))

        # Fake images
        generated_images = g.predict(noise)
        X = np.concatenate((image_batch, generated_images))
        # Labels
        y = np.zeros(2*BATCH_SIZE)
        y[:BATCH_SIZE] = 1


        # Train discriminator
        d.trainable = True
        d_loss = d.train_on_batch(X, y)

        # Train generator
        noise = np.random.normal(0, 1, size=(BATCH_SIZE, z_dim))
        y2 = np.ones(BATCH_SIZE)
        d.trainable = False
        g_loss = gan.train_on_batch(noise, y2)
```

```
# training
train_dataset = tf.data.Dataset.from_tensor_slices(x_train).shuffle(BUFFER_SIZE).batch(BATCH_SIZE)
for epoch in range(epochs):

    # for batchs
    for images in train_dataset:
        # latent variables
        noise = tf.random.normal([BATCH_SIZE, z_dim])

        # Gradient recording and updating
        with tf.GradientTape() as G_tape, tf.GradientTape() as D_tape:
            # Fake
            g_images = g(noise, training=True)

            # Output on real images
            real_output = d(images, training=True)

            # Output on fake images
            fake_output = d(g_images, training=True)

            #losses
            gen_loss = G_loss(fake_output)
            disc_loss = D_loss(real_output, fake_output)

        # Gradient computation
        G_gradients = G_tape.gradient(G_loss, g.trainable_variables)
        D_gradients = D_tape.gradient(D_loss, g.trainable_variables)

        # Updating
        G_optimizer.apply_gradients(zip(G_gradients, g.trainable_variables))
        D_optimizer.apply_gradients(zip(D_gradients, d.trainable_variables))
```

LIMOS LABORATOIRE D'INFORMATIQUE, DE MODÉLISATION ET D'OPTIMISATION DES SYSTÈMES

UNIVERSITÉ Clermont Auvergne