

DELIRES AUTOENCODERS AND GENERATIVE MODELS

Alasdair Newson

MVA - Delires

Introduction

- 1 Introduction
- 2 Autoencoders
 - Vanilla Autoencoder
 - Autoencoder variants
- 3 Generative models
 - Variational autoencoders
 - Generative Adversarial Networks
 - Texture and style models
- 4 Summary

Introduction

- Neural networks are often used for :
 - Classification/detection (MLPs, CNNs)
 - Modelling time-series, sequences (RNNs)
- All of these networks rely on the extraction of features to analyse data
- Idea : the network's internal representation of the data can be useful !
- **Autoencoders** and more generally **generative models** use this idea

Introduction

- What do you think of these faces?



Introduction

- What do you think of these faces?

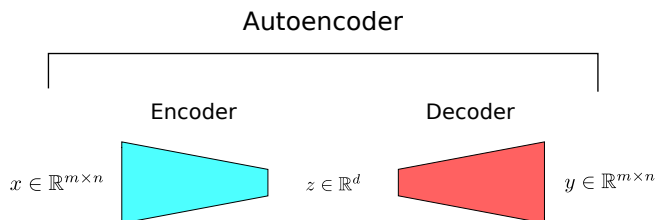


- These are all **generated by a generative model !!**
- In the next two lessons, we are going to see how this is possible

AUTOENCODERS

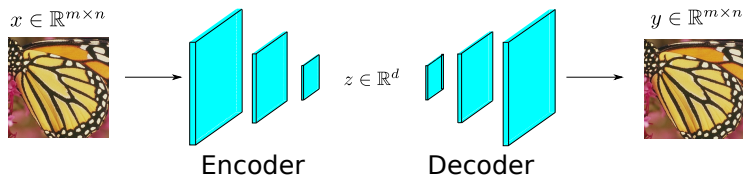
Autoencoders

- Autoencoders consist of two networks : an **encoder** and a **decoder**
 - Encoder : map data x to a smaller **latent space**
 - Decoder : map point z back from latent space to original data space
- Main idea : the latent space is a space where it is **easier to manipulate/understand data**
- More powerful and compact representation of data



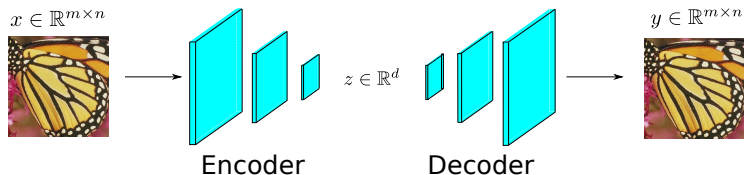
Autoencoders

- The autoencoder is trained to minimise some norm between the input x and the output y of the decoder
- In almost all cases, we have $d \ll mn$
- This forces the autoencoder to learn a compact and powerful latent space



Autoencoders

- Uses of autoencoders :
 - Data compression, dimensionality reduction
 - Classification (easier in latent space)
 - **Data generation/synthesis**



Autoencoders - some notation

- An AE is a neural network consisting of two sub-networks
 - The encoder Φ_e ,

$$\begin{aligned}\Phi_e: \mathbb{R}^{mn} &\rightarrow \mathbb{R}^d \\ x &\mapsto \Phi_e(x) = z\end{aligned}$$

- The decoder Φ_d ,

$$\begin{aligned}\Phi_d: \mathbb{R}^d &\rightarrow \mathbb{R}^{mn} \\ z &\mapsto \Phi_d(z) = y\end{aligned}$$

- As in other neural networks, the main components of AEs are **mlp's/convolutions**, **biases** and **non-linearities**

Autoencoders

- The autoencoder is trained to reproduce the input x as an output y , in the sense of some norm, having gone through the bottleneck of the network
- The norm most often used is the sum of squared differences (ℓ_2 -norm)

Autoencoding training minimisation problem

$$\begin{aligned}\mathcal{L}(x) &= \|y - x\|_2^2 \\ &= \sum_i^m \sum_j^n \left((\Phi_d \circ \Phi_e(x))_{i,j} - x_{i,j} \right)^2\end{aligned}$$

- Put simply : **output should look like input !**

Autoencoders - upsampling

- Most often, the autoencoder uses convolutions
- A key question is how to create the bottleneck : **downsampling**
- We know how to downsample :
 - Strided convolutions
 - Max pooling
- How about **upsampling** ?

Autoencoders - upsampling

- Upsampling can be carried out in several ways :
 - Simple interpolation (linear, bilinear, bicubic)
 - Transposed convolution
- **Transposed convolution** is probably the most common upsampling
 - Simultaneously upsamples and “convolves”
- Let's take a look at how this works

Autoencoders - transposed convolution

- Recall that we can write the convolution as a matrix/vector multiplication (see lecture on CNNs)
- For example, take the convolution with the Laplacian operator

$$w = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{pmatrix} \rightarrow A_w = \begin{pmatrix} 4 & -1 & \dots & -1 & \dots & & \\ -1 & 4 & -1 & \dots & -1 & \dots & \\ 0 & -1 & 4 & -1 & \dots & -1 & \dots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\ & & \dots & -1 & \dots & -1 & 4 \end{pmatrix}$$

- To carry out convolution + stride with subsampling s , we just remove certain rows from the matrix A_w (the elements not retained during subsampling)

Autoencoders - transposed convolution

- Let's take a concrete example :
 - We wish to upsample a 1D signal x of size 2 to size 4
 - Convolutional filter $w = [1, 2, 1]^T$
- First, we look at the convolution matrix at the higher resolution signal size 4

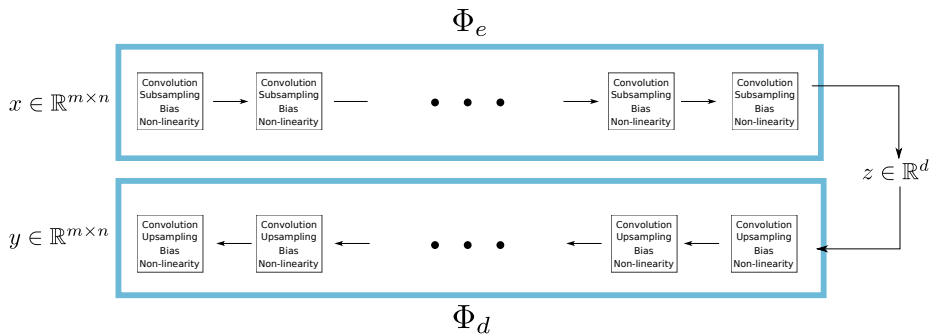
$$A_w = \begin{pmatrix} \mathbf{2} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{1} & \mathbf{2} & \mathbf{1} & \mathbf{1} \\ \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} & \mathbf{2} \end{pmatrix}$$

- Therefore, we have :

$$A_{w,s} = \begin{pmatrix} \mathbf{2} & \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{1} & \mathbf{2} & \mathbf{0} \end{pmatrix} \quad A_{w,s}^T = \begin{pmatrix} \mathbf{20} \\ \mathbf{11} \\ \mathbf{02} \\ \mathbf{00} \end{pmatrix}$$

Autoencoders

A generic autoencoder architecture



• $d \ll m * n$

Autoencoder variants

- Autoencoders come in many flavours, differ mainly by their loss functions
- Naive autoencoder loss \mathcal{L} can lead to certain problems
 - Overfitting to data, poor robustness
 - Latent space difficult to interpret, not necessarily meaningful w.r.t data space
- As is often the case in deep learning, this can be addressed using **regularisation**
- In practice, this means adding extra terms to \mathcal{L}

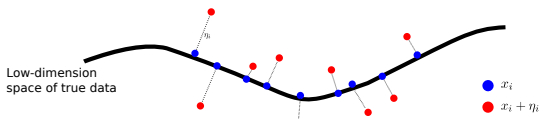
Denoising autoencoder

- First example : the denoising autoencoder
- We would like to make the encoder/decoder robust to **small perturbations** in the input data
- One solution : the **denoising autoencoder**

Denoising autoencoder

- Idea : add noise η to the input

$$\mathcal{L}(x) = \|\Phi_d \circ \Phi_e(x + \eta) - x\|_2^2$$



Sparse autoencoder

- Ideally, we want the code to be as **sparse** as possible
- Why? We want the smallest vector which accurately describes the data
 - Combinations of elements more difficult to interpret
 - Useful for classification

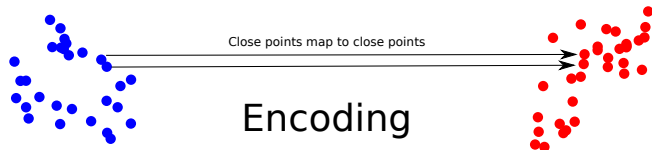
Sparse autoencoder

$$\mathcal{L}(x) = \|\Phi_d \circ \Phi_e(x) - x\|_2^2 + \lambda \|z\|_1$$

- The $\|z\|_1$ norm encourages sparsity in z

Contractive autoencoder

- Another approach to regularisation : contractive autoencoder
- Close points in data space map to close points in latent space (thus, contractive)



- How can we impose this?
- $\frac{\partial z_j}{\partial x_i}$ should be small, for all couples (i, j)

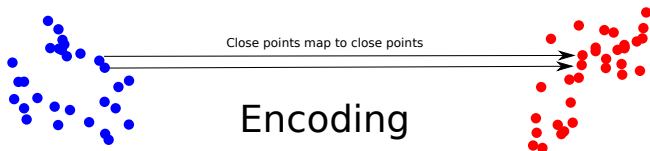
Contractive autoencoder

Contractive autoencoder

- Add the Frobenius (ℓ_2) norm of the Jacobian of z w.r.t. x , $J_x z$, to the cost function

$$\begin{aligned}\mathcal{L}(x) &= \|\Phi_d \circ \Phi_e(x) - x\|_2^2 + \lambda \|J_x E(x)\|_F^2 \\ &= \|\Phi_d \circ \Phi_e(x) - x\|_2^2 + \lambda \|J_x z\|_F^2\end{aligned}$$

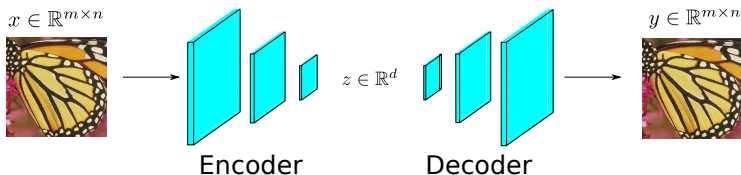
- Reminder, Jacobian : $J_x(z) = \begin{pmatrix} \frac{\partial z_1}{\partial x_1} & \cdots & \frac{\partial z_1}{\partial x_{mn}} \\ \vdots & \ddots & \vdots \\ \frac{\partial z_d}{\partial x_1} & \cdots & \frac{\partial z_d}{\partial x_{mn}} \end{pmatrix}$



Autoencoders - summary

Important points to remember

- Autoencoder consists of two networks : encoder and decoder
- These compress to and from a **smaller dimension latent space**
- This latent space represents data more **powerfully and compactly**



Autoencoders

- Example of autoencoder use : interpolation of complex data



Interpolation of complex data[†]

[†] Generative Visual Manipulation on the Natural Image Manifold, J-Y. Zhu, P. Krähenbühl, E. Shechtman, A. Efros, CVPR 2016

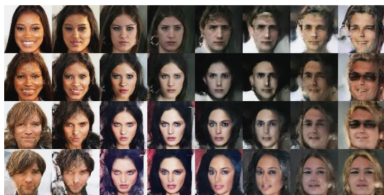
GENERATIVE MODELS

Generative models

- In many applications, it is desirable to **synthesise** data
 - Video post-production
 - Data augmentation
- Several types of generative models exist :
 - Restricted Boltzmann machines, Deep Belief models
 - **Variational autoencoders**
 - **Generative Adversarial Networks**
 - **Texture synthesis and style transfer models**
- The common idea in these models is the internal representation/latent space of the network

Generative models

- Modern generative models produce highly realistic, (relatively) high-definition images



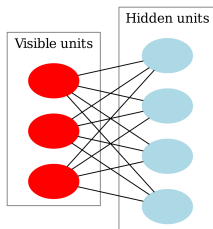
Synthesis examples from “Real NVP”[†]

- Before, we take a small detour to present Restricted Boltzmann Machines

[†] Density estimation using Real NVP, L. Dinh, J. Sohl-Dickstein, S. Bengio, arXiv :1605.08803 2016

Restricted Boltzmann machines

- A **Restricted Boltzmann machine** is a collection of binary random variables
- The variables' probability distribution is learned during training
- The variables form a bipartite graph



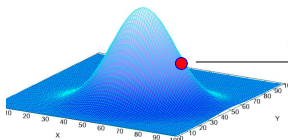
- Restricted Boltzmann machines are used less nowadays in favour of **variational autoencoders** and **Generative Adversarial Networks**, which we will see now

Variational autoencoder

- Suppose we want to produce **random examples of data**, how would we go about this?

Variational autoencoder

- Suppose we want to produce **random examples of data**, how would we go about this?
- We can model the latent space in a **probabilistic manner**
- Synthesis will then consist of :
 - 1 Sampling in the latent space
 - 2 Decoding to produce the random image



Probabilistic model in latent space

Decoding



Synthesis of random image

Variational autoencoder - variational Bayesian approach

- The Variational Autoencoder (VAE) encourages the latent code z to follow a certain distribution, via the loss function
- This is in turn achieved by using a **Variational Bayesian** approach

Variational autoencoder - variational Bayesian approach

- The Variational Autoencoder (VAE) encourages the latent code z to follow a certain distribution, via the loss function
- This is in turn achieved by using a **Variational Bayesian** approach
- The Variational Bayesian approach is a methodology to approximate the **posterior distribution** of unobserved variables in graphical models
- We will need some tools from statistics : conditional probability, Bayes theorem, marginal distributions, distribution divergences ...
- Keep in mind that the main goal here is to **choose a loss function which will encourage the latent space to follow a probability distribution**

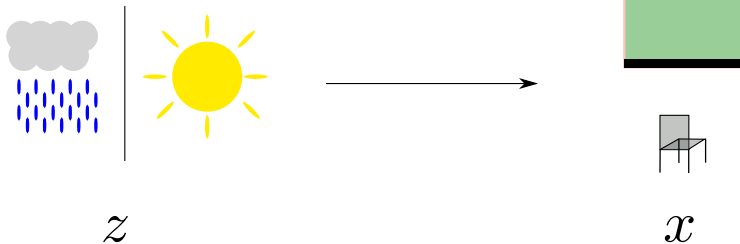
Variational autoencoder - variational Bayesian approach

- Let's take an example. Suppose we have a student A , who comes to lessons or not. The event of A 's presence in the lesson is x . This is the **observed variable**
- We also know that sometimes it rains, and that A 's presence in the class depends on whether it rains or not. Let us denote the event of it raining with z . This is the **latent variable**
- Suppose that we do not directly know whether it is raining (we cannot look out of the window), but can only observe whether the student A is in the lesson



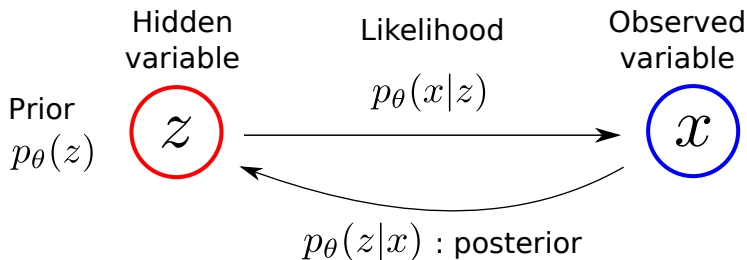
Variational autoencoder - variational Bayesian approach

- The probability of A being in the lesson, $\mathbb{P}(x)$, is the **marginal probability**
- The probability of it raining $\mathbb{P}(z)$ is the **prior probability**
- The probability of A being in the lesson given z , $\mathbb{P}(x|z)$ is the **likelihood**
- The probability of A it raining, given x , $\mathbb{P}(z|x)$ is the **posterior probability**



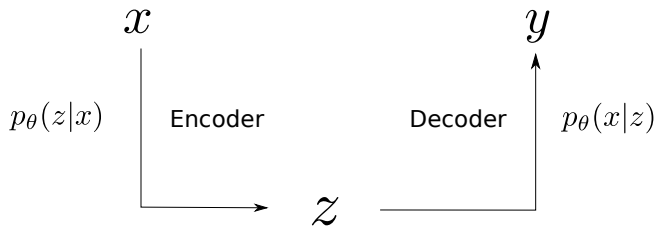
Variational autoencoder - variational Bayesian approach

- At this point, we assume that all the probabilities discussed have a **probability density function**
- We suppose that the distributions come from families of distributions parameterised by the parameters θ ($p_\theta(x)$, $p_\theta(x|z)$ etc)



Variational autoencoder - variational Bayesian approach

- There are some clear analogies with the autoencoder
- Encoder : posterior $p_{\theta}(z|x)$
- Decoder : likelihood $p_{\theta}(x|z)$



- We want to **impose the prior** $p_{\theta}(z)$!!

Variational autoencoder - variational Bayesian approach

- Often, we know (or we can at least estimate) the likelihood $p_{\theta}(x|z)$
- However, often the posterior distribution $p_{\theta}(z|x)$ is difficult to calculate, or intractable. Why is this the case?

Variational autoencoder - variational Bayesian approach

- Often, we know (or we can at least estimate) the likelihood $p_{\theta}(x|z)$
- However, often the posterior distribution $p_{\theta}(z|x)$ is difficult to calculate, or intractable. Why is this the case?

$$\begin{aligned} p_{\theta}(z|x) &= \frac{p_{\theta}(x|z) p_{\theta}(z)}{p_{\theta}(x)} \\ &= \frac{p_{\theta}(x|z) p_{\theta}(z)}{\int p_{\theta}(x, z) dz} \end{aligned}$$

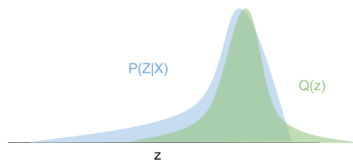
Bayes's rule

Marginal distribution

- $\int p_{\theta}(x, z) dz$ can be a very high-dimensional integral
- Calculating the posterior probability is known as the **inference problem**

Variational autoencoder - variational Bayesian approach

- So, how do we approach the problem of inference?
- The variational Bayesian approach consists in using an **approximate distribution** $q_{\phi}(z|x) \approx p_{\theta}(z|x)$, which is easier to manipulate



- What does it mean for two probability distributions to be **similar**?
 - Often, this is defined using the **Kullback-Leibler divergence**

Variational autoencoder

Kullback-Leibler divergence

Let p and q be two probability distributions defined over the same domain. The Kullback-Leibler divergence is defined as

$$KL(p \parallel q) = \int p(x) \log \frac{p(x)}{q(x)} dx \quad (1)$$

Variational autoencoder

Kullback-Leibler divergence

Let p and q be two probability distributions defined over the same domain. The Kullback-Leibler divergence is defined as

$$KL(p \parallel q) = \int p(x) \log \frac{p(x)}{q(x)} dx \quad (1)$$

The Kullback-Leibler divergence has some interesting properties :

- Non-negative : $KL(p \parallel q) \geq 0$
- $KL(p \parallel q) = 0 \iff p = q$ almost everywhere
- Non-symmetric : $KL(p \parallel q) \neq KL(q \parallel p)$

The second point is quite important. Why? Because we know that **by minimising the KL divergence we are necessarily forcing p and q closer together.**

Variational autoencoder

- Let's come back to variational Bayesian methods now. We wish to approximate $p_\theta(z|x)$ with $q_\phi(z|x)$. To do this, we will find a q_{ϕ^*} :

$$q_{\phi^*} = \arg \min_{q_\phi} KL(q_\phi(z|x) || p_\theta(z|x))^\dagger \quad (2)$$

- Unfortunately, this does not help us much. Why? **Because we don't know $p_\theta(z|x)$!**
- We will have to minimise $KL(q_\phi(z|x) || p_\theta(z|x))$ some other way

[†] You might notice that the q_ϕ before the p_θ . We do not go into the technical reasons here ...

Variational autoencoders - The evidence lower bound

Evidence of Lower Bound (ELBO, lower bound of $\log p_\theta(x)$)

$$\text{ELBO}(q_\phi) = \mathbb{E}_{q_\phi} [\log(p_\theta(x, z))] - \mathbb{E}_{q_\phi} [\log q_\phi(z|x)]$$

- This is known as the “Evidence of Lower Bound” because :

$$\log p_\theta(x) = \text{ELBO}(q_\phi) + KL(q_\phi(z|x) || p_\theta(z|x))$$

- KL divergence is positive : the ELBO is a lower bound for $\log p_\theta(x)$ (the “evidence”)
- **By maximising the ELBO, we minimise the KL divergence between $q_\phi(z|x)$ and $p_\theta(z|x)$**

Variational autoencoders - Variational Autoencoder loss function

- We can rewrite the ELBO in the following manner

$$\text{ELBO}(q_\phi) = \mathbb{E}_{q_\phi} [\log(p_\theta(x|z))] - KL(q_\phi(z|x) || p_\theta(z))$$

- **We know all of these terms!** : we can use it as a VAE loss function !

Variational Autoencoder loss function

$$\mathcal{L}(x; \theta, \phi) = \underbrace{\mathbb{E}_{q_\phi} [\log(p_\theta(x|z))]}_{\text{Reconstruction error}} - \underbrace{KL(q_\phi(z|x) || p_\theta(z))}_{\text{Enforce the prior distribution}}$$

- **Important note!** : we want to **maximise** this loss function !

Variational autoencoders

Variational Autoencoder summary

- 1 We modelled the autoencoding process using a probabilistic (Bayesian) framework, with an observed and a hidden variable
- 2 We wanted to calculate the posterior distribution $p_{\theta}(z|x)$, but this is complicated
- 3 We used an approximation $q_{\phi}(z|x)$ to $p_{\theta}(z|x)$
- 4 We used the ELBO as a loss function to minimise $KL(q_{\phi}(z|x)||p_{\theta}(z))$
 - Ensures a good reconstruction
 - Encourages the latent space to follow our chosen distribution (the prior $p_{\theta}(z)$)

Variational Autoencoder

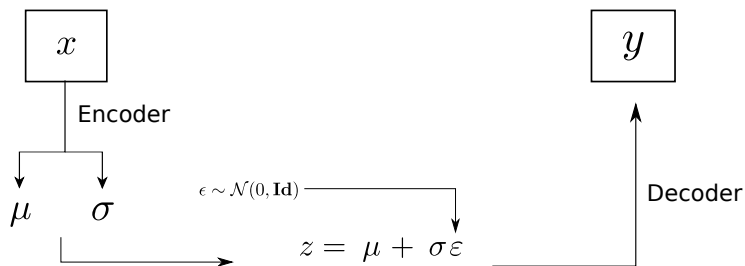
- There remains one more important detail : how to backpropagate through samples of z ? **Random variable, not differentiable**
- Solution : **“reparametrisation trick”**, make the random element an network input
- In the Gaussian case, where q_ϕ is a multivariate Gaussian vector, with mean μ and diagonal covariance matrix σId , this gives

$$z = \mu + \sigma \epsilon, \quad \epsilon \sim \mathcal{N}(0, \text{Id})$$

- μ and σ are produced by the encoder
- Thus, backpropagation can be carried out w.r.t to the parameters ϕ and θ

Variational autoencoder in practice

- The variational autoencoder is actually quite simple to implement
- Take the case of Gaussian $q_\phi(z|x)$

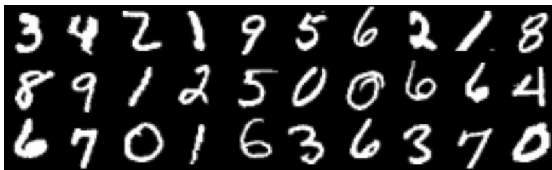


- Encoder and decoder can be MLPs, CNNs ...
- What is the loss in practice?

Variational autoencoder in practice

- Let us take the following case, well-adapted to the **mnist dataset** :
 - Prior : $p_\theta(z) \sim \mathcal{N}(0, \text{Id})$
 - Variational approximation : $q_\phi(z|x) \sim \mathcal{N}(\mu, \sigma \text{Id})$, where $(\mu, \sigma) = E(x)$
 - Likelihood : $p_\theta(x|z) \sim \text{Ber}(y)$, where $y = D(z)$

$$\mathcal{L} = \underbrace{\sum_{i=1}^{mn} x_i \log y_i + (1 - x_i) \log(1 - y_i)}_{\text{Reconstruction error}} - \underbrace{\frac{1}{2} \sum_{j=1}^d (1 + \log(\sigma_j^2) - \mu_j^2 - \sigma_j^2)}_{\text{KL divergence}}$$



Variational autoencoder results

- Some results of variational autoencoders on mnist data : random samples



(a) 2-D latent space

(b) 5-D latent space

(c) 10-D latent space

(d) 20-D latent space

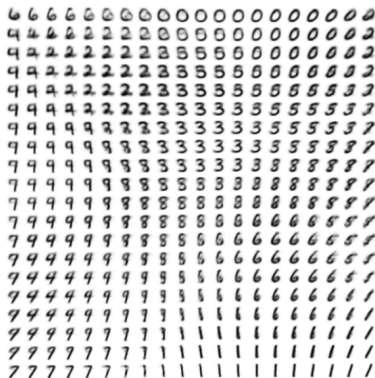
Auto-Encoding Variational Bayes, D. P. Kingma, M. Welling, *arXiv preprint arXiv :1312.6114*, 2013

Variational autoencoder results

- Some results of VAEs on mnist, face data : uniform samples



(a) Learned Frey Face manifold



(b) Learned MNIST manifold

Auto-Encoding Variational Bayes, *D. P. Kingma, M. Welling, arXiv preprint arXiv :1312.6114, 2013*

Variational autoencoder results

- Some results of VAEs on more complex digits data



Variational autoencoder

Variational Autoencoders : summary

- Rigorous framework to autoencode data onto a probabilistically modelled latent space
- Advantages
 - Theoretically-motivated, loss function meaningful
 - Learn to and from mapping (encoder and decoder)
- Drawbacks
 - Have to re-write loss function for each different model, not always easy
 - In practice, do not produce as complex examples as **Generative Adversarial Networks**, which we will see next week

GENERATIVE ADVERSARIAL NETWORKS

Generative Adversarial networks

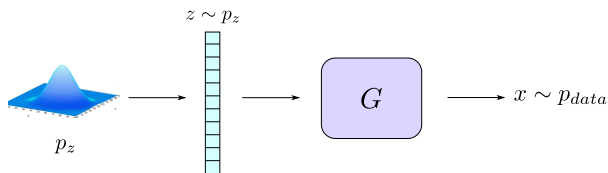
- We saw that variational autoencoders were not straightforward to adapt to new situations
- **Generative adversarial networks** (GANs)* are another generative model that generate random examples of high-dimensional data



* **Generative Adversarial Nets**, *Goodfellow et al*, *NIPS 2014*

Generative Adversarial networks

- The GAN contains only the **decoder** part of an autoencoder
 - The code z is **explicitly sampled** from a chosen distribution p_z (contrary to the VAE)
- The decoder is referred to here as the “Generator”



- We suppose that the data in the database follows a distribution p_{data}
- We want to make the distribution of $x = G(z)$, p_G similar to p_{data} *

* Why can we not do this via the KL divergence as before? Too high dimensionality (previously, we worked in the latent space)

Generative Adversarial networks

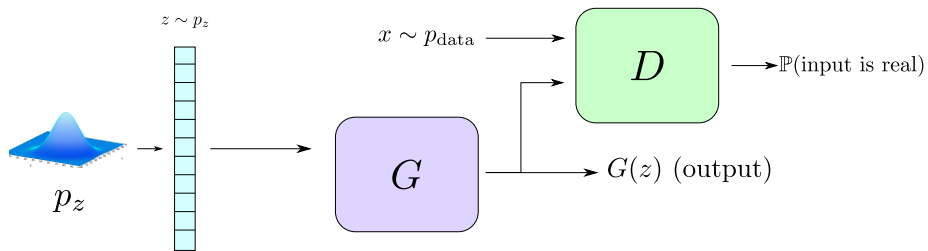
- However, with no reconstruction error, how do we make x look like the data ?
- Answer : Train another network : a **Discriminator** D (or “Adversarial Network”)

Generative Adversarial networks

- However, with no reconstruction error, how do we make x look like the data ?
- Answer : Train another network : a **Discriminator D** (or “Adversarial Network”)
- $D : \mathbb{R}^{mn} \rightarrow [0, 1]$ is trained to **identify “good” (or “true”)** examples of the data
- $G : \mathbb{R}^z \rightarrow \mathbb{R}^{mn}$ is trained to **produce realistic data examples**
- The two networks are trained at the same time, and **each try to fool the other !**

Generative Adversarial networks

- The full GAN architecture looks like this



Generative Adversarial networks

- The discriminator is a really interesting idea, why?
- Reliable and powerful image/data models are difficult to establish
- It is difficult to say whether an image is “good” or not
 - The discriminator acts as a **learned image norm** !
 - It can be used for other purposes also ...
- How is this is achieved? Via a well-designed loss function

Generative Adversarial networks

GAN loss

- Train generator G and the discriminator A in a minmax optimisation problem

$$\min_G \max_D \underbrace{\mathbb{E}_{x \sim p_{data}} [\log D(x)]}_{D \text{ is trying to recognize true data}} + \underbrace{\mathbb{E}_{z \sim p_z} [\log (1 - D(G(z)))]}_{G \text{ is trying to fool } D, \text{ but } D \text{ is trying not to be fooled}}$$

Generative Adversarial networks

GAN loss

- Train generator G and the discriminator A in a minmax optimisation problem

$$\min_G \max_D \underbrace{\mathbb{E}_{x \sim p_{data}} [\log D(x)]}_{D \text{ is trying to recognize true data}} + \underbrace{\mathbb{E}_{z \sim p_z} [\log (1 - D(G(z)))]}_{G \text{ is trying to fool } D, \text{ but } D \text{ is trying not to be fooled}}$$

- Minimisation w.r.t G
 - Second term** is low, $\implies 1 - D(G(z))$ is close to 0 $\implies D$ is recognising $G(z)$ as a true data example : **G has fooled D**

Generative Adversarial networks

GAN loss

- Train generator G and the discriminator A in a minmax optimisation problem

$$\min_G \max_D \underbrace{\mathbb{E}_{x \sim p_{data}} [\log D(x)]}_{D \text{ is trying to recognize true data}} + \underbrace{\mathbb{E}_{z \sim p_z} [\log (1 - D(G(z)))]}_{G \text{ is trying to fool } D, \text{ but } D \text{ is trying not to be fooled}}$$

- Minimisation w.r.t G
 - **Second term** is low, $\implies 1 - D(G(z))$ is close to 0 $\implies D$ is recognising $G(z)$ as a true data example : **G has fooled D**
- Maximisation w.r.t D
 - **First term** is high $\implies D(x)$ is close to 1 : **D is learning to recognize true data**
 - **Second term** is high $\implies 1 - D(G(z))$ is close to 1 : **D is not getting fooled by G**

Generative Adversarial networks

- At the beginning of the training, the examples from G are not very good : D can spot them easily
- At the end of training, the discriminator should not be able to tell the true data from the generated data : $p_G = p_{data}$
- Optimisation alternates between minimisation and maximisation steps

Generative Adversarial networks

- At the beginning of the training, the examples from G are not very good : D can spot them easily
- At the end of training, the discriminator should not be able to tell the true data from the generated data : $p_G = p_{data}$
- Optimisation alternates between minimisation and maximisation steps
- Are we sure that this loss is well-designed for this purpose ?
- In fact, we can prove that this is the case

Generative Adversarial networks

- First, we establish the following lemma :

Optimal GAN discriminator D^*

For a fixed G , the optimal D is $D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$

Generative Adversarial networks

- First, we establish the following lemma :

Optimal GAN discriminator D^*

For a fixed G , the optimal D is $D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$

$$\begin{aligned}\mathcal{L}(G, D) &= \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))] \\ &= \int_x p_{data}(x) \log(D(x)) dx + \int_z p_z(z) \log(1 - D(G(z))) dz \\ &= \int_x p_{data}(x) \log(D(x)) + p_G(x) \log(1 - D(x)) dx.\end{aligned}$$

Generative Adversarial networks

- First, we establish the following lemma :

Optimal GAN discriminator D^*

For a fixed G , the optimal D is $D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$

$$\begin{aligned}\mathcal{L}(G, D) &= \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))] \\ &= \int_x p_{data}(x) \log(D(x)) dx + \int_z p_z(z) \log(1 - D(G(z))) dz \\ &= \int_x p_{data}(x) \log(D(x)) + p_G(x) \log(1 - D(x)) dx.\end{aligned}$$

- For every x , the maximum of the previous equation w.r.t $D(x)$ is

$$D^*(x) = \frac{p_{data}(x)}{p_{data}(x) + p_G(x)}$$

Generative Adversarial networks

- Given this lemma, we can now state the main optimality theorem

Global optimum of the GAN loss function

The global optimum of the GAN loss function is achieved if and only if $p_G = p_{data}$. At this point $\mathcal{L}(G, D) = -\log 4$.

Generative Adversarial networks

- Given this lemma, we can now state the main optimality theorem

Global optimum of the GAN loss function

The global optimum of the GAN loss function is achieved if and only if $p_G = p_{data}$. At this point $\mathcal{L}(G, D) = -\log 4$.

- First, our previous lemma allows us to rewrite the loss function

$$\max_D \mathcal{L}(G, D) = \mathbb{E}_{x \sim p_{data}(x)} [\log D^*(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D^*(G(z)))] \quad (3)$$

$$= \mathbb{E}_{x \sim p_{data}(x)} \left[\log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + \mathbb{E}_{x \sim p_G(x)} \left[\log \left(\frac{p_G(x)}{p_{data}(x) + p_G(x)} \right) \right] \quad (4)$$

Generative Adversarial networks

- Given this lemma, we can now state the main optimality theorem

Global optimum of the GAN loss function

The global optimum of the GAN loss function is achieved if and only if $p_G = p_{data}$. At this point $\mathcal{L}(G, D) = -\log 4$.

- First, our previous lemma allows us to rewrite the loss function

$$\max_D \mathcal{L}(G, D) = \mathbb{E}_{x \sim p_{data}(x)} [\log D^*(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D^*(G(z)))] \quad (3)$$

$$= \mathbb{E}_{x \sim p_{data}(x)} \left[\log \frac{p_{data}(x)}{p_{data}(x) + p_G(x)} \right] + \mathbb{E}_{x \sim p_G(x)} \left[\log \left(\frac{p_G(x)}{p_{data}(x) + p_G(x)} \right) \right] \quad (4)$$

- Therefore, if $p_G = p_{data}$, then $\mathcal{L}(G, D) = \log \frac{1}{2} + \log \frac{1}{2} = \log(-4)$

Generative Adversarial networks

- Now, we are going to show that $\log(-4)$ is the optimal value of the loss function

Generative Adversarial networks

- Now, we are going to show that $\log(-4)$ is the optimal value of the loss function
- First, we remark that :

$$-\log(4) = \mathbb{E}_{x \sim p_{data}(x)} [-\log(2)] + \mathbb{E}_{x \sim p_G(x)} [-\log(2)]. \quad (5)$$

Generative Adversarial networks

- Now, we are going to show that $\log(-4)$ is the optimal value of the loss function
- First, we remark that :

$$-\log(4) = \mathbb{E}_{x \sim p_{data}(x)} [-\log(2)] + \mathbb{E}_{x \sim p_G(x)} [-\log(2)]. \quad (5)$$

- Therefore, by subtracting Equation 5 from Equation 4, we have

$$\begin{aligned} \mathcal{L}(G, D^*) &= -\log(4) + \int p_{data}(x) \log \frac{p_{data}(x)}{\frac{1}{2}(p_{data}(x) + p_G(x))} dx + \\ &\quad \int p_G(x) \log \frac{p_{data}(x)}{\frac{1}{2}(p_{data}(x) + p_G(x))} dx \\ &= -\log(4) + KL\left(p_{data} \parallel \frac{p_{data} + p_G}{2}\right) + KL\left(p_G \parallel \frac{p_{data} + p_G}{2}\right). \end{aligned}$$

Generative Adversarial networks

- This can also be rewritten as

$$\mathcal{L}(G, D^*) = -\log(4) + 2 \mathbf{JSD}(p_{\text{data}} \parallel p_G).$$

- The **JSD** is the **Jensen-Shannon divergence**
- This is another distance between distributions. For p and q , we have :

$$JSD(p, q) = \frac{1}{2}KL\left(p \parallel \frac{1}{2}(p + q)\right) + \frac{1}{2}KL\left(q \parallel \frac{1}{2}(p + q)\right)$$

Generative Adversarial networks

- This can also be rewritten as

$$\mathcal{L}(G, D^*) = -\log(4) + 2 \mathbf{JSD}(p_{\text{data}} \parallel p_G).$$

- The **JSD** is the **Jensen-Shannon divergence**
- This is another distance between distributions. For p and q , we have :

$$JSD(p, q) = \frac{1}{2}KL\left(p \parallel \frac{1}{2}(p + q)\right) + \frac{1}{2}KL\left(q \parallel \frac{1}{2}(p + q)\right)$$

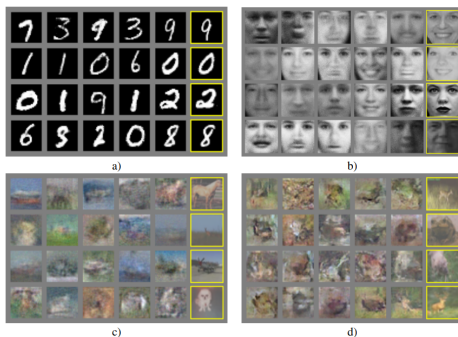
- The JSD is **non-negative and equal to zero if and only if**
 $p_{\text{data}} = p_G$
- Therefore $-\log(4)$ is the optimal value, and only reached when
 $p_{\text{data}} = p_G$

Generative Adversarial networks

- To summarise, we know that **by minimising the GAN loss, we are sure to encourage p_G to approximate p_{data}**
- In spite of this theoretical result (and others), training GANs is very difficult and is a current hot topic of research

Generative Adversarial networks

- Here are some results of the original GAN paper*



- In the space of four years, these results have been vastly improved on
- There are many, many GAN variants. We present a few now

*Generative Adversarial Nets, Goodfellow et al, NIPS 2014

Conditional Generative Adversarial networks

- The Conditional GAN allows a label \mathbf{c} to be added to the loss function
- It is then possible to generate examples of a given class

$$\min_G \max_D [\log D(x|\mathbf{c})] + [\log (1 - D(G(z|\mathbf{c})))]$$

* **Conditional Generative Adversarial Nets**, *Mirza, M. and Osindero, S.*, *arXiv preprint arXiv :1411.1784*, 2014

Generative Adversarial networks

- Examples of results of Conditional GAN



* **Conditional Generative Adversarial Nets**, Mirza, M. and Osindero, S., *arXiv preprint arXiv :1411.1784*, 2014

Generative Adversarial networks

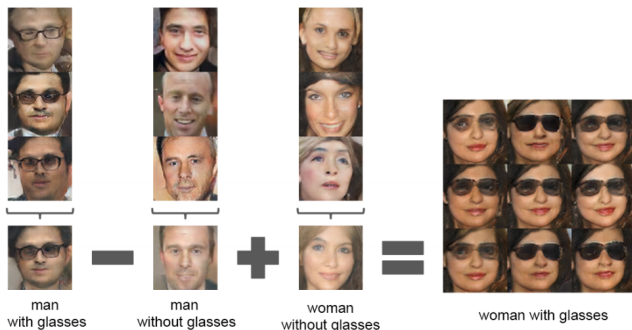
- Deep Convolutional Generative Adversarial Networks (DCGAN)
- More complex data, sharper generation



* **Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks**, Radford, A. and Chintala, S., *arXiv :1511.06434*, 2015

Generative Adversarial networks

- Deep Convolutional Generative Adversarial Networks (DCGAN)
- Linear algebra in the latent space



* **Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks**, Radford, A. and Chintala, S., *arXiv :1511.06434*, 2015

Generative Adversarial networks

- A big problem of the GAN is known as **mode collapse**. This occurs when the training leads a GAN to produce the same result all the time

Generative Adversarial networks

- A big problem of the GAN is known as **mode collapse**. This occurs when the training leads a GAN to produce the same result all the time
- The “Wasserstein GAN”[†] modifies the loss function by using a different distance between probabilities : the Wasserstein distance

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

- $\Pi(p, q)$ is the set of all the joint distributions of (x, y) whose marginals are p and q
- Some sequences of distributions **converge under the Wasserstein distance, but not other distances** (the KL divergence, for example)
- Using this formulation stabilises the GAN training

[†]Wasserstein GAN, Ajovsky, M., Chintala, S. and, Bottou, L. arXiv preprint arXiv :1701.07875, 2017

Generative Adversarial networks

- Very recently, extremely realistic, high-resolution results have been achieved by different research teams[†]



[†] **Progressive growing of gans for improved quality, stability, and variation**, Karras, T., Aila, T., Laine, S., and Lehtine, J., *arXiv preprint arXiv :1710.10196*, 2017

Generative Adversarial networks

- Very recently, extremely realistic, high-resolution results have been achieved by different research teams



[†] **Progressive growing of gans for improved quality, stability, and variation**, Karras, T., Aila, T., Laine, S., and Lehtine, J., *arXiv preprint arXiv :1710.10196*, 2017

Generative Adversarial networks

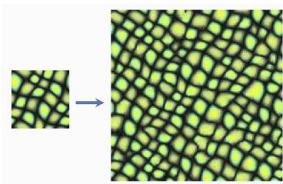
Summary on GANs

- GANs are a powerful and generic way of producing **random examples of complex images**
- However, they are **notoriously difficult to train**, this is a current area of research
- A significant disadvantage with respect to variational autoencoders is that **GANs do not train an encoder** : it is a one-way transformation
 - Often, for restoration or other inverse problems, it is useful to have an “inverse” transformation

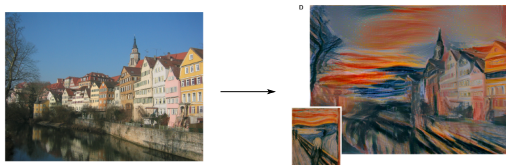
TEXTURE SYNTHESIS AND STYLE TRANSFER MODELS

Texture synthesis and style transfer

- We have seen two methods for creating random examples of complex images
- There is another type of generative model specifically designed for **texture synthesis and style transfer**
- These are two very common tasks in image and video editing



Texture synthesis



Style transfer

Texture synthesis and style transfer

- This set of methods is based on the work of Gatys et al.^{†,‡}, similar approach as **deep dream** and variants
- We suppose that texture and style are **coded somewhere in the layers of a neural network**
 - Small patterns, repetitive motifs
- We will iterate gradient descent on an image to encourage similar representations inside the network



$$x^{n+1} = x^n + \nabla_x \mathcal{L}$$



[†] **Texture synthesis using convolutional neural networks**, Gatys, L. A., Ecker, A. S, and Bethge, M., *NIPS*, 2015

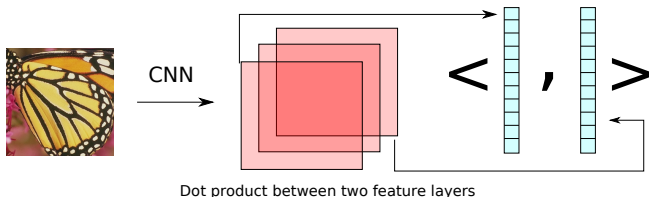
[‡] **A Neural Algorithm of Artistic Style**, Gatys, L. A., Ecker, A. S, and Bethge, M., *arXiv :1508.06576*, 2015

Texture synthesis

- First, let's take the case of **texture synthesis**, suppose we have an image \hat{x} whose texture we want to copy
- Let $w_{i,j}^\ell$ be the response of a neural network (often a classification CNN) to an image at layer ℓ for channel i at position j

Texture synthesis

- First, let's take the case of **texture synthesis**, suppose we have an image \hat{x} whose texture we want to copy
- Let $u_{i,j}^\ell$ be the response of a neural network (often a classification CNN) to an image at layer ℓ for channel i at position j
- We define $C_{i,j}^\ell = \sum_k u_{i,k}^\ell u_{j,k}^\ell$, the dot product between two channels
- This represents the **correlations between different channels** in the CNN : a characteristic of the texture



Texture synthesis

- Why is this quantity characteristic of texture? Textures are **stationary random processes**



- This means that the statistical properties of textures **should not depend on spatial location**[†]
- Therefore, we carry out the dot product to remove spatial dependency

[†] **A Parametric Texture Model Based on Joint Statistics of Complex Wavelet Coefficients**, Portilla, J and Simoncelli, E. P, *International Journal of Computer Vision*, 2000

Texture synthesis and style transfer

- We try to find an output image y which minimises the following loss :

$$\mathcal{L}_{\text{Texture}}(y; \hat{x}) = \sum_{\ell} w_{\ell} \sum_{i,j} \frac{1}{2} \left(C_{i,j}^{\ell} - \hat{C}_{i,j}^{\ell} \right)^2$$

- w_{ℓ} is a weighting function for each layer

Texture synthesis algorithm

$y^0 \leftarrow$ random white noise

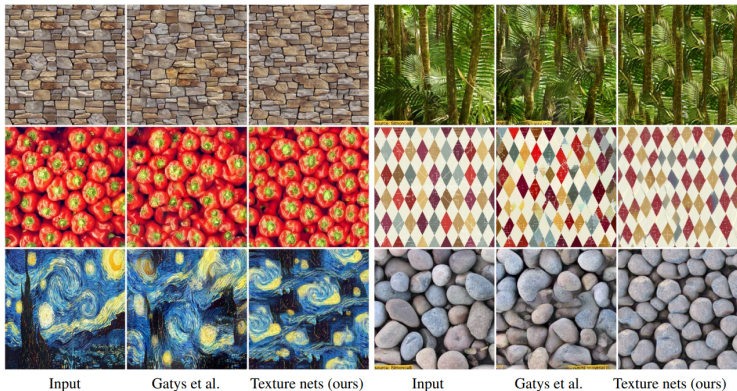
for $i = 1$ to N **do**

$y^{n+1} = y^n - \varepsilon \nabla_x \mathcal{L}_{\text{Texture}}(y^n)$

end for

Texture synthesis

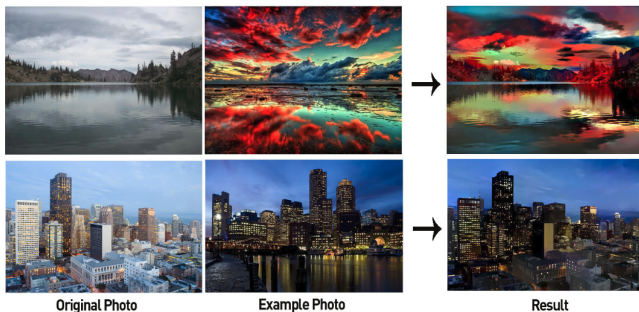
- Here are some image texture synthesis examples with this method, and variants



† **Texture Networks : Feed-forward Synthesis of Textures and Stylized Image**, *Ulyanov et al., arXiv, 2016*

Style transfer

- Now, we look at the case of **style transfer**. We have two images :
 - \tilde{x} whose **content** we wish to copy
 - \hat{x} whose **style** we wish to copy
- Contrary to texture, **content is spatially dependent**



† Image from **Deep Photo Style Transfer**, Luan et al., arXiv :1703.07511v3, 2017

Style transfer

- We define a content loss :

$$\mathcal{L}_{\text{Content}}(y; \tilde{x}, \ell) = \sum_{i,j} \left(u_{i,j}^{\ell} - \tilde{u}_{i,j}^{\ell} \right)^2$$

- The final loss is a mixture of content and style (texture) losses :

$$\mathcal{L}(y; \tilde{x}, \hat{x}) = \alpha \mathcal{L}_{\text{content}}(y; \tilde{x}) + \beta \mathcal{L}_{\text{texture}}(y; \hat{x})$$

- The algorithm to produce the end result is the same : random initialisation and iterate gradient descent

Style transfer - some results[†]



[†] <https://github.com/lengstrom/fast-style-transfer/>

Texture synthesis and style transfer

Summary of texture synthesis and style transfer

- Inter-channel correlations of neural network layers provide a powerful model of texture/style
- This gives a **flexible, easy-to-implement** algorithm : random initialisation and iterate gradient descent
- Main drawback : only applicable to a restricted class of problems (texture, style)

SUMMARY

Conclusion

We have seen three types of generative models

- 1 Variational autoencoders
 - An Autoencoder whose latent space is encouraged to follow a certain distribution
 - Variational Bayesian formulation
- 2 Generative Adversarial Networks
 - A generator trained to create realistic data
 - A discriminator trained to identify true and false data examples
- 3 Models for texture synthesis and style transfer
 - Iterative method to encourage similar intra-layer correlations of neural network
 - Start with an initial point, iterate gradient descent w.r.t image

Summary of advantages and weaknesses of Generative Models

Model/method	Advantages	Disadvantages
VAE	<ul style="list-style-type: none">• Rigorous formulation• Encoder and decoder trained	<ul style="list-style-type: none">• Loss must be rewritten for different probability distributions (not easy)• In practice, more limited applications than GANs
GAN	<ul style="list-style-type: none">• Highly flexible• Applied to complex data, impressive results	<ul style="list-style-type: none">• Difficult to train• No reverse transformation (encoder)
Texture/style model	<ul style="list-style-type: none">• Versatile, simple algorithm• Produces best texture/style results	<ul style="list-style-type: none">• Only applicable to limited cases

References

A few references

- Kingma, Diederik, and Welling, **Auto-encoding Variational Bayes**, arXiv :1312.6114, 2013
- Goodfellow et al, **Generative Adversarial Nets**, NIPS 2014
- Gatys, L. A., Ecker, A. S, and Bethge, M., **Texture synthesis using convolutional neural networks**, NIPS, 2015
- Gatys, L. A., Ecker, A. S, and Bethge, M., **A Neural Algorithm of Artistic Style**, arXiv :1508.06576, 2015
- Radford, A. and Chintala, S., **Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks**, arXiv :1511.06434, 2015
- Zhu, Krähenbühl, Schechtman, Efros, **Generative Visual Manipulation on the Natural Image Manifold**, ECCV,2016
- Karras, T., Aila, T., Laine, S., and Lehtine, J., **Progressive growing of gans for improved quality, stability, and variation**, arXiv preprint arXiv :1710.10196, 2017