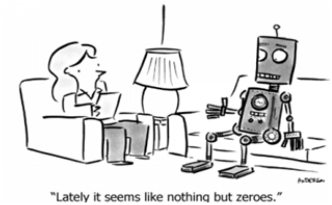
Transformers DNN

Gianni FRANCHI
ENSTA-Paris



ViT [web1]

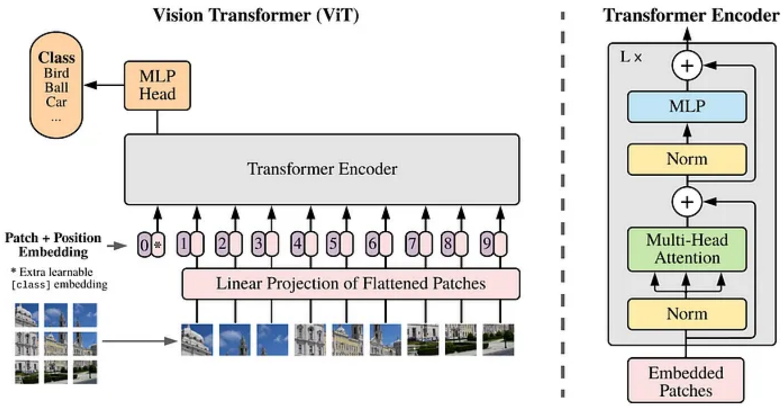
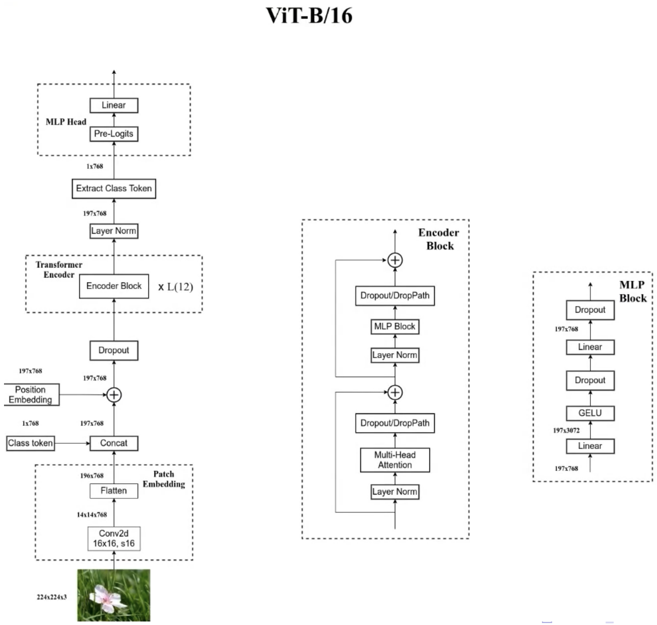


Figure: Representation structure of ViT

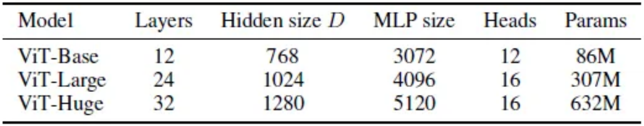
ViT [web1]



ViT-B/16
MLP Head
Linear
Pre-Logits
1x768
Extract Class Token
197x768
Layer Norm
Transformer Encoder
Encoder Block
x L(12)
Dropout
197x768
197x768
Position Embedding
1x768
197x768
Class token
Concat
196x768
Patch Embedding
Flatten
14x14x768
Conv2d 16x16, s16
224x224x3
Encoder Block
Dropout/DropPath
MLP Block
Layer Norm
Dropout/DropPath
Multi-Head Attention
Layer Norm
MLP Block
Dropout
197x768
Linear
Dropout
GELU
197x3072
Linear
197x768

ViT [web1]

ViT B corresponds to ViT base, **ViT L** corresponds to ViT large, and ViT H corresponds to ViT huge. patch size is the size of the image slice (there are also in the source code) 32×32); layers is the number of times the encoder block is stacked; Hidden size is the length of the token vector; The MLP size is four times the hidden size, that is, the number of nodes in the first full connection layer of the MLP block in the encoder block; Heads is the number of heads in multi head attention.



Model	Layers	Hidden size D	MLP size	Heads	Params
ViT-Base	12	768	3072	12	86M
ViT-Large	24	1024	4096	16	307M
ViT-Huge	32	1280	5120	16	632M

Analyzing layer representations of CNNs vs VIT [Raghu2021]

Analyzing (hidden) layer representations of neural networks is challenging because their features are distributed across a large number of neurons. So they propose to study a kind of correlation between on layer X and one layer Y.

Analyzing layer representations of CNNs vs VIT [Raghu2021]

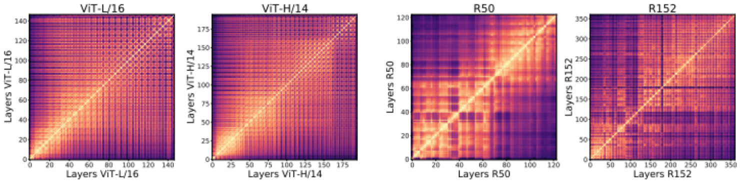


Figure: Representation structure of ViTs and convolutional networks show significant differences, with **ViTs having highly similar representations throughout the model,** while the **ResNet** models show **much lower similarity between lower and higher layers**

Analyzing layer representations of CNNs vs ViT [Raghu2021]

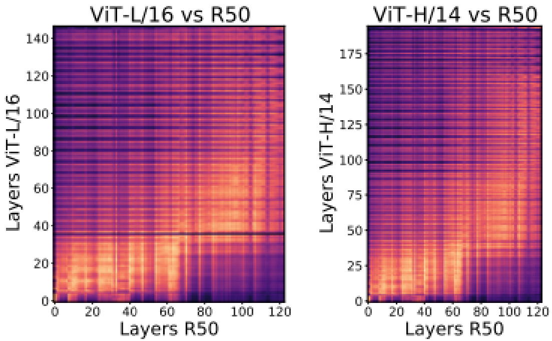


Figure: Representation structure of ViTs vs ResNet illustrate that a **larger number of lower layers in the ResNet are similar to a smaller set of the lowest ViT layers**

Local and Global Information in Layer Representations [Raghu2021]

How much global information is aggregated by early self-attention layers in ViT?

Analyzing Attention Distances:

Each self-attention layer comprises multiple self-attention heads, and for each head we can compute the average distance between the query patch position and the locations it attends to. This reveals how much local vs global information each self-attention layer is aggregating for the representation. Specifically, they **weight the pixel distances by the attention weights** for each attention head and average over 5000 datapoints.

Local and Global Information in Layer Representations [Raghu2021]

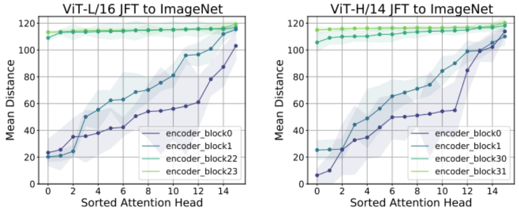


Figure: With less training data, lower attention layers do not learn to attend locally.

Local and Global Information in Layer Representations [Raghu2021]

We observe that even in the lowest layers of ViT, self-attention layers have a mix of local heads (small distances) and global heads (large distances). This is in contrast to CNNs, which are hardcoded to attend only locally in the lower layers.

Effective Receptive Fields **[Raghu2021]**

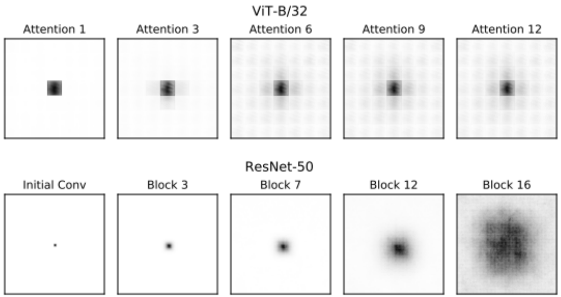


Figure: ResNet effective receptive fields are highly local and grow gradually; ViT effective receptive fields shift from local to global.

ViT vs CNN [Ghiasi2022]

They show that :

- patch-wise image activation patterns for ViT features essentially behave like saliency maps
- the behavior of ViTs and CNNs, finding that ViTs make better use of background information and rely less on high-frequency, textural attributes.
- investigate the effect of natural language supervision with CLIP on the types of features extracted by ViTs. They find CLIP-trained models include various features clearly catered to detecting components of images corresponding to caption text, such as prepositions, adjectives, and conceptual categories.

What is CLIP [Radford2021]

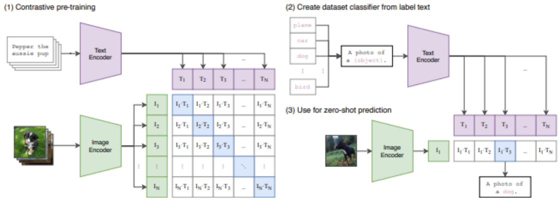
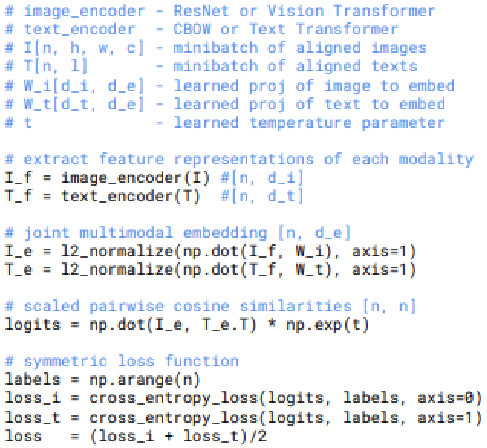


Figure: Summary of our approach. While standard image models jointly train an image feature extractor and a linear classifier to predict some label, CLIP jointly trains an image encoder and a text encoder to predict the correct pairings of a batch of (image, text) training examples. At test time the learned text encoder synthesizes a zero-shot linear classifier by embedding the names or descriptions of the target dataset's classes.

What is CLIP [Radford2021]



```
# image_encoder - ResNet or Vision Transformer
# text_encoder  - CBOW or Text Transformer
# I[n, h, w, c] - minibatch of aligned images
# T[n, l]       - minibatch of aligned texts
# W_i[d_i, d_e] - learned proj of image to embed
# W_t[d_t, d_e] - learned proj of text to embed
# t             - learned temperature parameter

# extract feature representations of each modality
I_f = image_encoder(I) #[n, d_i]
T_f = text_encoder(T)  #[n, d_t]

# joint multimodal embedding [n, d_e]
I_e = l2_normalize(np.dot(I_f, W_i), axis=1)
T_e = l2_normalize(np.dot(T_f, W_t), axis=1)

# scaled pairwise cosine similarities [n, n]
logits = np.dot(I_e, T_e.T) * np.exp(t)

# symmetric loss function
labels = np.arange(n)
loss_i = cross_entropy_loss(logits, labels, axis=0)
loss_t = cross_entropy_loss(logits, labels, axis=1)
loss   = (loss_i + loss_t)/2
```

Figure: Numpy-like pseudocode for the core of an implementation of CLIP.

What is CLIP [Radford2021]

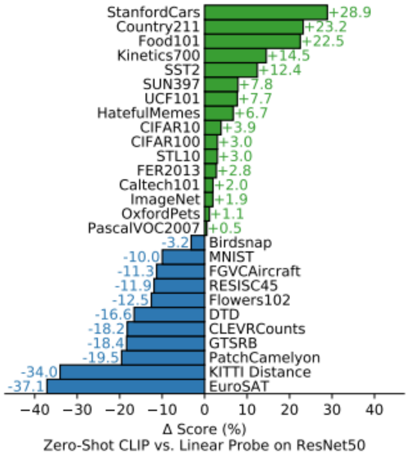


Figure: **Zero-shot CLIP is competitive with a fully supervised baseline.** Across a 27 dataset eval suite, a zero-shot CLIP classifier outperforms a fully supervised linear classifier fitted on ResNet-50 features on 16 datasets, including ImageNet

ViT representation [Ghiasi2022]

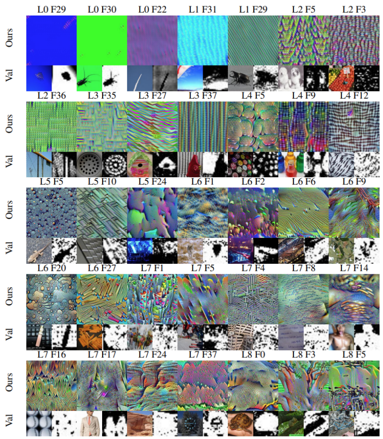


Figure 17: Visualization of ViT-base-patch16

Figure: Visualization of ViT-base-patch16

ViT representation [Ghiasi2022]

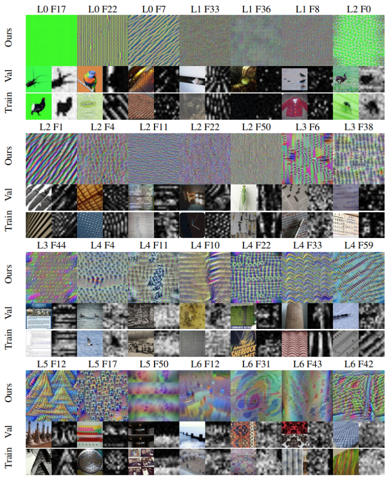


Figure: Visualization of a CLIP model with ViT-base-patch16 as its visual part.

Bibliography

- **[web1]** https://programmer.group/613ada5f581ff.html
- **[Raghu2021]** Raghu, M., Unterthiner, T., Kornblith, S., Zhang, C., & Dosovitskiy, A. (2021). Do vision transformers see like convolutional neural networks?. Advances in Neural Information Processing Systems, 34, 12116-12128.
- **[Ghiasi2022]** Ghiasi, A., Kazemi, H., Borgnia, E., Reich, S., Shu, M., Goldblum, M., ... & Goldstein, T. (2022). What do Vision Transformers Learn? A Visual Exploration. arXiv preprint arXiv:2212.06727.
- **[Radford2021]** Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., ... & Sutskever, I. (2021, July). Learning transferable visual models from natural language supervision. In International conference on machine learning (pp. 8748-8763). PMLR.

Introduction to Generative Adversarial Networks (GAN)

IA716 - Perception pour les systemes autonomes

Gianni Franchi

24/04/2023

Approximate Bayesian Computation

Let us consider that we have a set $\{X_i\}$ of data that follow a distribution $P(X)$. Our goal is to generate new data from this distribution but.
Let us assume we have access to a model of distribution $P(X/\theta)$ then we can generate new data X. This distribution is called the likelihood.
However, for particular problems, we may find that **we can not express the likelihood** in closed-form, or it is prohibitively costly to compute it.

Approximate Bayesian Computation

A solution is to approximate the likelihood.
We aim at using a function $\delta(\cdot)$ to obtain a practically good enough approximation to the true likelihood:

$$\lim_{\epsilon \to 0} \delta(X, \hat{X}, \epsilon) = P(X/\theta)$$

We introduce a tolerance parameter ϵ because the chance of generating a synthetic data-set $\hat{X}$ being equal to the observed data X is virtually null for most problems

GAN principle: Why Generative learning

We've only seen discriminative models in the past

- Given an image X, predict a label Y;
- Estimate $P(Y|X)$.

Discriminative models have several key limitations

- Can't model $P(X)$, i.e. the probability of seeing a certain image;
- Thus, can't sample from $P(X)$, i.e. can't generate new images;
- Fixed loss.

Generative models (in general) cope with all of above

- Can model $P(X)$
- Can generate new images.
- Learned loss link with perception (perceptual loss)

GAN principle

In generative adversarial networks (GANs), the task of learning a generative model is expressed as a two-player zero-sum game between two networks.

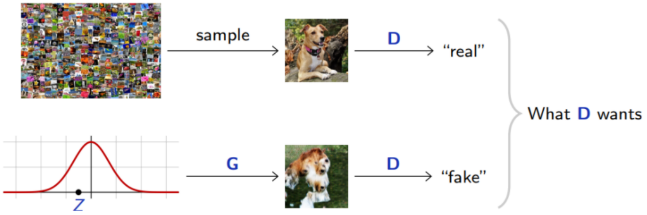


Figure: Principle of the GAN **[Goodfellow2014]**

With a random noise.

GAN principle[1]

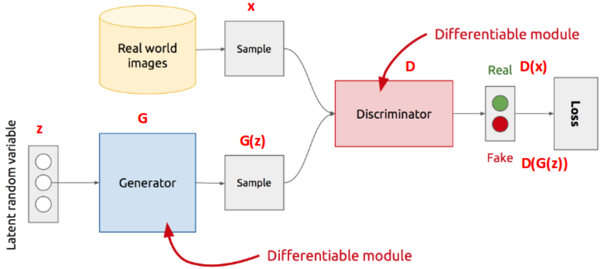


Figure: Principle of the GAN **[Goodfellow2014]**

We need two DNNs.

[1] Gilles Louppe

GAN principle : training **[Goodfellow2014]**

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

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

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

GAN principle : training [Goodfellow2014]

For a fixed generator g, the discriminator d can be trained by generating a two-class training set
$\mathrm{d} = \{(\mathrm{x}_1, y = 1), ..., (\mathrm{x}_N, y = 1), (g(\mathrm{z}_1; \theta), y = 0), ..., (g(\mathrm{z}_N; \theta), y = 0)\}$,
and minimizing the cross-entropy loss

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

However, the situation is slightly more complicated since we also want to train g to fool the discriminator, which is equivalent to maximize d's loss.

GAN principle : training **[Goodfellow2014]**

Let us consider the value function

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

For a fixed g, $V(\phi, \theta)$ is high if d is good at recognizing true from generated samples.
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.

GAN principle : training **[Goodfellow2014]**

Therefore, the ultimate goal is optmized this minimax loss:

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

GAN principle : training [Goodfellow2014]

Here I change $d(x;\phi)$ by $D(x)$ and also $g(x;\theta)$ by $G(x)$

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

$$V(G,D) = \mathbb{E}_{\mathrm{x}\sim p(\mathrm{x})}\left[\log D(\mathrm{x})\right] + \mathbb{E}_{\mathrm{z}\sim p(\mathrm{z})}\left[\log(1 - D(G(\mathrm{z})))\right]$$

Remember : If we have

- I a real interval;
- $\varphi : [a,b] \to I$ a derivable function and whose derivative has an integral;
- I a real interval;
- $f : I \to$ R a continuous function.

then :

$$\int_a^b f(\varphi(t))\varphi'(t)\,\mathrm{d}t = \int_{\varphi(a)}^{\varphi(b)} f(x)\,\mathrm{d}x.$$

GAN principle : training **[Goodfellow2014]**

So

$$\mathbb{E}_{z\sim p(z)}\left[\log(1 - D(G(z)))\right] = \mathbb{E}_{x\sim p_g(x)}\left[\log(1 - D(x))\right]$$

So we have

$$V(G,D) = \int_x (p(x)\log(D(x)) + p_g(x)\log(1 - D(x)))dx$$

Let us fix the generator and look for the best discriminator D^*

$$\frac{\partial V(G,D)}{\partial D} = \frac{\partial}{\partial D}\int_x (p(x)\log(D(x)) + p_g(x)\log(1 - D(x)))dx$$

GAN principle : training [Goodfellow2014]

We want to find

$$\frac{\partial}{\partial D} p \log(D) + p_g \log(1 - D) = 0$$

$$\frac{p}{D} - \frac{p_g}{(1 - D)} = 0$$

$$\frac{D}{(1 - D)} = \frac{p_g}{p}$$

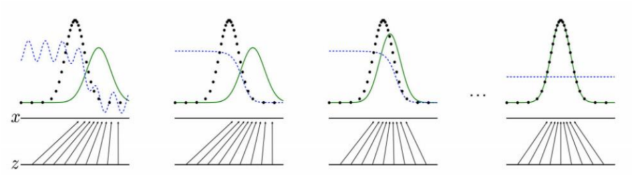
$$D = \frac{p_g}{p + p_g}$$

When the generator is perfectly driven $p = p_g$ then $D = 1/2$

GAN principle : training **[Goodfellow2014]**

Optimal solution to the adversarial game:

- Generator distribution = data distribution
- D = 1/2

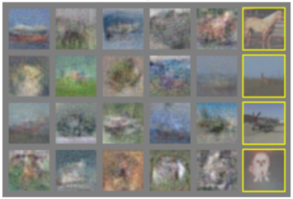


GAN results

Results

a)

b)


c)

d)

GAN issues:

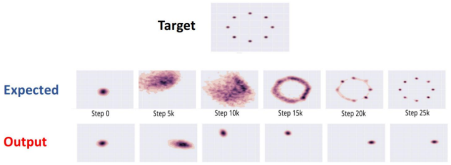
Disadvantages of Generative Adversarial Networks (GANs):

- **Training Instability:** GANs can be difficult to train, with the risk of instability, **mode collapse**, or **failure to converge**.
- **Computational Cost:** GANs can require a lot of computational resources and can be slow to train, especially for high-resolution images or large datasets.
- **Overfitting:** GANs can overfit the training data, producing synthetic data that is too similar to the training data and lacking diversity.
- **Bias and Fairness:** GANs can reflect the biases and unfairness present in the training data, leading to discriminatory or biased synthetic data.
- **Interpretability and Accountability:** GANs can be opaque and difficult to interpret or explain, making it challenging to ensure accountability, transparency, or fairness in their applications.

GAN issues: Mode collapse [Srivastava2017]

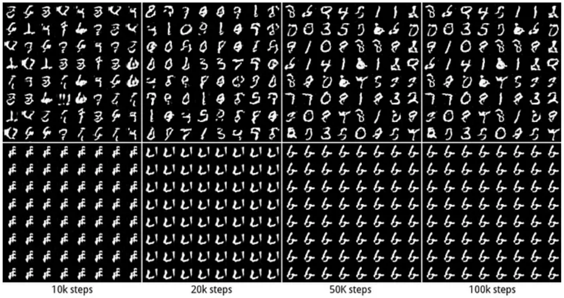
Training a standard GAN often results in pathological behaviors:

- Oscillations without convergence: contrary to standard loss minimization, alternating stochastic gradient descent has no guarantee of convergence.
- Vanishing gradients: when the classifier **d** is too good, the value function saturates and we end up with no gradient to update the generator.
- 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.



GAN issues: Mode collapse **[jonathan-hui]**

Real-life data distributions are multimodal. For example, in MNIST, there are 10 major modes from digit '0' to digit '9'. The samples below are generated by two different GANs. The top row produces all 10 modes while the second row creates a single mode only (the digit '6'). This problem is called mode collapse when only a few modes of data are generated.



GAN issues: Mode collapse **[jonathan-hui]**

The objective of the GAN generator is to create images that can fool the discriminator D the most.

GAN issues: Mode collapse **[jonathan-hui]**

But let's consider one extreme case where G is trained extensively without updates to D. The generated images will converge to find the optimal image x^* that fool D the most, the most realistic image from the discriminator perspective. In this extreme, x^* will be independent of z.

GAN issues: Mode collapse **[jonathan-hui]**

When we restart the training in the discriminator, the most effective way to detect generated images is to detect this single mode. Since the generator desensitizes the impact of z already, the gradient from the discriminator will likely push the single point around for the next most vulnerable mode. This is not hard to find. The generator produces such an imbalance of modes in training that it deteriorates its capability to detect others. Now, both networks are overfitted to exploit short-term opponent weakness.

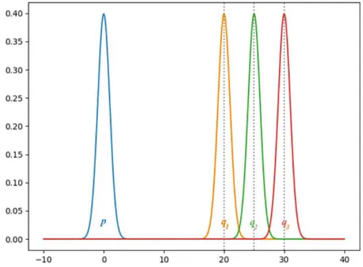
GAN issues: Vanishing Gradient **[jonathan-hui]**

Recall that when the discriminator is optimal, the objective function for the generator is:

$$V(G, D^*) = 2D_{JS}\left[p||p_g\right] + cst$$

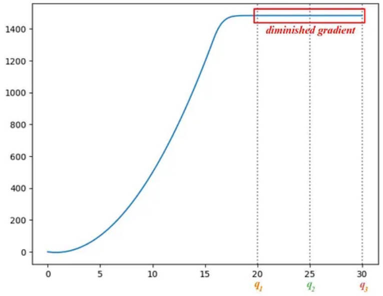
GAN issues: Vanishing Gradient **[jonathan-hui]**

Let's consider an example in which p and p_g are Gaussian distributed and the mean of p is zero. Let's consider p_g with different means to study the gradient of $D_{JS}\left[p||p_g\right]$. We denote these distribution q_1, q_2, q_3



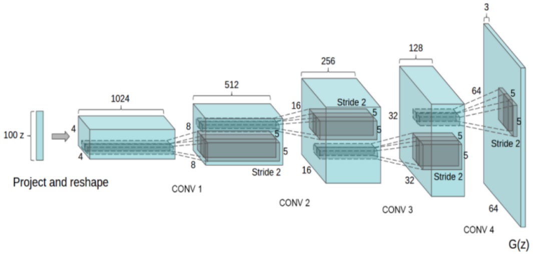
GAN issues: Vanishing Gradient **[jonathan-hui]**

As shown below, the gradient for the JS-divergence vanishes from q_1 to q_3. The GAN generator will learn extremely slow to nothing when the cost is saturated in those regions.



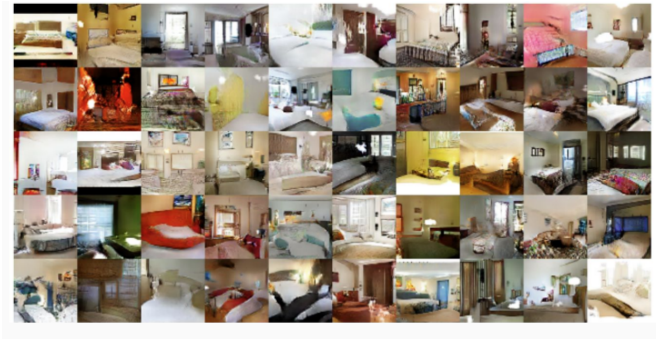
DCGAN [Radford2015]

Replace FC hidden layers with Convolutions/deconvolutional layers.



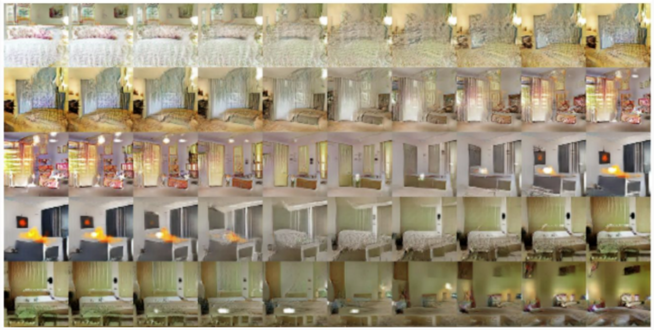
DCGAN [Radford2015]

Generations of realistic bedrooms pictures, from randomly generated latent variables



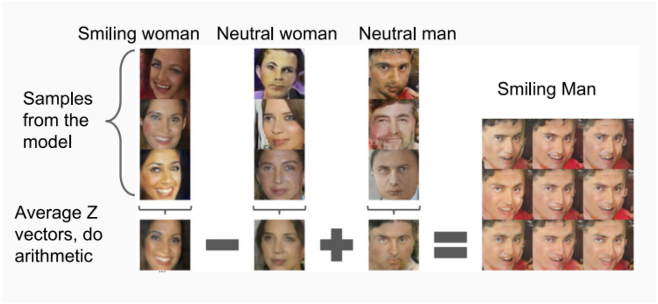
DCGAN [Radford2015]

Interpolation in between points in latent space.



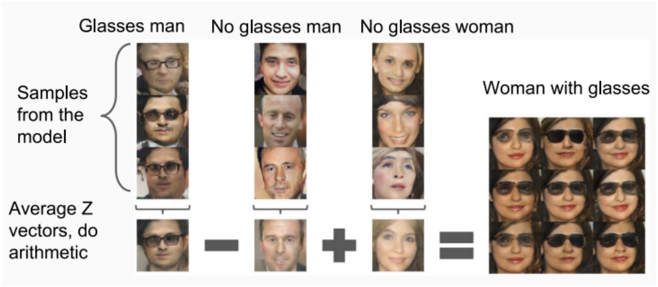
DCGAN [Radford2015]

Convolutional GAN - Arithmetic



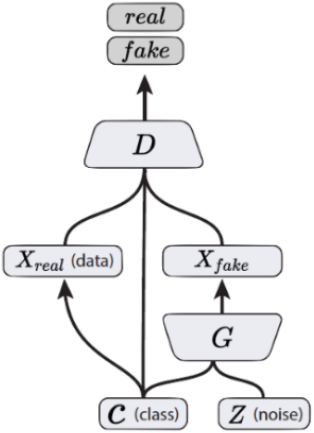
DCGAN [Radford2015]

Convolutional GAN - Arithmetic



Conditional GAN **[Mirza2014]**

Conditional generative adversarial network, or cGAN, is a type of GAN that involves the conditional generation of images by a generator model. Hence you can control the kind of output you want



Optimal transport **[alexhwilliams]**

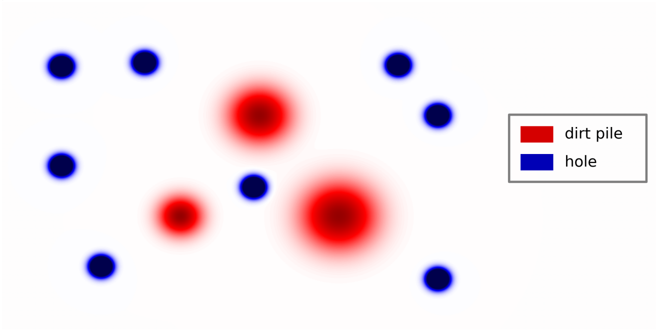
A fundamental problem in statistics and machine learning is to come up with useful measures of 'distance' between pairs of probability distributions. One can compute the Kullback-Leibler (KL) divergence from Q to P is defined by :

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right)$$

While the KL divergence is incredibly useful and fundamental in information theory, it also has its shortcomings.
For instance, one of the first things we learn about the KL divergence is that it is not symmetric A bigger problem is that the divergence may be infinite if the support of P and Q are not equal.

Optimal transport **[alexhwilliams]**

One of the nice aspects of optimal transport theory is that it can be grounded in physical intuition through the following thought experiment. Suppose we are given the task of filling several holes in the ground. The image below shows an overhead 2D view of this scenario - the three **red regions** correspond to dirt piles, and the **eight blue regions** correspond to holes.



Optimal transport **[alexhwilliams]**

Our goal is to come up with the most efficient transportation plan to which moves the dirt to fill all the holes. We assume **the total volume of the holes** is equal to **the total volume of the dirt piles**.

Optimal transport **[alexhwilliams]**

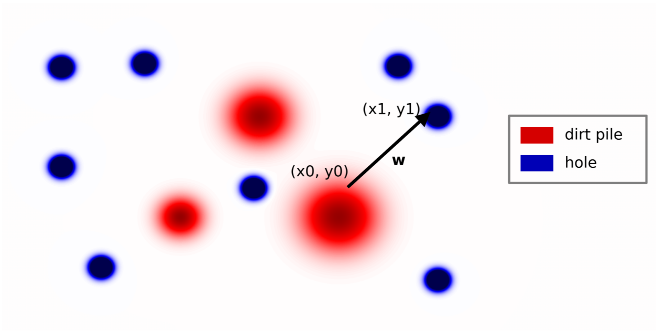
The "most efficient" plan is the one that minimizes the total transportation cost. To quantify this, let's say the transportation cost C of moving 1 unit of dirt from $(x_0, y_0) \to (x_1, y_1)$ is given by the squared Euclidean distance:

$$C(x_0, y_0, x_1, y_1) = (x_0 - x_1)^2 + (y_0 - y_1)^2$$

Now we'll define the transportation plan T, which tells us how many units of dirt to move from $(x_0, y_0) \to (x_1, y_1)$ which is given by

$$T(x_0, y_0, x_1, y_1) = \omega$$

Optimal transport **[alexhwilliams]**



Optimal transport **[alexhwilliams]**

The transportation plan, T, specifies an arrow like this from every possible starting position to every possible destination. Further, in addition to being non negative, the plan must satisfy the following two conditions:

$$\int\int T(x_0, y_0, x, y) dxdy = p(x_0, y_0) \ \forall x_0, y_0$$

$$\int\int T(x, y, x_1, y_1) dxdy = q(x_1, y_1) \ \forall x_1, y_1$$

Where $p(\cdot,\cdot)$ and $q(\cdot,\cdot)$ are density functions encoding the units of dirt and hole depth at each 2D location. Intuitively, the first constraint says that the amount of piled dirt at is "used up" or transported somewhere. The second constraint says that the hole at is "filled up" with the required amount of dirt (no more and no less).

Optimal transport **[alexhwilliams]**

Suppose we are given a function T that satisfies all of these conditions (i.e. we are given a feasible transport plan). Then the overall transport cost is given by:

$$\text{total cost} = \int\int\int\int C(x_0, y_0, x_1, y_1)\, T(x_0, y_0, x_1, y_1)\, dx_0\, dy_0\, dx_1\, dy_1$$

Xe multiply the amount of dirt transported, given by T, by the per unit transport cost, given by C. Integrating over all possible origins and destinations gives us the total cost.

Optimal transport **[alexhwilliams]**

Given two probability distributions μ_0 and μ_1, and a positive cost function $c : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}^+$ The Wasserstein distances is given by

$$OP(\mu_0, \mu_1) = \inf_{\gamma \in \Pi(\boldsymbol{\mu_0}, \boldsymbol{\mu_1})} \int c(x, y) \, \mathrm{d}\gamma(x, y)$$

where $\Pi(\mu_0, \mu_1)$ is the set of probability distributions γ with marginal distributions μ_0 and μ_1.

Optimal transport **[Arjovsky2017]**

When using $c(x, y) = \|x - y\|^p$ one defines Wasserstein distances. The p-Wasserstein distance W^p between μ_0 and $\mu_1$is defined as

$$W^p(\mu_0, \mu_1) = \inf_{\gamma \in \Pi(\mu_\mathbf{0}, \mu_\mathbf{1})} \int \|x - y\|^p \, \mathrm{d}\gamma(x, y)$$

Which is similar to solve the dual following dual problem for $p = 1$

$$W^p(\mu_0, \mu_1) = \sup_{\phi \in \mathrm{Lip}_\mathbf{1}} \left[E_{x \sim \mu_\mathbf{0}}(\phi(x)) - E_{x \sim \mu_\mathbf{0}}(\phi(x)) \right]$$

with

$$\mathrm{Lip}_1 = \{ f : \mathbb{R}^D \to \mathbb{R} \text{ such that } \forall (x, y) \|f(x) - f(y)\| \leq \|x - y\| \}$$

Wassestein GAN **[Arjovsky2017]**

GAN (Vanilla):

$$\min_{D_\theta} \max_{G_\phi} \mathbb{E}_{\mathsf{x} \sim p(\mathsf{x})} [\log D(\mathsf{x})] + \mathbb{E}_{\mathsf{z} \sim p(\mathsf{z})} [\log(1 - D(G(\mathsf{z})))]$$

Wassestein GAN :

$$\min_{D_\theta} \max_{G_\phi} \mathbb{E}_{\mathsf{x} \sim p(\mathsf{x})} [D(\mathsf{x})] - \mathbb{E}_{\mathsf{z} \sim p(\mathsf{z})} [(D(G(\mathsf{z})))]$$

We just got rid of the log and D is not a probability... but we now have a constrained optimization $D \in \mathrm{Lip}_1$ The original WGAN paper uses weight clipping to restrict the Lipschitz constant (heuristic)

Cycle GAN [geeksforgeeks]

We use two Conditional GAN:

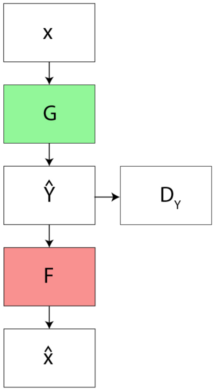
$$G_{Y \to X} : Y \to X$$

$$G_{X \to Y} : X \to Y$$

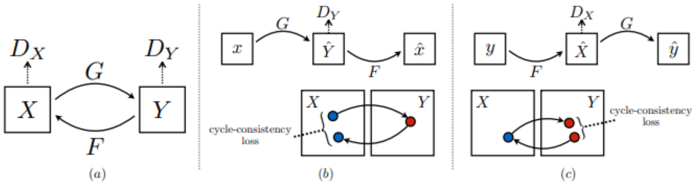
Cycle GAN **[geeksforgeeks]**

More specifically the CycleGAN architecture is different from other GANs in a way that it contains 2 mapping function (G or $G_{Y\rightarrow X}$ and F or $G_{X\rightarrow Y}$) that acts as generators and their corresponding Discriminators (D_x and D_y): The generator mapping functions are as follows:

Cycle GAN [geeksforgeeks]



Cycle GAN [geeksforgeeks]



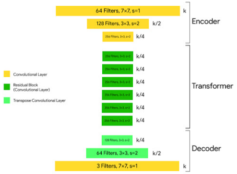
Cycle GAN [geeksforgeeks]

Each CycleGAN generator has three sections:

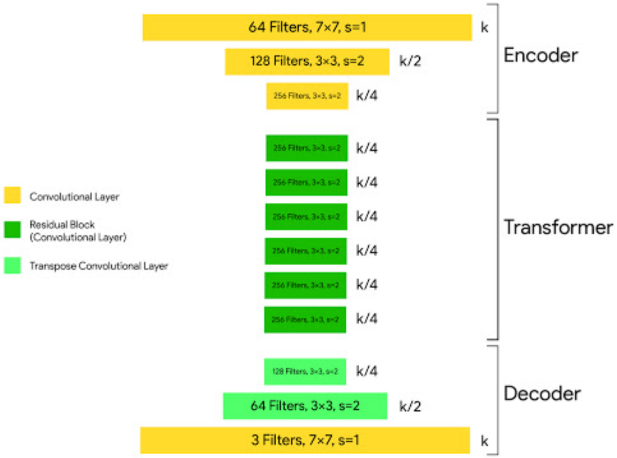
- Encoder
- Transformer
- Decoder

Cycle GAN **[geeksforgeeks]**

The input image is passed into the **encoder**. The encoder extracts features from the input image by using Convolutions and compressed the representation of image but increase the number of channels.
Then the output of encoder after activation function is applied is passed into the **transformer**. The transformer contains 6 or 9 residual blocks based on the size of input.
The output of transformer is then passed into the **decoder** which uses 2 -deconvolution block of fraction strides to increase the size of representation to original size.

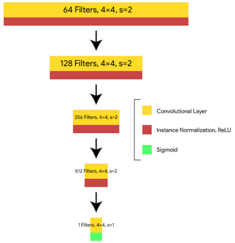


Cycle GAN [geeksforgeeks]



Cycle GAN [geeksforgeeks]

In discriminator the authors use **PatchGAN discriminator**. The difference between a PatchGAN and regular GAN discriminator is that rather the regular GAN maps from a 256×256 image to a single scalar output, which signifies 'real' or 'fake', whereas the PatchGAN maps from $256 \times 256$to an $N \times N$ (here 70×70) array of outputs X, where each X_{ij} signifies whether the patch ij in the image is real or fake.



Cycle GAN losses

Cycle GAN losses has several losses : First, the standard loss function for cGAN training is defined as follows:

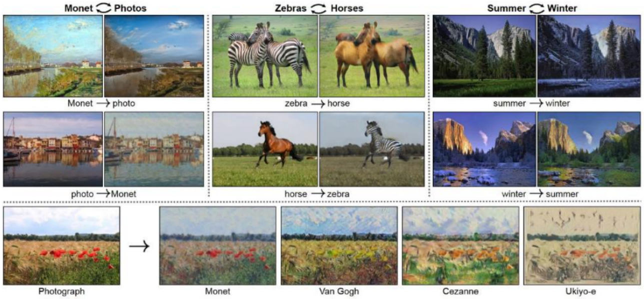
$$\mathcal{L}_{cGAN}(\theta_G, \theta_D) = \mathbb{E}_x[\log D_x(x \mid y)] + \mathbb{E}_z[\log(1 - D_x(G_{Y\to X}(z \mid y)))], \tag{1}$$

Cycle GAN losses

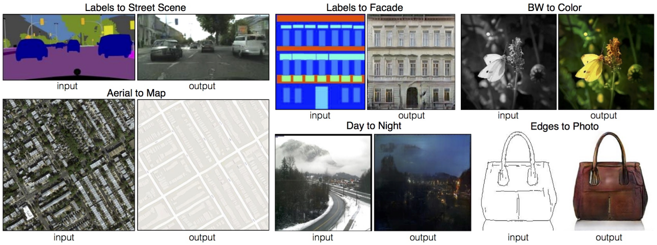
Cycle GAN losses has several losses : Secondly, in addition to the adversarial loss defined, it also have a cycle loss defined by:

$$\mathcal{L}_{L2}(\theta_G) = \mathbb{E}_{(x,y)}[\|y - G_{X\to Y}((G_{Y\to X}(y))\|_2] + \mathbb{E}_{(x,y)}[\|x - G_{Y\to X}((G_{X\to Y}(x))\|_2] \quad (2)$$

Cycada [Hoffman2018]

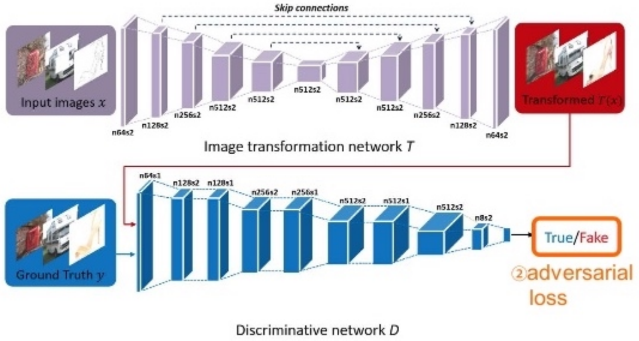


Pix2Pix [Phillip2017]



Pix2Pix [Phillip2017]

it is composed of a Conditional Generator and one discriminator. For Pix2Pix the discriminator is PatchGan and the Generator is a Unet.



Pix2Pix **[Phillip2017]**

Pix2pix has the following two losses :

$$\mathcal{L}_{cGAN}(\theta_G, \theta_D) = \mathbb{E}_x[\log D_x(x \mid y)] + \mathbb{E}_z[\log(1 - D_x(G_{Y\to X}(z \mid y)))], \tag{3}$$

In addition to the adversarial loss defined, an L1 loss, is added to the cost function of cGANs to reduce blur:

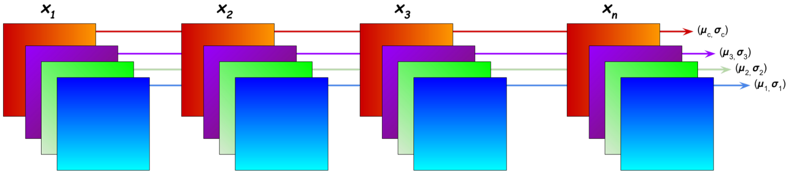
$$\mathcal{L}_{L1}(\theta_G) = \mathbb{E}_{(x,y,z)}[\|x - G(z \mid y)\|_1]. \tag{4}$$

Remember: regularization with Batch normalization

For every channel c we estimate

$$\mu_c = \frac{1}{NHW}\sum_{i=1}^{N}\sum_{j=1}^{H}\sum_{K=1}^{W} x_{icjk} \text{ and } \sigma_c = \frac{1}{NHW}\sum_{i=1}^{N}\sum_{j=1}^{H}\sum_{K=1}^{W}(x_{icjk}-\mu_c)^2 \tag{5}$$

$$\hat{x} = \frac{x-\mu_c}{\sqrt{\sigma_c^2+\epsilon}} \tag{6}$$

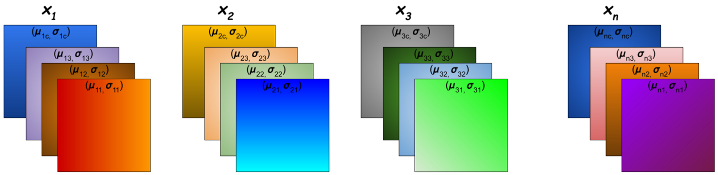


Remember: regularization Instance Normalization

For every channel c we estimate

$$\mu_{nc} = \frac{1}{HW}\sum_{j=1}^{H}\sum_{K=1}^{W} x_{ncjk} \text{ and } \sigma_{nc} = \frac{1}{HW}\sum_{j=1}^{H}\sum_{K=1}^{W}(x_{ncjk} - \mu_{nc})^2 \tag{7}$$

$$\hat{x} = \frac{x - \mu_{nc}}{\sqrt{\sigma_{nc}^2 + \epsilon}} \tag{8}$$


x_1
(μ_{1c}, σ_{1c})
(μ_{13}, σ_{13})
(μ_{12}, σ_{12})
(μ_{11}, σ_{11})
x_2
(μ_{2c}, σ_{2c})
(μ_{23}, σ_{23})
(μ_{22}, σ_{22})
(μ_{21}, σ_{21})
x_3
(μ_{3c}, σ_{3c})
(μ_{33}, σ_{33})
(μ_{32}, σ_{32})
(μ_{31}, σ_{31})
x_n
(μ_{nc}, σ_{nc})
(μ_{n3}, σ_{n3})
(μ_{n2}, σ_{n2})
(μ_{n1}, σ_{n1})

Remember: Batch normalization or Instance Normalization

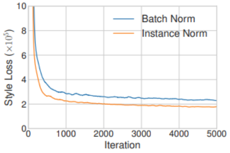
After we have assess $\hat{x}$ we to de-normalise the data

$$BN(x) = \gamma \frac{x - \mu(x)}{\sqrt{\sigma^2(x) + \epsilon}} + \beta \tag{9}$$

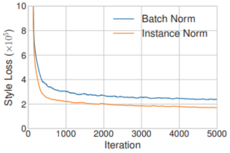
$$IN(x) = \gamma \frac{x - \mu(x)}{\sqrt{\sigma^2(x) + \epsilon}} + \beta \tag{10}$$

where γ and β are affine parameters learned during the training

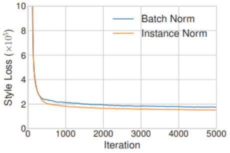
Batch normalization vs Instance Normalization



(a) Trained with original images.



(b) Trained with contrast normalized images.



(c) Trained with style normalized images.

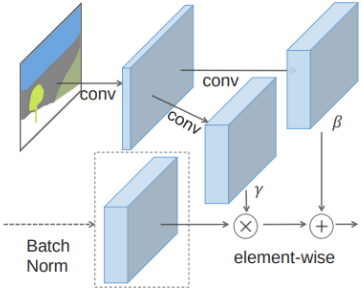
adaptive instance normalization (AdaIN) **[Huang2017]**

AdaIN receives a content input x and a style input y, and simply aligns the channelwise mean and variance of x to match those of y.

$$AdaIN(x, y) = \gamma(y)\frac{x - \mu(x)}{\sqrt{\sigma^2(x) + \epsilon}} + \beta(y) \tag{11}$$

in which we simply scale the normalized content input with $\gamma(y)$, and shift it with $\beta(y)$. Similar to IN, these statistics are computed across spatial locations.

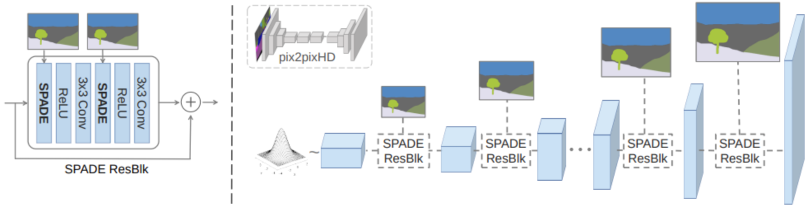
SPatially-Adaptive (DE)normalization (SPADE) [Park2019]



$$\mathit{Spade}(x, \text{label}) = \gamma(\text{label}) \frac{x - \mu(x)}{\sqrt{\sigma^2(x) + \epsilon}} + \beta(\text{label}) \tag{12}$$

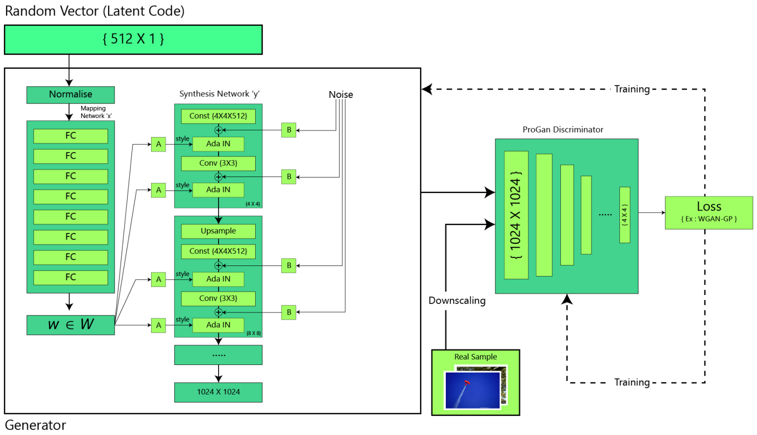
SPatially-Adaptive (DE)normalization (SPADE) [Park2019]

In the SPADE generator, each normalization layer uses the segmentation mask to modulate the layer activations.



The SPADE generator contains a series of the SPADE residual blocks with upsampling layers ant it is just a decoder.

Style GAN [geeksforgeeks]



Style GAN Novelties **[geeksforgeeks]**

Baseline Progressive Growing GANs: Style GAN uses baseline progressive GAN architecture which means the size of generated image increases gradually from a very low resolution (4×4) to high resolution (1024×1024). This is done by adding a new block to both the models to support the larger resolution after fitting the model on smaller resolution to make it more stable.

Style GAN Novelties **[geeksforgeeks]**

Mapping Network and Style Network: The goal of the mapping network is to generate the input latent vector into the intermediate vector whose different element control different visual features. Instead of directly providing latent vector to input layer the mapping is used. In this paper, the latent vector (z) of size 512 is mapped to another vector of 512 (w). The mapping function is implemented using 8-layer MLP (8-fully connected layers). The output of mapping network (w) then passed through a learned affine transformation (A) before passing into the synthesis network which AdaIN (Adaptive Instance Normalization) module. This model converts the encoded mapping into the generated image.

Style GAN Novelties **[geeksforgeeks]**

Removing traditional (Latent) input: Most previous style transfer model uses the random input to create the initial latent code of the generator i.e. the input of the 4×4 level. However the style-GAN authors concluded that the image generation features are controlled by w and AdaIN. Therefore they replace the initial input with the constant matrix of $4 \times 4 \times 512$. This also contributed to increase in the performance of the network.

Style GAN Novelties **[geeksforgeeks]**

Addition of Noisy: Input A Gaussian noise (represented by B) is added to each of these activation maps before the AdaIN operations. A different sample of noise is generated for each block and is interpreted on the basis of scaling factors of that layer.

Style GAN Novelties **[Karras2019]**

Generative Aversarial Networks: Style GAN (Karras et al., 2019)

Image size:
1024× 1024 px
(source: Karras et al.)

Bibliography

[Srivastava2017] Srivastava, A., Valkov, L., Russell, C., Gutmann, M. U.,& Sutton, C. (2017). Veegan: Reducing mode collapse in gans using implicit variational learning. Advances in neural information processing systems, 30.

[Goodfellow2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., ... & Bengio, Y. (2014). Generative adversarial nets. Advances in neural information processing systems, 27.

[Radford2015] Radford, A., Metz, L., & Chintala, S. (2015). Unsupervised representation learning with deep convolutional generative adversarial networks. arXiv preprint arXiv:1511.06434.

Bibliography

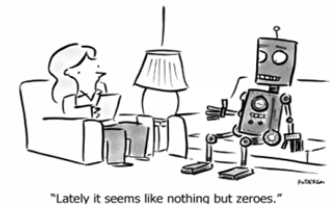
- **[Hoffman2018]** Hoffman, J., Tzeng, E., Park, T., Zhu, J. Y., Isola, P., Saenko, K., ... & Darrell, T. (2018, July). Cycada: Cycle-consistent adversarial domain adaptation. In International conference on machine learning (pp. 1989-1998). PMLR.
- **[Mirza2014]** Mirza, M., & Osindero, S. (2014). Conditional generative adversarial nets. arXiv preprint arXiv:1411.1784.
- **[jonathan-hui]** https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b
- **[geeksforgeeks]** https://www.geeksforgeeks.org/
- **[alexhwilliams]** http://alexhwilliams.info/itsneuronalblog/2020/10/09/optimal-transport/
- **[Phillip2017]** Isola, Phillip, et al. "Image-to-image translation with conditional adversarial networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

Bibliography

- **[Huang2017]** Huang, Xun, and Serge Belongie. "Arbitrary style transfer in real-time with adaptive instance normalization." Proceedings of the IEEE international conference on computer vision. 2017.
- **[Park2019]** Park, Taesung, et al. "Semantic image synthesis with spatially-adaptive normalization." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019.
- **[Karras2019]** Karras, Tero, Samuli Laine, and Timo Aila. "A style-based generator architecture for generative adversarial networks." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019.
- **[Arjovsky2017]** Arjovsky, Martin, Soumith Chintala, and LÃ©on Bottou. "Wasserstein generative adversarial networks." International conference on machine learning. PMLR, 2017.

Variational AutoEncoders (VAE)

Gianni FRANCHI
ENSTA-Paris

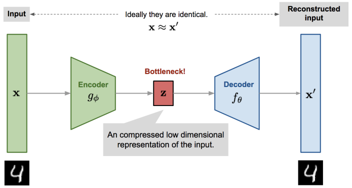


ENSTA
IP PARIS

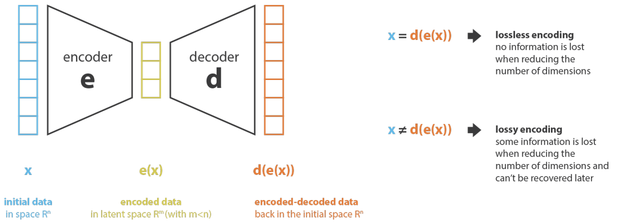
Autoencoders

Before we start talking about VAEs, let us quickly revisit autoencoders

- An autoencoder contains an encoder which takes the input X and maps it to a hidden representation
- The decoder then takes this hidden representation and tries to reconstruct the input from it.


Input
Ideally they are identical.
$\mathbf{x} \approx \mathbf{x}'$
Reconstructed input
$\mathbf{x}$
Encoder
g_ϕ
Bottleneck!
$\mathbf{z}$
Decoder
f_θ
$\mathbf{x}'$
An compressed low dimensional representation of the input.

Autoencoders [Roccatoward]


encoder
e
decoder
d
x
e(x)
d(e(x))
initial data
in space Rⁿ
encoded data
in latent space Rᵐ (with m<n)
encoded-decoded data
back in the initial space Rⁿ
x = d(e(x))
lossless encoding
no information is lost when reducing the number of dimensions
x ≠ d(e(x))
lossy encoding
some information is lost when reducing the number of dimensions and can't be recovered later

Autoencoders : Encoder

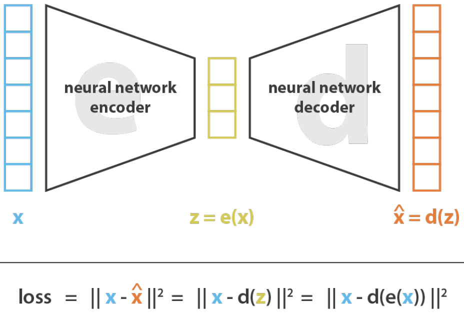
How do we choose the encoder?

- Typically we choose classical DNN architecture. Resnet **[He2016]**, Inception network **[Szegedy2016]**, VGG **[Simonyan2014]**.
- The choice depends on the application.
- But in general, we choose a pre-trained DNN.

Autoencoders : Decoder [Roccatoward]

How do we choose the Decoder?

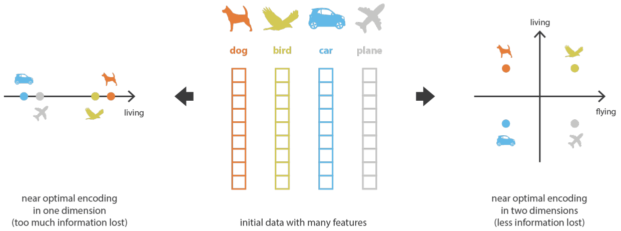
- Typically the decoder is smaller.
- Decoder might have skip connection to improve the training.
- Decoder might have multi-resolution feature maps.



Autoencoders : GOAL [Roccatoward]

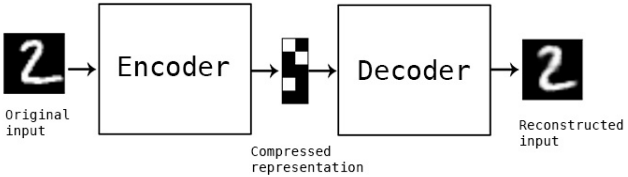
What is the goal of an autoencoder?

- Compress the data
- Find a discriminative/interesting representation
- Hide information



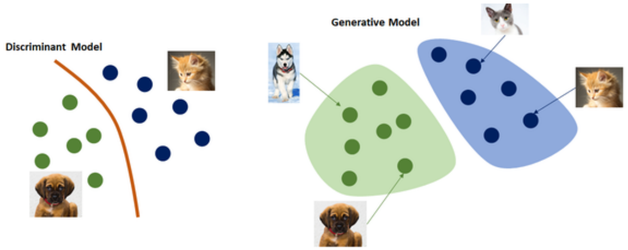
Autoencoders : Generation

Can we generate new images with an Auto encoder? **NO!** Encoder encode a data and decoder just decode.



Autoencoders : Generation

Why do we need to learn to generate? What is different?



Generation

- Generative Adversarial Network **[Goodfellow2014]**
- Variational Autoencoders **[Kingma2014]**
- Normalizing Flows **[Rezende2015]**

Variational Inference [Blei2017]

Let us consider a joint density of latent variables $z = \{z_1 : z_m\}$ and observations $x = \{x_1 : x_m\}$: $\mathcal{P}(x, z)$.
Using the Bayes' theorem we have:

$$\mathcal{P}(x, z) = \mathcal{P}(z)\mathcal{P}(x|z) \tag{1}$$

$$\mathcal{P}(z|x) = \frac{\mathcal{P}(z)\mathcal{P}(x|z)}{\mathcal{P}(x)} \tag{2}$$

- $\mathcal{P}(z)$ represents the Prior distribution;
- $\mathcal{P}(x|z)$ represents the likelihood distribution;
- $\mathcal{P}(z|x)$ represents the Posterior distribution.
- $\mathcal{P}(x)$ represents the evidence distribution.

Variational Inference [Blei2017]

Unfortunately, this integral requires exponential time to compute as it needs to be evaluated over all configurations of latent variables. We therefore need to approximate this posterior distribution. Variational inference approximates the posterior with a family of distributions $q_\lambda(z \mid x)$ parametrized by a parameter λ
The optimal approximate posterior is thus:

$$q^*_\lambda(z \mid x) = \arg\min_\lambda \mathbb{KL}(q_\lambda(z \mid x) \mid\mid p(z \mid x)) \tag{3}$$

Variational Inference [Blei2017]

The Kullback-Leibler divergence between two discrete probability distributions P and Q defined on the same probability space $\mathcal{X}$, is defined by :

$$\mathbb{KL}(Q \parallel P) = \sum_{x \in \mathcal{X}} Q(x) \log \left(\frac{Q(x)}{P(x)} \right) . \tag{4}$$

Please be careful the Kullback-Leibler divergence is not symmetric.

Variational Inference [Blei2017]

Let us develop the calculus

$$\mathbb{KL}(q_\lambda(z \mid x) \mid\mid p(z \mid x)) = \sum_z q_\lambda(z \mid x) \log\left(\frac{q_\lambda(z \mid x)}{p(z \mid x)}\right). \qquad (5)$$

$$\mathbb{KL}(q_\lambda(z \mid x) \mid\mid p(z \mid x)) = \mathrm{E}_q[\log q_\lambda(z \mid x)] - \mathrm{E}_q[\log p(z \mid x)] \quad (6)$$

using the fact that $p(z \mid x) = \frac{p(x,z)}{p(x)}$

Variational Inference [Blei2017]

$$\mathbb{KL}(q_\lambda(z \mid x) \mid\mid p(z \mid x)) = \underbrace{\mathrm{E}_q[\log q_\lambda(z \mid x)] - \mathrm{E}_q[\log p(x, z)]}_{-\mathrm{ELBO}(\lambda)} + \underbrace{\log p(x)}_{\text{log evidence}} \tag{7}$$

ELBO= Evidence Lower BOund

$$\log p(x) = \mathrm{ELBO}(\lambda) + \mathbb{KL}(q_\lambda(z \mid x) \mid\mid p(z \mid x)) \tag{8}$$

Finding the parameter λ that minimizes the Kullback-Leibler divergence is equivalent to finding λ that maximizes the ELBO. Since $\mathbb{KL}() \geq 0$, then $\log p(x) \geq \mathrm{ELBO}(\lambda)$, this explain the name of the loss.

Variational Inference **[Blei2017]**

$$\mathrm{ELBO}(\lambda) = \mathrm{E}_q[\log p(x, z)] - \mathrm{E}_q[\log q_\lambda(z \mid x)] \tag{9}$$

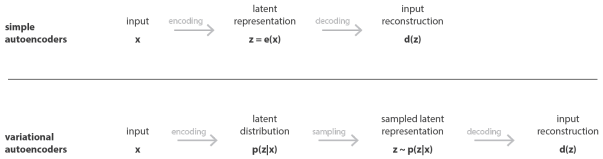
using the fact that $p(x, z) = p(x \mid z)p(z)$ we have :

$$\mathrm{ELBO}(\lambda) = \mathrm{E}_q[\log p(x \mid z)] - \mathrm{E}_q[\log q_\lambda(z \mid x)] + \mathrm{E}_q[\log p(z)] \tag{10}$$

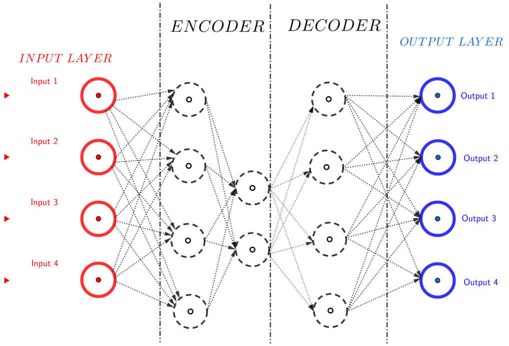
$$\mathrm{ELBO}(\lambda) = -\mathbb{KL}(q_\lambda(z \mid x) \mid\mid p(z)) + \mathrm{E}_q[\log p(x \mid z)] \tag{11}$$

We can interpret the Kullback-Leibler divergence term as a regularizer, and the expected likelihood term as a reconstruction "loss".

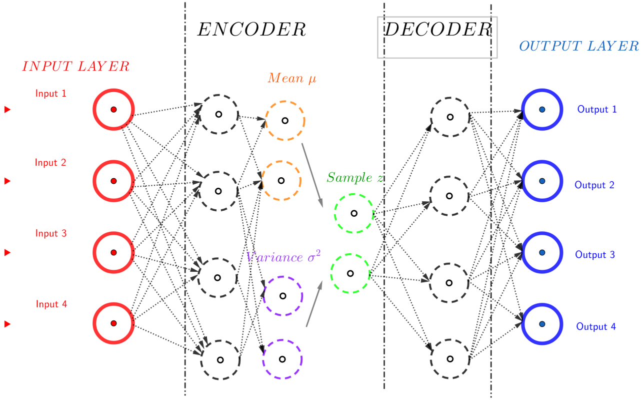
Variational Autoencoders [Kingma2019]



Autoencoders



Variational Autoencoders [Kingma2019]



Variational Autoencoders **[Kingma2019]**

The **Encoder:** is a neural network that outputs a representation z of data x. In probability model terms, the inference network parametrizes the approximate posterior of the latent variables z. The inference network outputs parameters to the distribution $q(z \mid x)$.
The **Decoder** is a neural net that learns to reconstruct the data x given a representation z. In terms of probability models, the likelihood of the data x given latent variables z is parametrized by a generative network. The generative network outputs parameters to the likelihood distribution $p(x \mid z)$.

Reparametreziation trick **[Kingma2019]**

When we want to optimize a deep learning problem we often have something to optimize of the following form :

$$\mathcal{L}(\theta) = \mathbb{E}_{p(z)}[f_\theta(z)]$$

Let us say that we want to take the gradient $\mathcal{L}(\theta)$ with respect to θ.
We consider that p is a density and $f_\theta(z)$ is differentiate.
We can compute the gradient as the mean of gradients.

Reparametreziation trick **[Kingma2019]**

$$\nabla_\theta \mathcal{L}(\theta) = \nabla_\theta \mathbb{E}_{p(z)}[f_\theta(z)]$$

$$\nabla_\theta \mathcal{L}(\theta) = \nabla_\theta \int [f_\theta(z)] p(z) dz$$

$$\nabla_\theta \mathcal{L}(\theta) = \int \nabla_\theta [f_\theta(z)] p(z) dz$$

$$\nabla_\theta \mathcal{L}(\theta) = \mathbb{E}_{p(z)}[\nabla_\theta f_\theta(z)]$$

But what happens if our density p is also parameterized by θ?

Reparametreziation trick **[Kingma2019]**

$$\nabla_\theta \mathcal{L}(\theta) = \nabla_\theta \mathbb{E}_{p_\theta(z)}[f_\theta(z)]$$

$$\nabla_\theta \mathcal{L}(\theta) = \nabla_\theta \int f_\theta(z) p_\theta(z) dz$$

$$\nabla_\theta \mathcal{L}(\theta) = \int \nabla_\theta [f_\theta(z) p_\theta(z)] dz$$

$$\nabla_\theta \mathcal{L}(\theta) = \int f_\theta(z) \nabla_\theta [p_\theta(z)] dz + \mathbb{E}_{p_\theta(z)}[\nabla_\theta f_\theta(z)]$$

The first term of the last equation is not guaranteed to be an expectation.

Reparametreziation trick **[Kingma2019]**

The Reparametreziation trick consist in finding a random variable ϵ that follow a distribution $q(\epsilon)$ and a function t such that :

$$z = t(\theta, \epsilon) \sim p_\theta$$

We must have $q(\epsilon)d\epsilon = p_\theta(z)dz$

Reparametreziation trick **[Kingma2019]**

Then we have

$$\nabla_\theta \mathcal{L}(\theta) = \nabla_\theta \int f_\theta(z) p_\theta(z) dz$$

$$\nabla_\theta \mathcal{L}(\theta) = \nabla_\theta \int f_\theta(z) q_\epsilon(\epsilon) d\epsilon$$

$$\nabla_\theta \mathcal{L}(\theta) = \int \nabla_\theta [f_\theta(z)] q_\epsilon(\epsilon) d\epsilon$$

Please note that $f_\theta(z)$ is function of two variables hence: $f_\theta(z) = f(z, \theta)$. In addition $z = t(\theta, \epsilon)$ so we have $f_\theta(z) = f(t(\theta, \epsilon), \theta)$.

Reparametreziation trick [Kingma2019]

Let us write that $f = h \circ g$

- with $h : \mathbb{R}^2 \to \mathbb{R}$ a differentiable function $(h : (x_1, x_2) \to f(x_1, x_2))$;
- with $g : \mathbb{R}^2 \to \mathbb{R}^2$ a differentiable function $(g : (\epsilon, \theta) \to (t(\theta, \epsilon), \theta))$
- with $f : \mathbb{R}^2 \to \mathbb{R}$ a differentiable function;

Hence, the chain rule results is:

$$\frac{\partial f}{\partial \theta} = \sum_{k=1}^{2} \frac{\partial h}{\partial g_k} \underbrace{\frac{\partial g_k}{\partial \theta}}_{\text{recursive case}}$$

so we have:

$$\frac{\partial f_\theta(z)}{\partial \theta} = \frac{\partial f_\theta(z)}{\partial z} \frac{\partial z}{\partial \theta} + \frac{\partial f_\theta(z)}{\partial \theta}$$

Reparametreziation trick **[Kingma2019]**

$$\nabla_\theta \mathcal{L}(\theta) = \nabla_\theta \int f_\theta(z)]p_\theta(z)dz$$

$$\nabla_\theta \mathcal{L}(\theta) = \nabla_\theta \int \nabla_\theta[f_\theta(z)]q_\epsilon(\epsilon)d\epsilon$$

$$\nabla_\theta \mathcal{L}(\theta) = \int \frac{\partial f_\theta(z)}{\partial z}\frac{\partial z}{\partial \theta} + \frac{\partial f_\theta(z)}{\partial \theta} q_\epsilon(\epsilon)d\epsilon$$

Reparametreziation trick [Kingma2019]

- the encoder outputs the mean and standard deviation $q_\phi(x) = (\mu(x), \sigma(x))$
- we generate a random noise $\epsilon \sim \mathcal{N}(0,1)$
- we perform the reparametrization $z = \mu(x) + \sigma(x) \times \epsilon$
- the decoder generate the image $p_\theta(x \mid z) = \hat{x}$

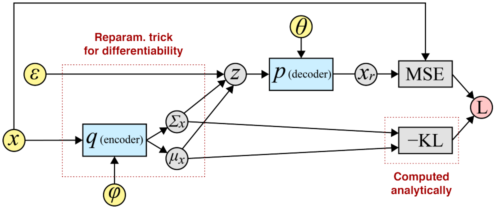


Figure: Reparametreziation trick[1]

[1] https://gregorygundersen.com/blog/2018/04/29/reparameterization/

Results VAE on CIFAR **[Kingma2019]**

BETA VAE **[Higgins2016]**

They have been researches on **disentangling** the latent representation.
What does it mean?
If each variable in the inferred latent representation z is only sensitive to one single generative factor and relatively invariant to other factors, we will say this representation is disentangled or factorized.
Why do we want that?
One benefit that often comes with disentangled representation is good interpretability and straightforward generalization to various tasks.

BETA VAE **[Higgins2016]**

$\beta-$VAE **[Higgins2016]** is a modification of Variational Autoencoder with a special emphasis to discover **disentangled** latent space. The loss is

$$\arg\max_{\lambda} = \mathrm{E}_{q_\lambda}[\log p_\theta(x \mid z)] \tag{12}$$

$$\text{subject to } \mathbb{KL}(q_\lambda(z \mid x) \mid\mid p(z)) < \delta \tag{13}$$

They want the distance between the real and estimated posterior distributions to be small.

BETA VAE **[Higgins2016]**

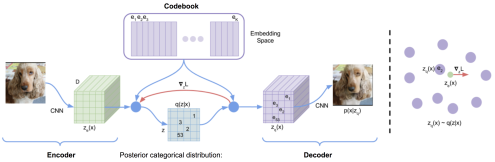
Using the Theorem of Lagrangian multiplier the loss become:

$$\mathcal{L}(\lambda, \theta) = \mathrm{E}_{q_\lambda}[\log p(x \mid z)] - \beta \mathbb{KL}(q_\lambda(z \mid x) \mid\mid p(z)) \tag{14}$$

When $\beta = 1$, it is same as VAE. When $\beta > 1$, it applies a stronger constraint on the latent bottleneck and limits the representation capacity of z.
A higher β encourages more efficient latent encoding and further encourages the disentanglement.

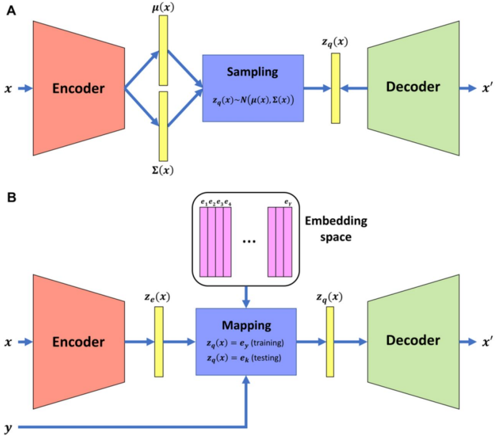
VQ-VAE [Oord2017]

The VQ-VAE ("Vector Quantised-Variational AutoEncoder"; model learns a discrete latent variable by the encoder, since discrete representations may be a more natural fit for problems like language, speech, reasoning, etc.

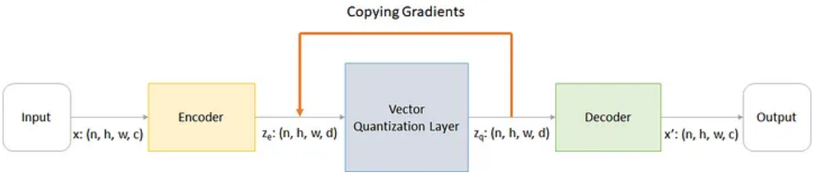


$$q(\mathbf{z} = \mathbf{e}_k|\mathbf{x}) = \begin{cases} 1 & \text{if } k = \arg\min_i \|\mathbf{z}_e(\mathbf{x}) - \mathbf{e}_i\|_2 \\ 0 & \text{otherwise.} \end{cases}$$

VQ-VAE [Oord2017] [Yifan2020]



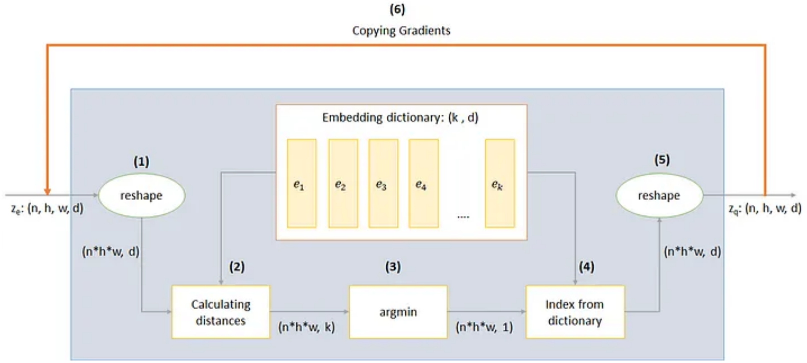
VQ-VAE [web1]


Copying Gradients
Input
x: (n, h, w, c)
Encoder
z_e: (n, h, w, d)
Vector Quantization Layer
z_q: (n, h, w, d)
Decoder
x': (n, h, w, c)
Output

VQ-VAE

- Encoder takes in images x: (n, h, w, c) and give outputs z_e: (n, h, w, d)
- Vector Quantization layer takes z_e and selects embeddings from a dictionary based on distance and outputs z_q
- Decoder consumes z_q and outputs x' trying to recreate input x

VQ-VAE [web1]


(6)
Copying Gradients
Embedding dictionary: (k , d)
e_1
e_2
e_3
e_4
....
e_k
(1)
reshape
z_e: (n, h, w, d)
(n*h*w, d)
(2)
Calculating distances
(n*h*w, k)
(3)
argmin
(n*h*w, 1)
(4)
Index from dictionary
(5)
reshape
(n*h*w, d)
z_q: (n, h, w, d)

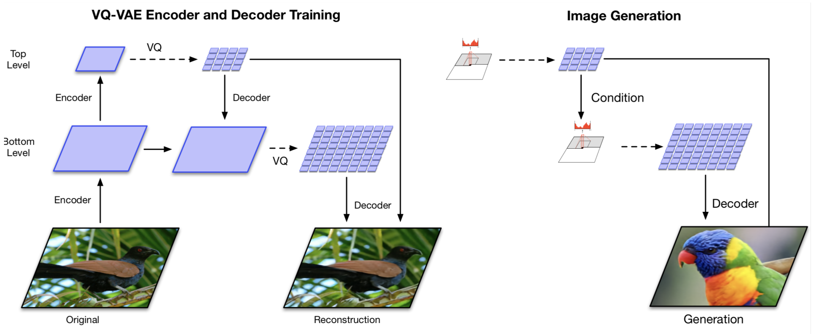
VQ-VAE

The total loss is actually composed of three components:

- Reconstruction loss: reconstruction-loss $= -log(p(x|z_q))$
- Codebook loss codebook-loss $= \|sg[z_e(x)] - e\|^2$ where sg represents stop gradient operator meaning no gradient
- Commitment (affectation of the code) loss commitment-loss $= \beta\|z_e(x) - sg[e]\|^2$ β is a hyperparameter that controls how much we want to weigh commitment loss compared to other components

Results VQ-VAE **[Oord2017]**

VQ-VAE [Razavi2019]



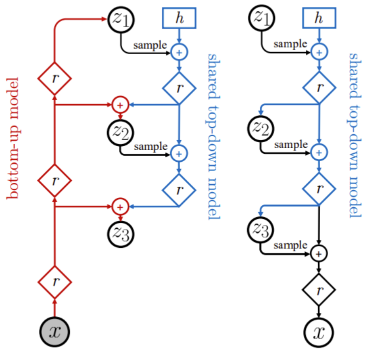
VQ-VAE [Razavi2019]

deep hierarchical VAEs **[Vahdat2020]**

The latent variables are partitioned into disjoint groups $\boldsymbol{z} = \{z_1, \ldots, z_l\}$.
The prior is represented by $p(\boldsymbol{z}) = \prod_l p(z_l|z_{<l})$ the approximate posterior is given by $q(\boldsymbol{z}|z_{<l}, x) = \prod_l p(z_l|z_{<l})$ The ELBO is equal to :

$$\mathcal{L}_{\text{VAE}} = \mathrm{E}_{q(\boldsymbol{z}|z_{<l},x)}[\log p(x \mid z)] - \mathbb{KL}(q(z_1|x) \mid\mid p(z_1)) - \sum_l \mathrm{E}_{q(z_{<l}|x)}[\mathbb{KL}(q(z_l|z_{<l}, x)) \mid\mid p(z_l|z_{<l}))]$$

VQ-VAE [Vahdat2020]



(a) Bidirectional Encoder (b) Generative Model

Figure: The neural networks implementing an encoder q(z|x) and generative model p(x, z) for a 3-group hierarchical VAE. r denotes residual neural networks, + denotes feature combination, and h a trainable parameter.

Bibliography

[Blei2017] Blei, David M., Alp Kucukelbir, and Jon D. McAuliffe. "Variational inference: A review for statisticians." Journal of the American statistical Association 112.518 (2017): 859-877.

[Kingma2019] Kingma, Diederik P., and Max Welling. "An introduction to variational autoencoders." arXiv preprint arXiv:1906.02691 (2019).

[Kingma2014] Kingma, Diederik P., and Max Welling. "Auto-encoding variational bayes." arXiv preprint arXiv:1312.6114 (2013).

[Roccatoward] Joseph Rocca. "`https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73`"

Bibliography

[Higgins2016] Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., ... & Lerchner, A. (2016). beta-vae: Learning basic visual concepts with a constrained variational framework.

[Oord2017] Oord, Aaron van den, Oriol Vinyals, and Koray Kavukcuoglu. "Neural discrete representation learning." arXiv preprint arXiv:1711.00937 (2017).

[Razavi2019] Razavi, Ali, Aaron van den Oord, and Oriol Vinyals. "Generating diverse high-fidelity images with vq-vae-2." arXiv preprint arXiv:1906.00446 (2019).

Bibliography

- **[Yifan2020]** Xue, Yifan, Michael Q. Ding, and Xinghua Lu. "Learning to encode cellular responses to systematic perturbations with deep generative models." NPJ systems biology and applications 6.1 (2020): 1-11.
- **[He2016]** He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
- **[Simonyan2014]** Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).
- **[Szegedy2016]** Szegedy, Christian, et al. "Rethinking the inception architecture for computer vision." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

Bibliography

[Goodfellow2014] Goodfellow, Ian J., et al. "Generative adversarial networks." arXiv preprint arXiv:1406.2661 (2014).

[Rezende2015] Rezende, Danilo, and Shakir Mohamed. "Variational inference with normalizing flows." International Conference on Machine Learning. PMLR, 2015.

[web1] https://shashank7-iitd.medium.com/understanding-vector-quantized-variational-autoencoders-vq-vae-323d710a888a

[Vahdat2020] Vahdat, Arash, and Jan Kautz. "NVAE: A deep hierarchical variational autoencoder." Advances in neural information processing systems 33 (2020): 19667-19679.

Diffusion models

Gianni FRANCHI
ENSTA-Paris

"Lately it seems like nothing but zeroes."

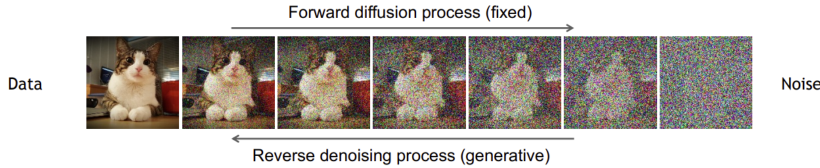
Diffusion Model [CVPR TUTO]

Denoising Diffusion Models

Learning to generate by denoising

Denoising diffusion models consist of two processes:

- Forward diffusion process that gradually adds noise to input
- Reverse denoising process that learns to generate data by denoising

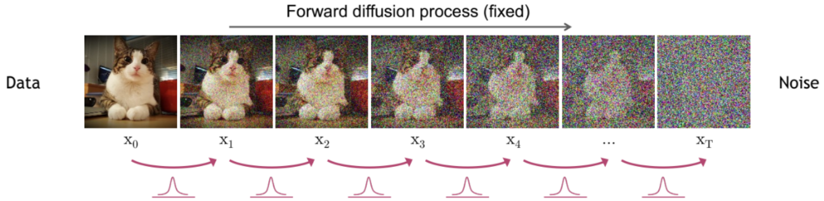

Forward diffusion process (fixed)
Data
Noise
Reverse denoising process (generative)

Sohl-Dickstein et al., Deep Unsupervised Learning using Nonequilibrium Thermodynamics, ICML 2015
Ho et al., Denoising Diffusion Probabilistic Models, NeurIPS 2020
Song et al., Score-Based Generative Modeling through Stochastic Differential Equations, ICLR 2021

Diffusion Model [CVPR TUTO]

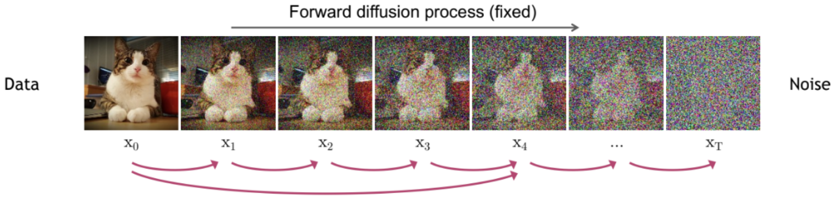
Forward Diffusion Process

The formal definition of the forward process in T steps:



$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x_t}; \sqrt{1-\beta_t}\mathbf{x_{t-1}}, \beta_t\mathbf{I}) \Rightarrow q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}) \quad \text{(joint)}$$

Diffusion Kernel



Define $\bar{\alpha}_t = \prod_{s=1}^{t}(1-\beta_s)$ $\Rightarrow$ $q(\mathbf{x}_t|\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_t; \sqrt{\bar{\alpha}_t}\mathbf{x}_0, (1-\bar{\alpha}_t)\mathbf{I}))$ (Diffusion Kernel)

For sampling: $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\ \epsilon$ where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

β_t values schedule (i.e., the noise schedule) is designed such that $\bar{\alpha}_T \rightarrow 0$ and $q(\mathbf{x}_T|\mathbf{x}_0) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$

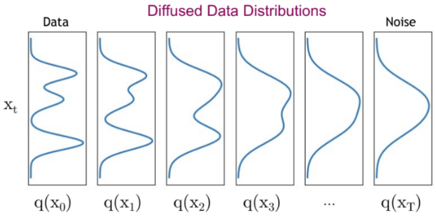
Diffusion Model [CVPR TUTO]

What happens to a distribution in the forward diffusion?

So far, we discussed the diffusion kernel $q(\mathbf{x}_t|\mathbf{x}_0)$ but what about $q(\mathbf{x}_t)$?

$$\underbrace{q(\mathbf{x}_t)}_{\text{Diffused data dist.}} = \int \underbrace{q(\mathbf{x}_0, \mathbf{x}_t)}_{\text{Joint dist.}}\, d\mathbf{x}_0 = \int \underbrace{q(\mathbf{x}_0)}_{\text{Input data dist.}}\, \underbrace{q(\mathbf{x}_t|\mathbf{x}_0)}_{\text{Diffusion kernel}}\, d\mathbf{x}_0$$

The diffusion kernel is Gaussian convolution.



We can sample $\mathbf{x}_t \sim q(\mathbf{x}_t)$ by first sampling $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ and then sampling $\mathbf{x}_t \sim q(\mathbf{x}_t|\mathbf{x}_0)$ (i.e., ancestral sampling).

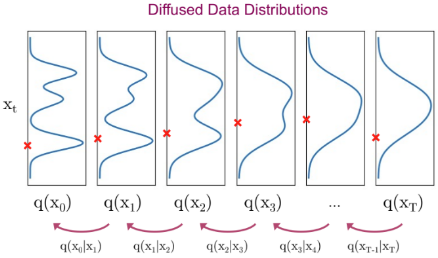
Generative Learning by Denoising

Recall, that the diffusion parameters are designed such that $q(\mathbf{x}_T) \approx \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I}))$

Generation:

Sample $\mathbf{x}_T \sim \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$

Iteratively sample $\mathbf{x}_{t-1} \sim \underbrace{q(\mathbf{x}_{t-1}|\mathbf{x}_t)}_{\text{True Denoising Dist.}}$

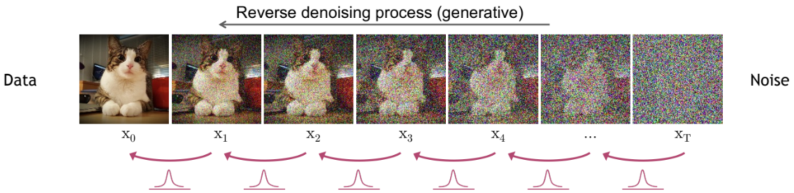


In general, $q(\mathbf{x}_{t-1}|\mathbf{x}_t) \propto q(\mathbf{x}_{t-1})q(\mathbf{x}_t|\mathbf{x}_{t-1})$ is intractable.

Can we approximate $q(\mathbf{x}_{t-1}|\mathbf{x}_t)$? Yes, we can use a Normal distribution if β_t is small in each forward diffusion step.

Reverse Denoising Process

Formal definition of forward and reverse processes in T steps:

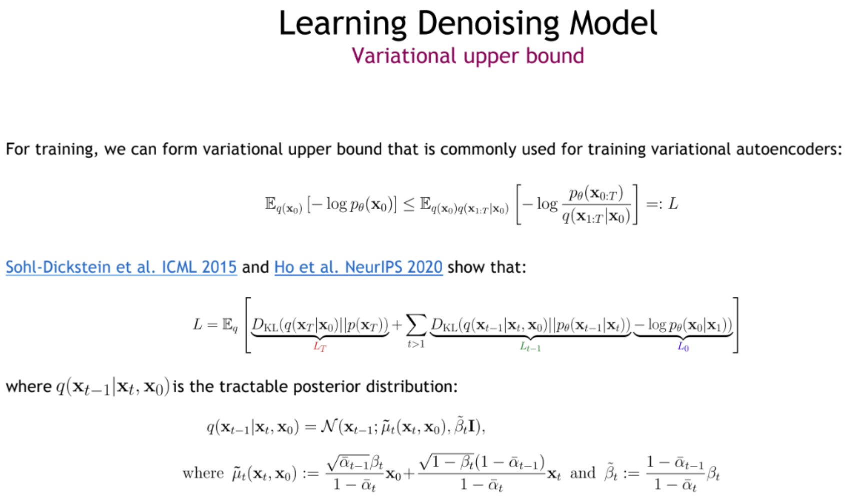


$$p(\mathbf{x}_T) = \mathcal{N}(\mathbf{x}_T; \mathbf{0}, \mathbf{I})$$
$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \underbrace{\mu_\theta(\mathbf{x}_t, t)}_{\substack{\text{Trainable network} \\ \text{(U-net, Denoising Autoencoder)}}}, \sigma_t^2\mathbf{I})$$

$\Rightarrow$ $$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T)\prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$$

23

Diffusion Model [CVPR TUTO]



Learning Denoising Model

Variational upper bound

For training, we can form variational upper bound that is commonly used for training variational autoencoders:

$$\mathbb{E}_{q(\mathbf{x}_0)}\left[-\log p_\theta(\mathbf{x}_0)\right] \leq \mathbb{E}_{q(\mathbf{x}_0)q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\left[-\log \frac{p_\theta(\mathbf{x}_{0:T})}{q(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] =: L$$

Sohl-Dickstein et al. ICML 2015 and Ho et al. NeurIPS 2020 show that:

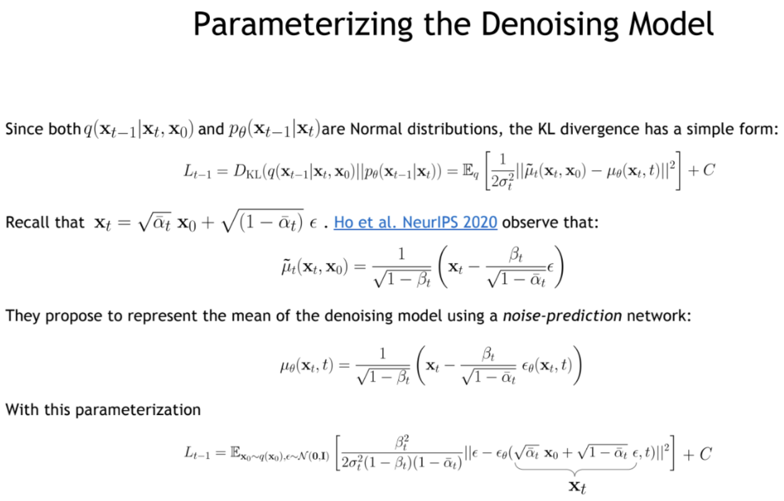
$$L = \mathbb{E}_q\left[\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_T|\mathbf{x}_0)||p(\mathbf{x}_T))}_{L_T} + \sum_{t>1}\underbrace{D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t))}_{L_{t-1}}\underbrace{-\log p_\theta(\mathbf{x}_0|\mathbf{x}_1))}_{L_0}\right]$$

where $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ is the tractable posterior distribution:

$$q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0) = \mathcal{N}(\mathbf{x}_{t-1};\tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0),\tilde{\beta}_t\mathbf{I}),$$

$$\text{where}\quad \tilde{\mu}_t(\mathbf{x}_t,\mathbf{x}_0) := \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0 + \frac{\sqrt{1-\beta_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t \quad\text{and}\quad \tilde{\beta}_t := \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t$$

Diffusion Model [CVPR TUTO]



Parameterizing the Denoising Model

Since both $q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)$ and $p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$ are Normal distributions, the KL divergence has a simple form:

$$L_{t-1} = D_{\mathrm{KL}}(q(\mathbf{x}_{t-1}|\mathbf{x}_t, \mathbf{x}_0)||p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)) = \mathbb{E}_q \left[\frac{1}{2\sigma_t^2}||\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) - \mu_\theta(\mathbf{x}_t, t)||^2\right] + C$$

Recall that $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\ \epsilon$. Ho et al. NeurIPS 2020 observe that:

$$\tilde{\mu}_t(\mathbf{x}_t, \mathbf{x}_0) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon\right)$$

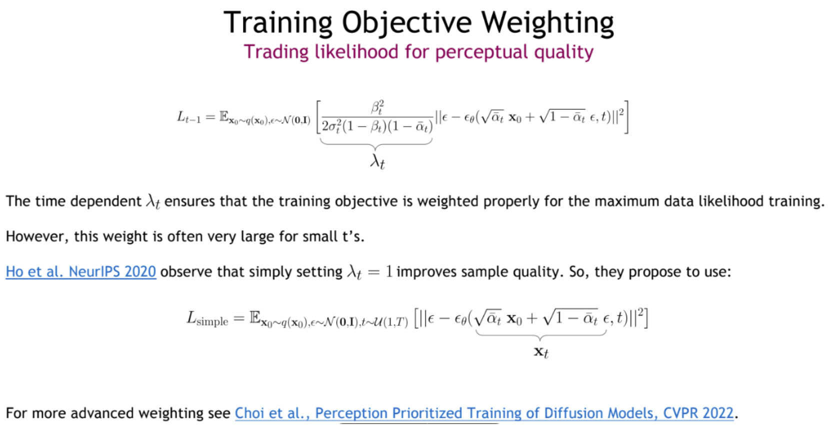
They propose to represent the mean of the denoising model using a *noise-prediction* network:

$$\mu_\theta(\mathbf{x}_t, t) = \frac{1}{\sqrt{1-\beta_t}}\left(\mathbf{x}_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\ \epsilon_\theta(\mathbf{x}_t, t)\right)$$

With this parameterization

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})}\left[\frac{\beta_t^2}{2\sigma_t^2(1-\beta_t)(1-\bar{\alpha}_t)}||\epsilon - \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\ \epsilon}_{\mathbf{x}_t}, t)||^2\right] + C$$

Diffusion Model [CVPR TUTO]



Training Objective Weighting

Trading likelihood for perceptual quality

$$L_{t-1} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})} \left[\underbrace{\frac{\beta_t^2}{2\sigma_t^2 (1-\beta_t)(1-\bar{\alpha}_t)}}_{\lambda_t} ||\epsilon - \epsilon_\theta(\sqrt{\bar{\alpha}_t}\, \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon, t)||^2 \right]$$

The time dependent λ_t ensures that the training objective is weighted properly for the maximum data likelihood training.

However, this weight is often very large for small t's.

Ho et al. NeurIPS 2020 observe that simply setting $\lambda_t = 1$ improves sample quality. So, they propose to use:

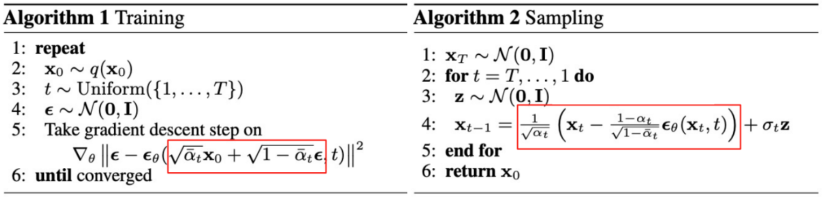
$$L_{\text{simple}} = \mathbb{E}_{\mathbf{x}_0 \sim q(\mathbf{x}_0), \epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I}), t \sim \mathcal{U}(1,T)} \left[||\epsilon - \epsilon_\theta(\underbrace{\sqrt{\bar{\alpha}_t}\, \mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\, \epsilon}_{\mathbf{x}_t}, t)||^2 \right]$$

For more advanced weighting see Choi et al., Perception Prioritized Training of Diffusion Models, CVPR 2022.

Diffusion Model [CVPR TUTO]

Summary

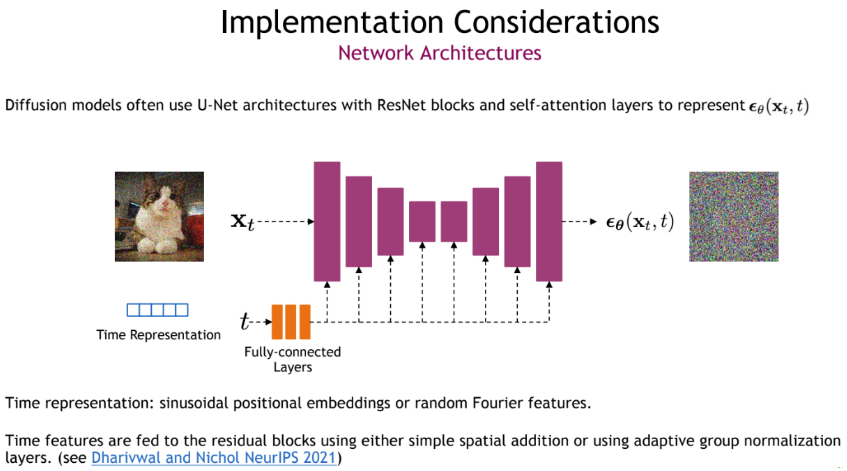
Training and Sample Generation



```
Algorithm 1 Training
1: repeat
2:   $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3:   $t \sim \mathrm{Uniform}(\{1, \dots, T\})$
4:   $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5:   Take gradient descent step on
         $\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\boxed{\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}}, t) \right\|^2$
6: until converged
```

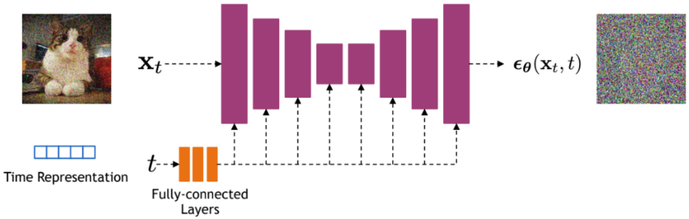
```
Algorithm 2 Sampling
1: $\mathbf{x}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
2: for $t = T, \dots, 1$ do
3:   $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
4:   $\mathbf{x}_{t-1} = \boxed{\frac{1}{\sqrt{\alpha_t}}\left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t)\right)} + \sigma_t \mathbf{z}$
5: end for
6: return $\mathbf{x}_0$
```

Diffusion Model [CVPR TUTO]



Implementation Considerations
Network Architectures

Diffusion models often use U-Net architectures with ResNet blocks and self-attention layers to represent $\epsilon_\theta(\mathbf{x}_t, t)$



Time representation: sinusoidal positional embeddings or random Fourier features.

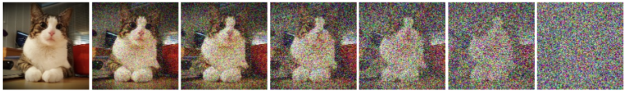
Time features are fed to the residual blocks using either simple spatial addition or using adaptive group normalization layers. (see Dharivwal and Nichol NeurIPS 2021)

Diffusion Model [CVPR TUTO]

Diffusion Parameters

Noise Schedule

$$q(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t\mathbf{I})$$



$$p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \sigma_t^2\mathbf{I})$$

Above, β_t and σ_t^2 control the variance of the forward diffusion and reverse denoising processes respectively.

Often a linear schedule is used for β_t, and σ_t^2 is set equal to β_t.

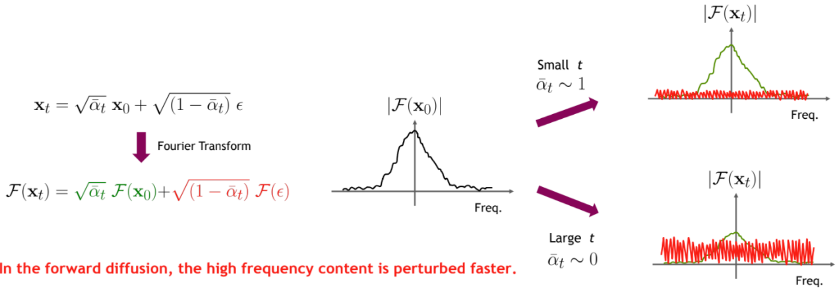
Kingma et al. NeurIPS 2022 introduce a new parameterization of diffusion models using signal-to-noise ratio (SNR), and show how to learn the noise schedule by minimizing the variance of the training objective.

We can also train σ_t^2 while training the diffusion model by minimizing the variational bound (Improved DPM by Nichol and Dhariwal ICML 2021) or after training the diffusion model (Analytic-DPM by Bao et al. ICLR 2022).

Diffusion Model [CVPR TUTO]

What happens to an image in the forward diffusion process?

Recall that sampling from $q(\mathbf{x}_t|\mathbf{x}_0)$ is done using $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\ \epsilon$ where $\epsilon \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$

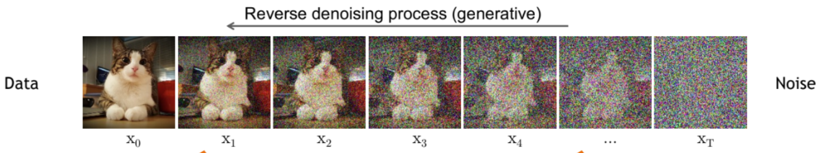


$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\ \mathbf{x}_0 + \sqrt{(1-\bar{\alpha}_t)}\ \epsilon$
Fourier Transform
$\mathcal{F}(\mathbf{x}_t) = \sqrt{\bar{\alpha}_t}\ \mathcal{F}(\mathbf{x}_0) + \sqrt{(1-\bar{\alpha}_t)}\ \mathcal{F}(\epsilon)$
$|\mathcal{F}(\mathbf{x}_0)|$
Freq.
Small t
$\bar{\alpha}_t \sim 1$
$|\mathcal{F}(\mathbf{x}_t)|$
Freq.
Large t
$\bar{\alpha}_t \sim 0$
$|\mathcal{F}(\mathbf{x}_t)|$
Freq.

In the forward diffusion, the high frequency content is perturbed faster.

Diffusion Model [CVPR TUTO]

Content-Detail Tradeoff



The denoising model is specialized for generating the high-frequency content (i.e., low-level details)

The denoising model is specialized for generating the low-frequency content (i.e., coarse content)

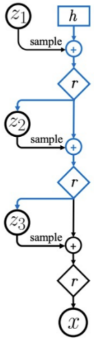
The weighting of the training objective for different timesteps is important!

Connection to VAEs

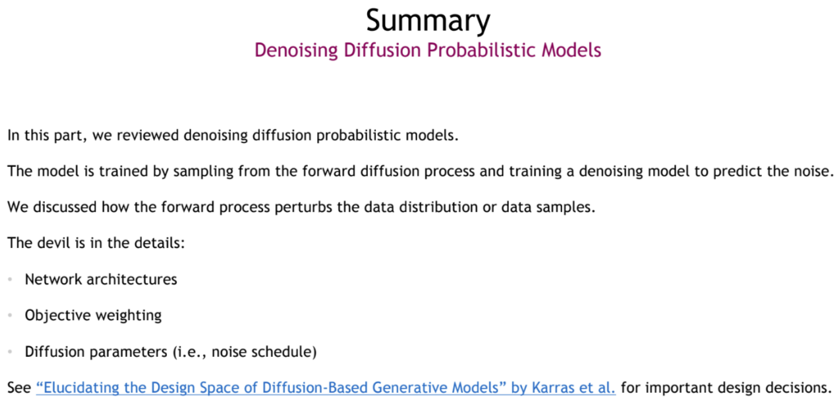
Diffusion models can be considered as a special form of hierarchical VAEs.

However, in diffusion models:

- The encoder is fixed
- The latent variables have the same dimension as the data
- The denoising model is shared across different timestep
- The model is trained with some reweighting of the variational bound.



Diffusion Model [CVPR TUTO]



Summary

Denoising Diffusion Probabilistic Models

In this part, we reviewed denoising diffusion probabilistic models.

The model is trained by sampling from the forward diffusion process and training a denoising model to predict the noise.

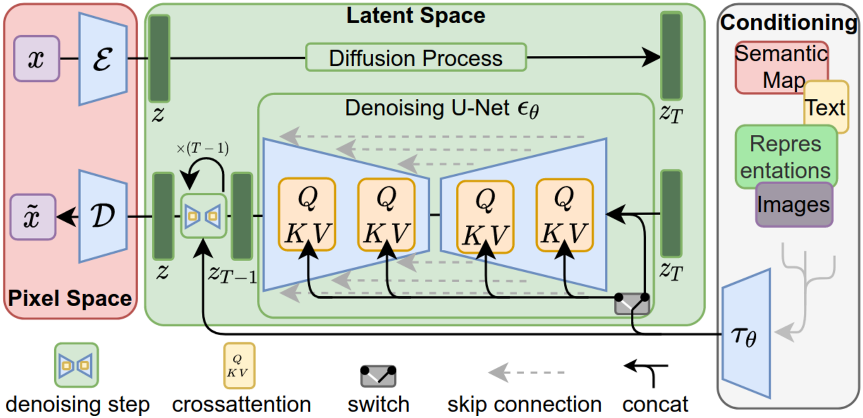
We discussed how the forward process perturbs the data distribution or data samples.

The devil is in the details:

- Network architectures
- Objective weighting
- Diffusion parameters (i.e., noise schedule)

See "Elucidating the Design Space of Diffusion-Based Generative Models" by Karras et al. for important design decisions.

Stable Diffusion [Rombach2022]



Stable Diffusion [Rombach2022]

- **Approach:** The key idea is to perform diffusion steps in the latent space of an autoencoder instead of in the pixel space of images. This reduces the dimensionality of input tensors to the U-Net compared to the original images.
- **Benefits:**
  - Accelerates training/finetuning of the U-Net, reducing GPU RAM requirements during training. Shortens sampling time independently of the scheduler, as each pass through the U-Net becomes faster.
  - Additional Enhancement: A new multimodal conditioning mechanism through cross-attention, providing an improved Stable Diffusion model.

Bibliography

[CVPR TUTO] Karsten Kreis, Ruiqi Gao, and Arash Vahdat
Denoising Diffusion-based Generative Modeling: Foundations and Applications, CVPR Tutorial

[Rombach2022] Rombach, R., Blattmann, A., Lorenz, D., Esser, P., & Ommer, B. (2022). High-resolution image synthesis with latent diffusion models. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 10684-10695).