Advanced Deep Learning Architectures
COMP 5214 & ELEC 5680

Instructor: Dr. Qifeng Chen
https://cqf.io

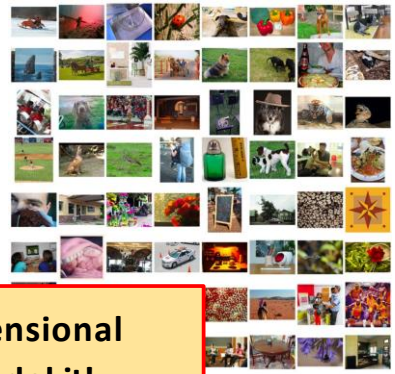
Slides by Santiago Pascual de la Puente

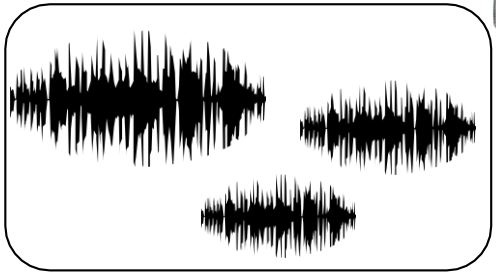
What is a generative model?

We have datapoints that emerge from some generating process (landscapes in the nature, speech from people, etc.)

$\mathbf{X} = \{x_1, x_2, \ldots, x_N\}$

Each point x_n comes from an M-dimensional probability distribution $P(X)$ → Model it!

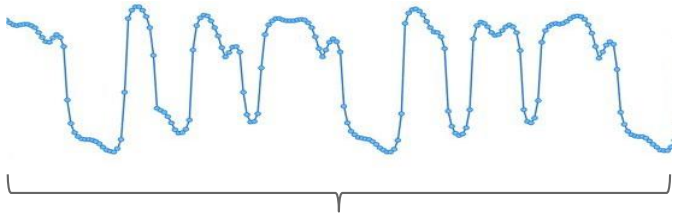


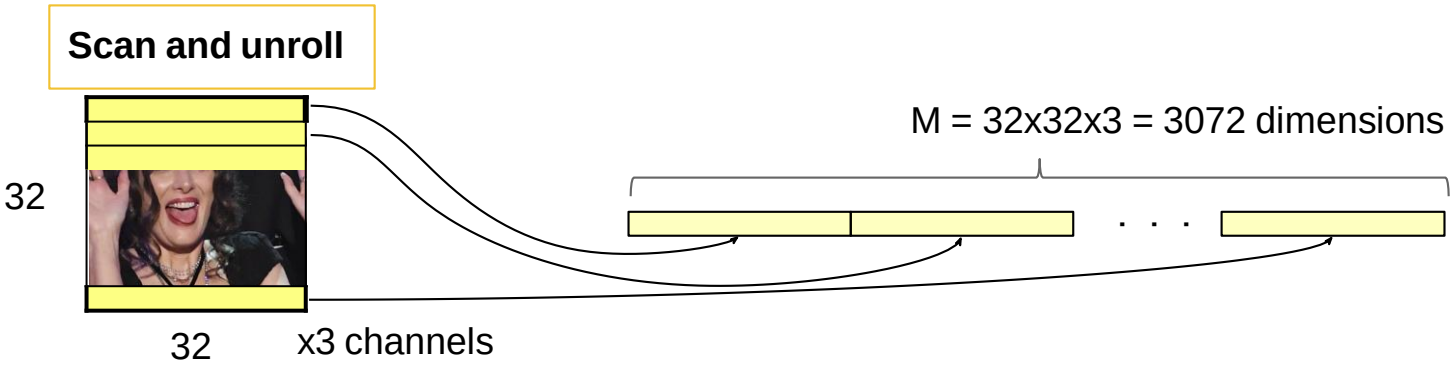


Having our dataset **X** with example datapoints (images, waveforms, written text, etc.) each point $\mathbf{x}_n$ lays in some M-dimensional space.

What is a generative model?

We can generate any type of data, like speech waveform samples or image pixels.





What is a generative model?

Our learned model should be able to make up new samples from the distribution, not just copy and paste existing samples!

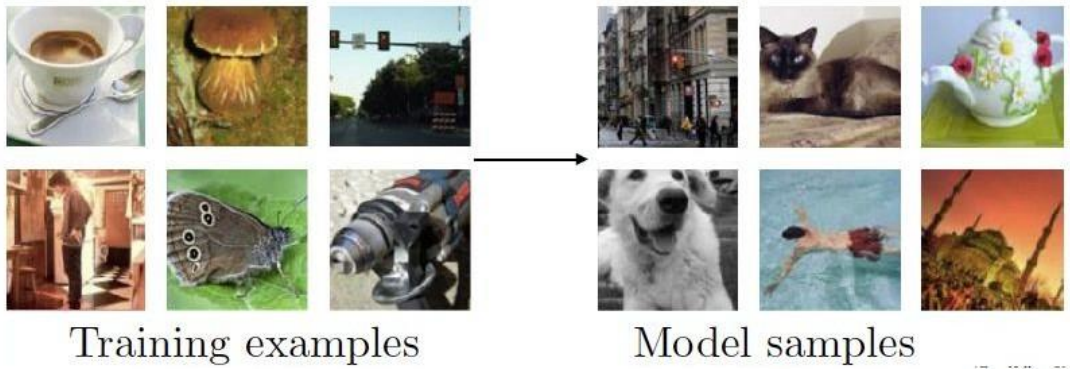


Figure from NIPS 2016 Tutorial: Generative Adversarial Networks (I. Goodfellow)

Taxonomy

We will see models that learn the probability density function:

- Explicitly (we impose a known loss function)
  - (1) With approximate density → Variational Auto-Encoders (VAEs)
  - (2) With tractable density & likelihood-based:
    - PixelCNN, Wavenet.
    - Flow-based models: real-NVP, GLOW.
- Implicitly (we “do not know” the loss function)
  - (3) Generative Adversarial Networks (GANs).

Likelihood-based Models

PixelRNN, PixelCNN & Wavenet

Factorizing the joint distribution

- Model **explicitly** the **joint probability distribution** of data streams **x** as a product of element-wise conditional distributions for each element x_i in the stream.
    - **Example:** An image **x** of size (n, n) is decomposed scanning pixels in raster mode (Row by row and pixel by pixel within every row)
    - Apply **probability chain rule**: x_i is the i-*th* pixel in the image.

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

Factorizing the joint distribution

- To model highly nonlinear and long-range correlations between pixels and their conditional distributions, we will need a powerful non-linear sequential model.
    - **Q: How can we deal with this (which model)?**

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

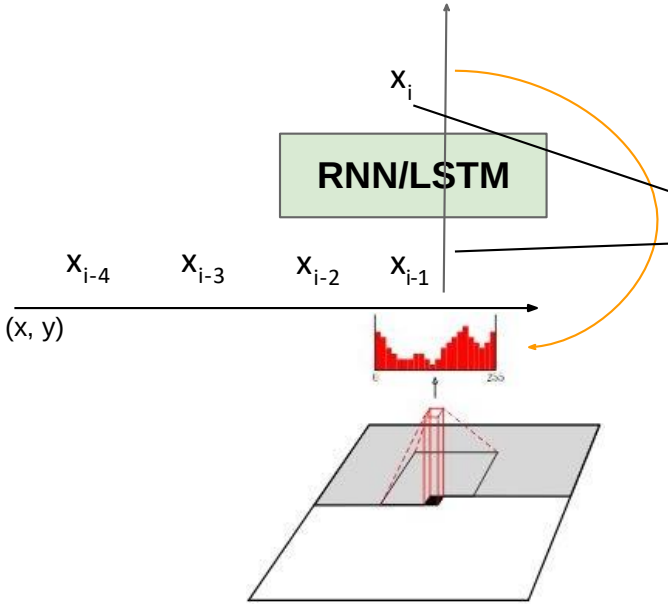
Factorizing the joint distribution

- To model highly nonlinear and long-range correlations between pixels and their conditional distributions, we will need a powerful non-linear sequential model.
    - **Q: How can we deal with this (which model)?**
        - A: Recurrent Neural Network.

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

PixelRNN

An RNN predicts the probability of each sample x_i with a categorical output distribution: Softmax



Recall RNN formulation:

$$h_t = f(W \cdot x_t + U \cdot h_{t-1} + b)$$

At every pixel, in a greyscale image: 256 classes (8 bits/pixel).

(van den Oord et al. 2016)

Factorizing the joint distribution

- To model highly nonlinear and long-range correlations between pixels and their conditional distributions, we will need a powerful non-linear sequential model.
  - **Q: How can we deal with this (which model)?**
    - A: Recurrent Neural Network.
    - B: Convolutional Neural Network.

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

(van den Oord et al. 2016)[2]

Factorizing the joint distribution

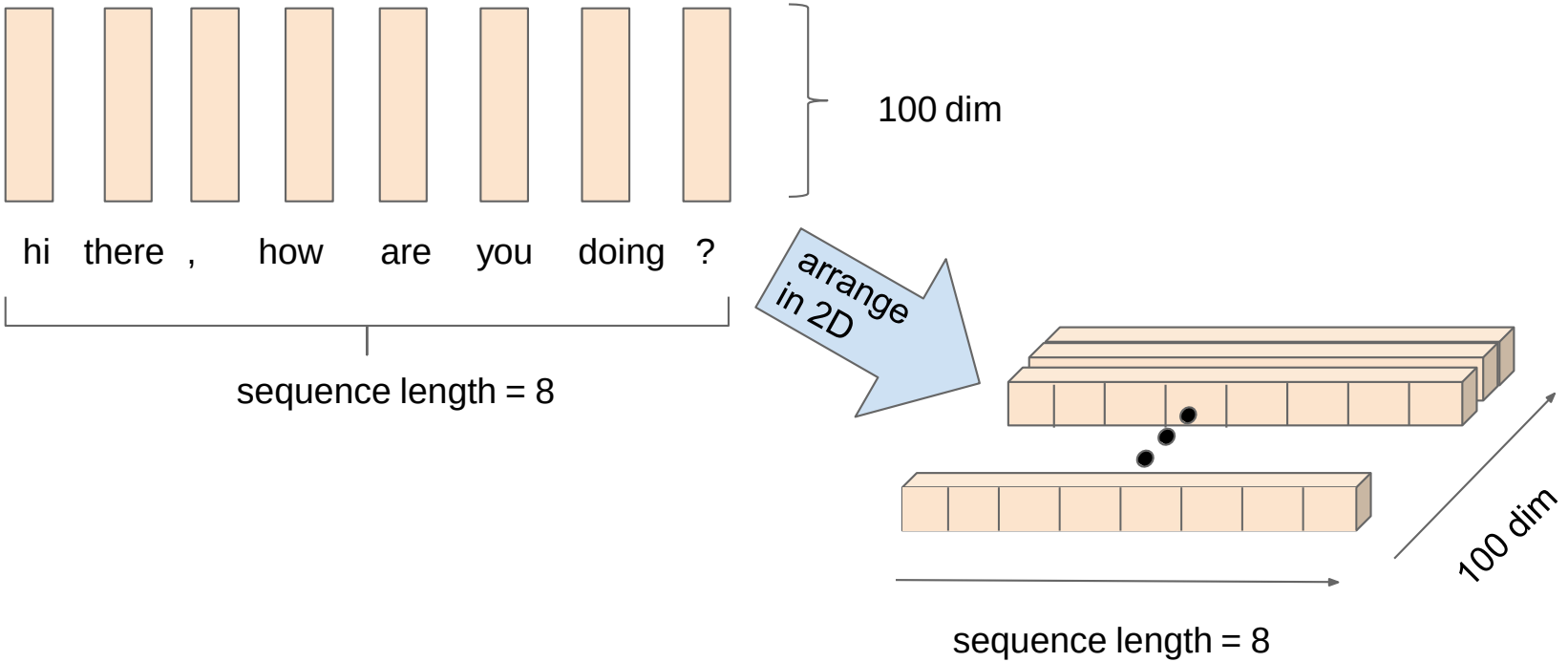
- To model highly nonlinear and long-range correlations between pixels and their conditional distributions, we will need a powerful non-linear sequential model.
    - **Q: How can we deal with this (which model)?**
        - A: Recurrent Neural Network.
        - B: Convolutional Neural Network.

$$p(\mathbf{x}) = \prod_{i=1}^{n^2} p(x_i | x_1, ..., x_{i-1})$$

My CNN is going causal...

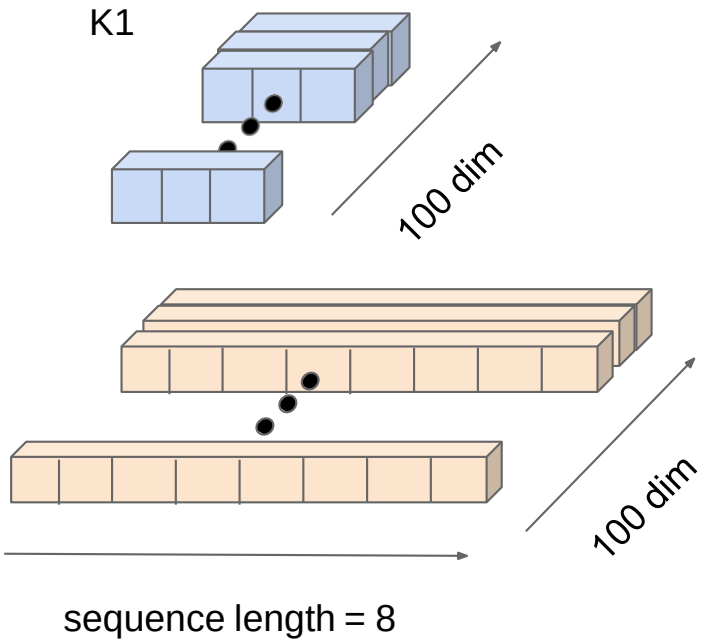
Focus in 1D to exemplify causalitaion

Let's say we have a sequence of 100 dimensional vectors describing words.



My CNN is going causal...

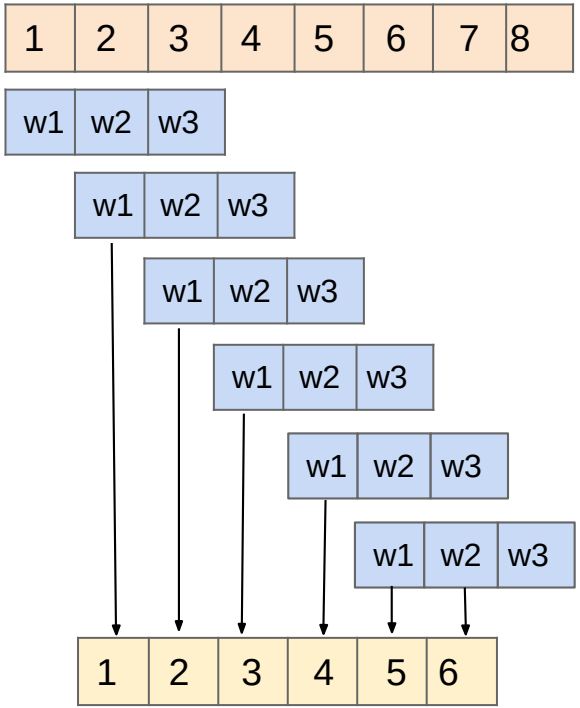
We can apply a 1D convolutional activation over the 2D matrix: for an arbitrary kernel of width=3



Each 1D convolutional kernel is a 2D matrix of size (3, 100)

My CNN is going causal...

Keep in mind we are working with 100 dimensions although here we depict just one for simplicity

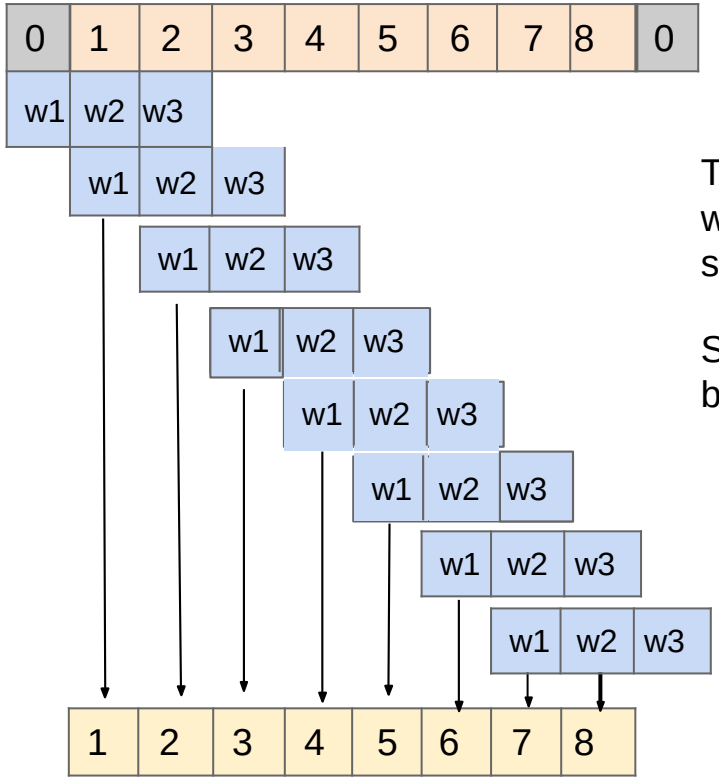


The length result of the convolution is well known to be:
seqlength - kwidth + 1 = 8 - 3 + 1 = 6

So the output matrix will be (6, 100) because there was no padding

My CNN is going causal...

When we add zero padding, we normally do so on both sides of the sequence (as in image padding)

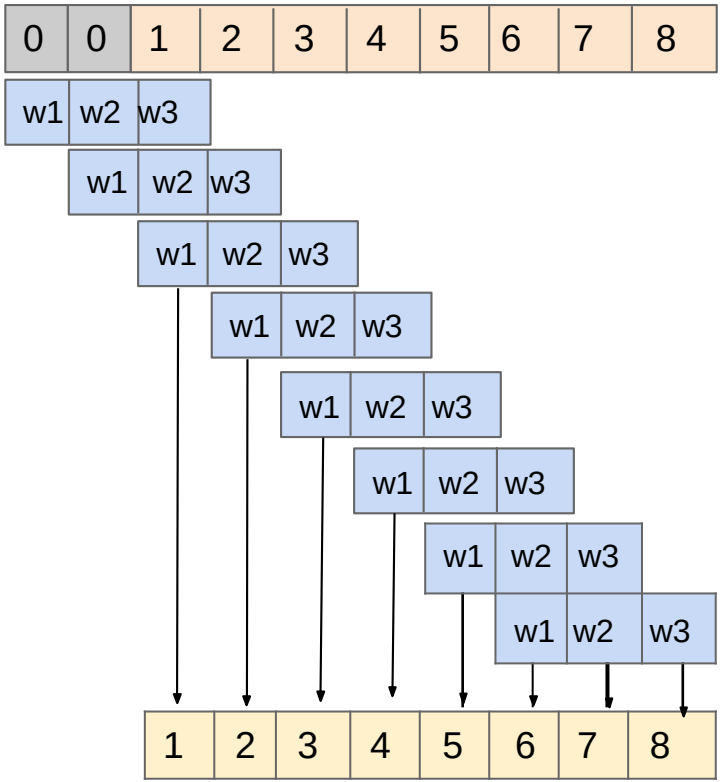


The length result of the convolution is well known to be:
seqlength - kwidth + 1 = 10 - 3 + 1 = 8

So the output matrix will be (8, 100) because we had padding

My CNN went causal

Add the zero padding just on the left side of the sequence, not symmetrically



The length result of the convolution is well known to be:
seqlength - kwidth + 1 = 10 - 3 + 1 = 8

So the output matrix will be (8, 100) because we had padding

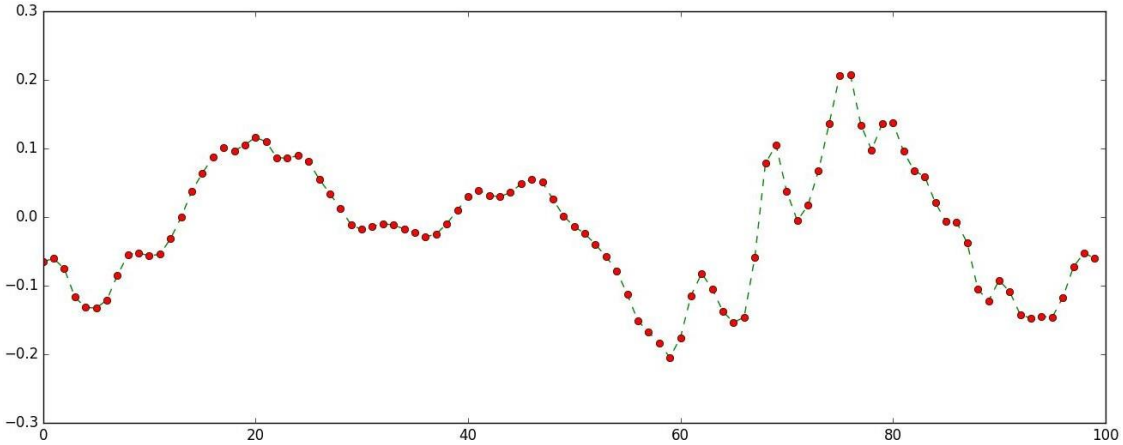
HOWEVER: now every time-step t depends on the two previous inputs as well as the current time-step → every output **is causal**

Roughly: We make a causal convolution by padding left the sequence with (kwidth - 1) zeros

PixelCNN/Wavenet

We can simply exemplify how causal convolutions help us with the SoTA model for audio generation Wavenet. We have a one dimensional stream of samples of length T:

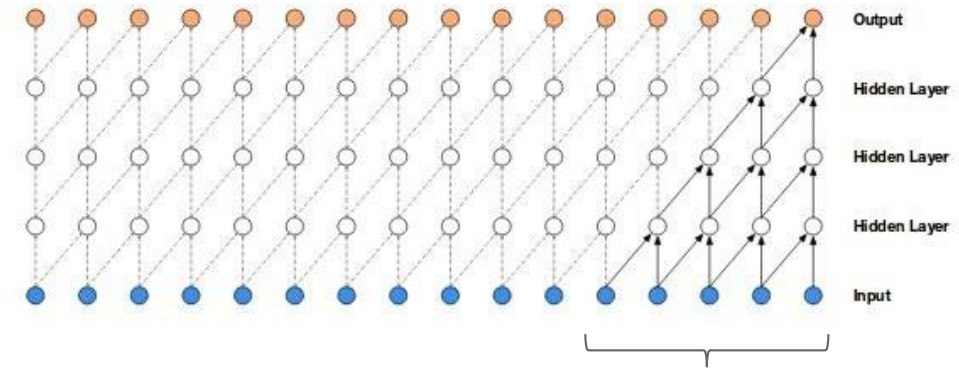
$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})$$



PixelCNN/Wavenet

We can simply exemplify how causal convolutions help us with the SoTA model for audio generation Wavenet. We have a one dimensional stream of samples of length T (e.g. audio):

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})$$



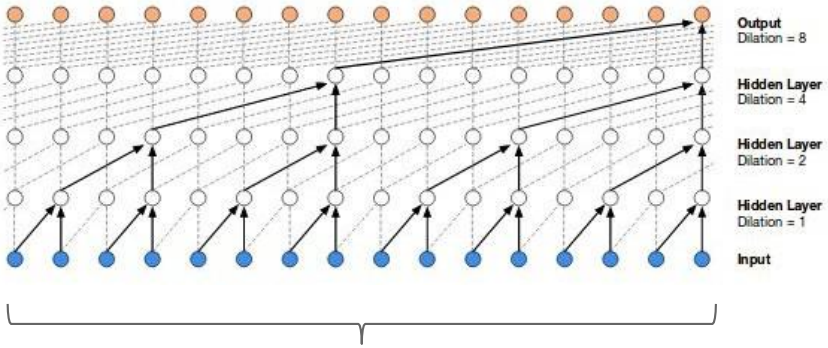
receptive field hast to be T

[(van den Oord et al. 2016)[3]]

PixelCNN/Wavenet

We can simply exemplify how causal convolutions help us with the SoTA model for audio generation Wavenet. We have a one dimensional stream of samples of length T:

$$p(\mathbf{x}) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1})$$



receptive field hast to be T

Dilated convolutions help us reach a receptive field sufficiently large to emulate an RNN!

Samples available online

Conditional PixelCNN/Wavenet

- Q: Can we make the generative model learn the natural distribution of **X** and perform a specific task conditioned on it?
  - A: Yes! we can condition each sample to an embedding/feature vector **h**, which can represent encoded text, or speaker identity, generation speed, etc.

$$p(\mathbf{x} \mid \mathbf{h}) = \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1}, \mathbf{h})$$

Conditional PixelCNN/Wavenet

- PixelCNN produces very sharp and realistic samples, but…
- **Q:** Is this a cheap generative model computationally?

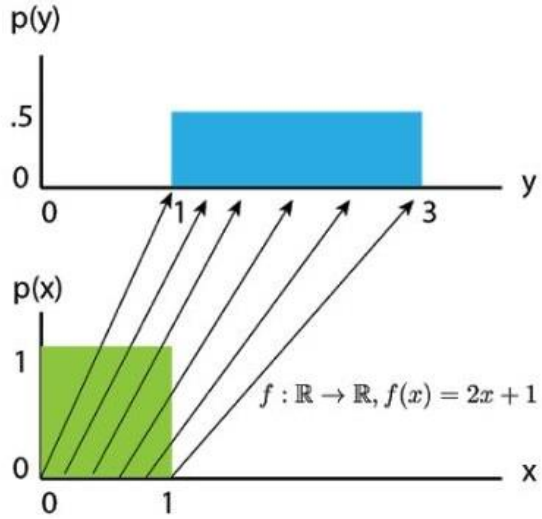
Conditional PixelCNN/Wavenet

- PixelCNN produces very sharp and realistic samples, but…
- **Q:** Is this a cheap generative model computationally?
  - **A:** Nope, take the example of wavenet: Forward 16.000 times in auto-regressive way to predict 1second of audio at standard 16 kHz sampling.

Normalizing Flows

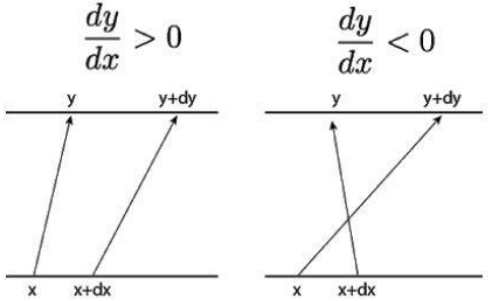
Normalizing Flows

Fundamentals of NF: learn invertible, volume-tracking transformations of distributions that we can manipulate easily [3].



Green square: Uniform(0, 1). Blue square: $Y = f(X) = 2X + 1$. Y is thus a simple affine (scale and shift) transformation of the underlying source distribution X.

Normalizing Flows



Zoom on a particular x and infinitesimally nearby point $x + dx$. Left: locally increasing function ($\frac{dy}{dx} > 0$). Right: locally decreasing function ($\frac{dy}{dx} < 0$).

To preserve total probability, the change of $p(x)$ along interval dx must be equivalent to change of $p(y)$ along interval dy:

$$p(x)dx = p(y)dy$$

Normalizing Flows

To preserve total probability, the change of $p(x)$ along interval dx must be equivalent to change of $p(y)$ along interval dy:

$$p(x)dx = p(y)dy$$

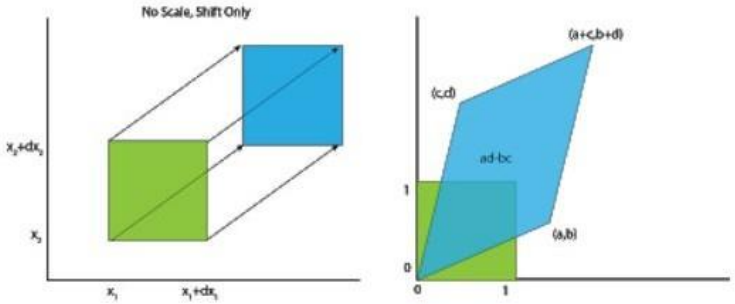
We thus only care about the amount of change in y and not its direction:

$$p(y) = p(x)|\frac{dx}{dy}|$$

$$\log p(y) = \log p(x) \log |\frac{dx}{dy}|$$

Normalizing Flows

Going N-dimensional: a non-linear transformation $f(x)$ is pictured as linear pieces of surface/volume that get transformed, and the total amount of volume change is the determinant of the transformation matrix:



Unit square in X becomes a deformed parallelogram in Y with transformation matrix $[[a, b]; [c, d]]$. Its determinant, and thus the amount of volume change, is $ad - bc$.

Normalizing Flows

Going N-dimensional: a non-linear transformation $f(x)$ is pictured as linear pieces of surface/volume that get transformed, and the total amount of volume change is the determinant of the transformation matrix:

$$y = f(x)$$

$$p(y) = p(f^{-1}(y)) \cdot |detJ(f^{-1}(y))|$$

$$\log p(y) = \log p(f^{-1}(y)) + \log |detJ(f^{-1}(y))|$$

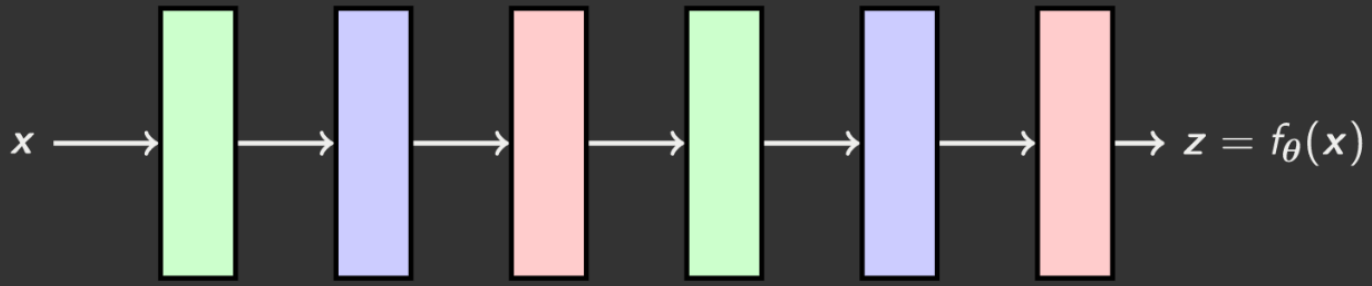
Normalizing Flows

NFs are based on the concept of bijective transformations (bijectors). A bijector will implement:

- A forward transformation $y = f(x)$ where $f : \mathbb{R}^d \rightarrow \mathbb{R}^d$.
- its inverse transformation $x = f^{-1}(y)$.
- the inverse log determinant of the Jacobian $\log |detJ(f^{-1}(y))|$ (ILDJ).

If a bijector has tunable parameters, then the bijector can actually be learned to transform our base distribution to suit arbitrary densities.

FLOWS: MAIN IDEA



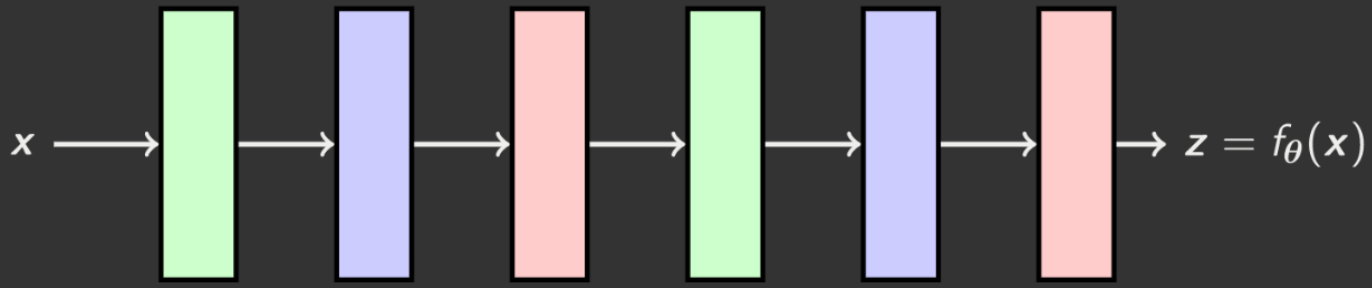
Generally: $\mathbf{z} \sim p_Z(\mathbf{z})$

Normalizing Flow: $\mathbf{z} \sim \mathcal{N}(0, 1)$

Key questions:

- How to train?
- How to evaluate $p_\theta(x)$?
- How to sample?

FLOWS: TRAINING



$$\max_\theta \sum_i \log p_\theta \left(x^{(i)} \right)$$

Need $f(\cdot)$ to be invertible and differentiable

CHANGE OF VARIABLES FORMULA

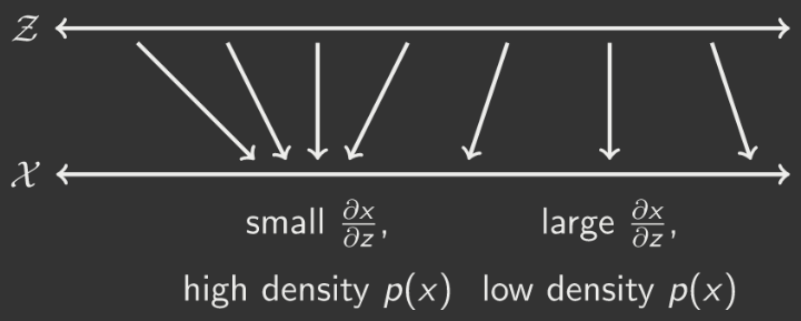
Let f^{-1} denote a differentiable, bijective mapping from space $\mathcal{Z}$ to space $\mathcal{X}$, i.e., it must be 1-to-1 and cover all of $\mathcal{X}$.

Since f^{-1} defines a one-to-one correspondence between values $\mathbf{z} \in \mathcal{Z}$ and $\mathbf{x} \in \mathcal{X}$, we can think of it as a change-of-variables transformation.

Change-of-Variables Formula from probability theory. If $\mathbf{z} = f(\mathbf{x})$, then

$$p_X(\mathbf{x}) = p_Z(\mathbf{z}) \left| det \left(\frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right) \right|$$

Intuition for the Jacobian term:



FLOWS: TRAINING

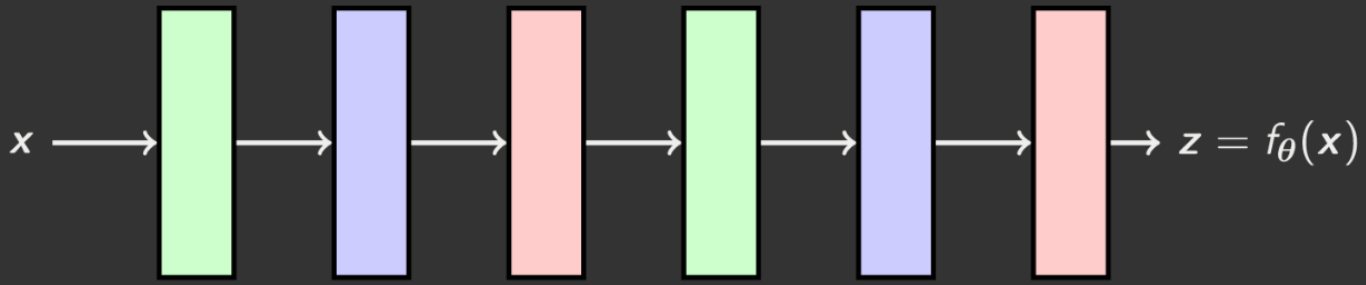
$$\max_\theta \sum_i \log p_\theta \left(x^{(i)} \right)$$

$$\begin{aligned} \mathbf{z} &= f_\theta(\mathbf{x}) \\ p_\theta(\mathbf{x}) &= p_Z(\mathbf{z}) \left| det \left(\frac{\partial \mathbf{z}}{\partial \mathbf{x}} \right) \right| \\ &= p_Z(f_\theta(\mathbf{x})) \left| det \left(\frac{\partial f_\theta(\mathbf{x})}{\partial \mathbf{x}} \right) \right| \end{aligned}$$

$$\max_\theta \sum_i \log p_\theta \left(x^{(i)} \right) = \max_\theta \sum_i \log p_Z(f_\theta(\mathbf{x}^{(i)})) + \log \left| det \left(\frac{\partial f_\theta(\mathbf{x})}{\partial \mathbf{x}} \right) \right|$$

Assuming we have an expression for p_Z, we can optimize this with Stochastic Gradient Descent

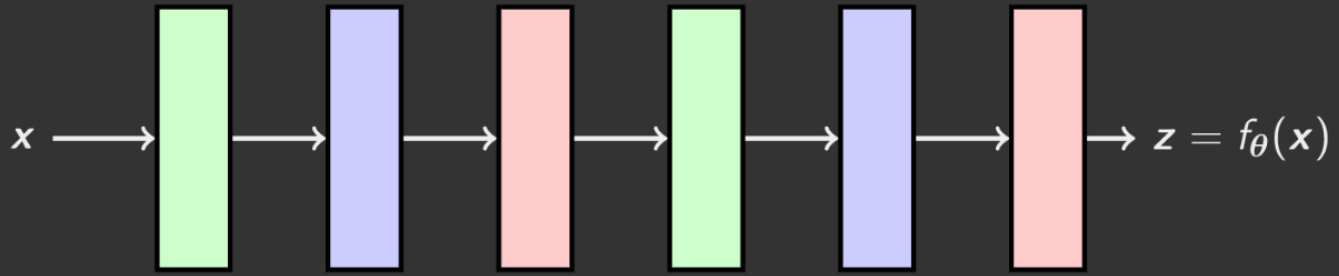
FLOWS: SAMPLING



Step 1: sample $\mathbf{z} \sim p_Z(\mathbf{z})$

Step 2: $\mathbf{x} = f_\theta^{-1}(\mathbf{z})$

CHANGE OF VARIABLES FORMULA



Problems?

- The mapping f needs to be invertible, with an easy-to-compute inverse.
- It needs to be differentiable, so that the Jacobian $\frac{\partial \mathbf{x}}{\partial \mathbf{z}}$ is defined.
- We need to be able to compute the (log) determinant.

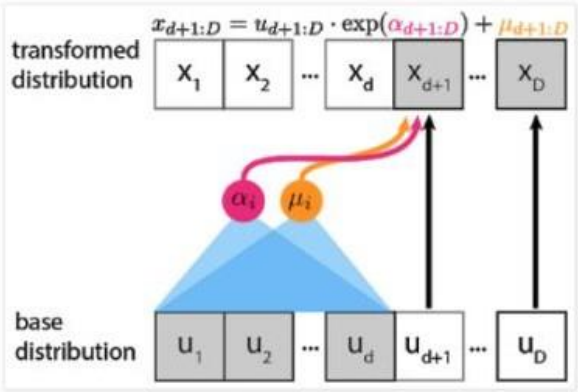
Real-NVP

Real-Non Volume Preserving Flows

Let $1 < k < d$, $\odot$ element-wise multiplication and t, s two mappings $\mathbb{R}^k \to \mathbb{R}^{d-k}$. R-NVPs are defined as

$$\boldsymbol{y}_{1:k} = \boldsymbol{z}_{1:k},$$

$$\boldsymbol{y}_{k+1:d} = \boldsymbol{z}_{k+1:d} \odot \exp(s(\boldsymbol{z}_{1:k})) + t(\boldsymbol{z}_{1:k})$$



Schematic of the data split. $\boldsymbol{u}$ is $\boldsymbol{z}$, $\boldsymbol{x}$ is $\boldsymbol{y}$, α is s and μ is t in the equations.

Real-Non Volume Preserving Flows

Forward transformation (sampling) is very straight-forward, copy first part of dimensions and scaling and shifting the other part by learnable parameters. This is fully parallelizable, and inverse transformation (inference) is like so as well!

$$\boldsymbol{z}_{1:k} = \boldsymbol{y}_{1:k}$$

$$\boldsymbol{z}_{k+1:d} = (\boldsymbol{y}_{k+1:d} - t(\boldsymbol{y}_{1:k}))/\exp(s(\boldsymbol{y}_{1:k}))$$

This operation is the affine coupling layer.

Real-Non Volume Preserving Flows

The determinant of this layer is as simple as:

$$\frac{\partial y}{\partial x^T} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial y_{d+1:D}}{\partial x_{1:d}^T} & \operatorname{diag}\left(\exp\left[s\left(x_{1:d}\right)\right]\right) \end{bmatrix}$$

Where $s(x_{1:d})$ is the predicted scale vector, easy peasy. This means it is not necessary to compute s or t Jacobians, so s and t can be arbitrarily complex (e.g. Resnets).

Real-Non Volume Preserving Flows

The determinant of this layer is as simple as:

Copy

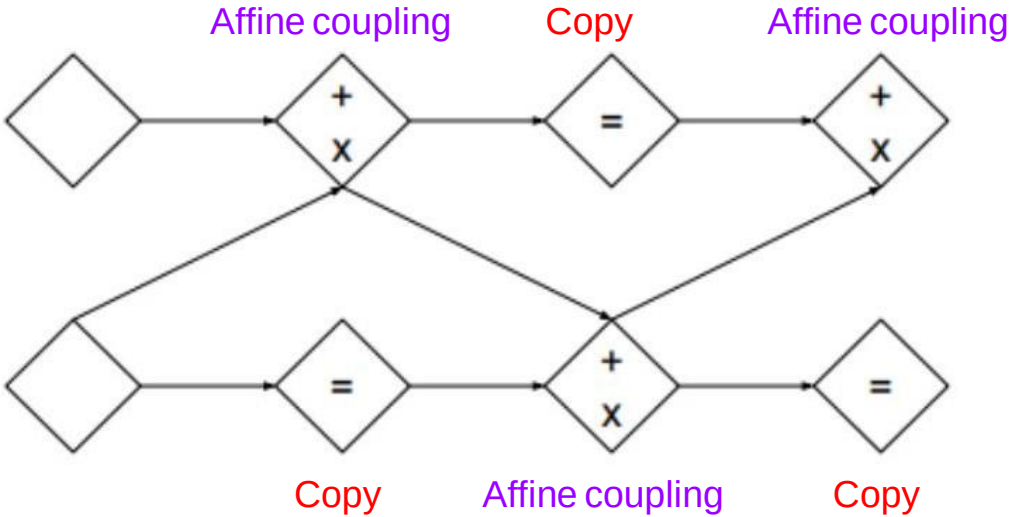
$$\frac{\partial y}{\partial x^T} = \begin{bmatrix} \mathbb{I}_d & 0 \\ \frac{\partial y_{d+1:D}}{\partial x_{1:d}^T} & \operatorname{diag}\left(\exp\left[s\left(x_{1:d}\right)\right]\right) \end{bmatrix}$$

Affine coupling

Where $s(x_{1:d})$ is the predicted scale vector, easy peasy. This means it is not necessary to compute s or t Jacobians, so s and t can be arbitrarily complex (e.g. Resnets).

Real-NVP Permutations

Due to certain dimensions being just copied and forwarded, every intermediate flow permutates the state of the intermediate vector. This way all dimensions are effectively transformed after enough levels.



GLOW

GLOW

The structure follows the same factorizing multi-scale structure as in Real-NVP, but proposing their own step of flow. Notation as in the paper regarding what we are going to see:

$$\boldsymbol{z} \sim p_\theta(\boldsymbol{z})$$

$$\boldsymbol{x} \sim g_\theta(\boldsymbol{z})$$

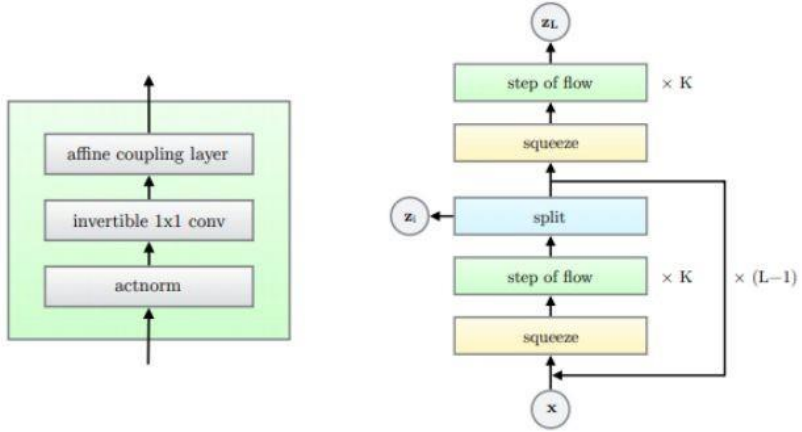
$$\mathbf{x} \overset{\mathbf{f}_1}{\longleftrightarrow} \mathbf{h}_1 \overset{\mathbf{f}_2}{\longleftrightarrow} \mathbf{h}_2 \cdots \overset{\mathbf{f}_K}{\longleftrightarrow} \mathbf{z}$$

The objective (loss), as it is an NF-based model, is the negative log-likelihood (that simple!)

$$L(D) = \frac{1}{N} \sum_{i=1}^{N} -\log p_\theta(\boldsymbol{x_i})$$

GLOW

The structure follows the same factorizing multi-scale structure as in Real-NVP, but proposing their own step of flow. There a total of K flows, with a total of L levels.



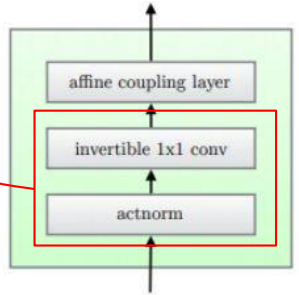
(a) One step of our flow.

(b) Multi-scale architecture (Dinh et al., 2016).

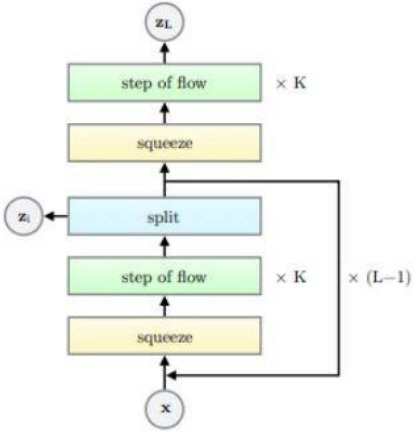
GLOW

The structure follows the same factorizing multi-scale structure as in Real-NVP, but proposing their own step of flow. There a total of K flows, with a total of L levels.

Improvements over Real-NVP.



(a) One step of our flow.



(b) Multi-scale architecture (Dinh et al., 2016).

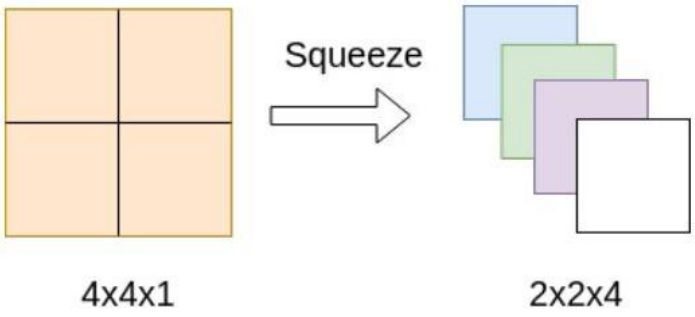
GLOW step of flow formulations

Table 1: The three main components of our proposed flow, their reverses, and their log-determinants. Here, $\mathbf{x}$ signifies the input of the layer, and $\mathbf{y}$ signifies its output. Both $\mathbf{x}$ and $\mathbf{y}$ are tensors of shape $[h \times w \times c]$ with spatial dimensions (h, w) and channel dimension c. With (i, j) we denote spatial indices into tensors $\mathbf{x}$ and $\mathbf{y}$. The function $\texttt{NN}()$ is a nonlinear mapping, such as a (shallow) convolutional neural network like in ResNets (He et al., 2016) and RealNVP (Dinh et al., 2016).

Description	Function	Reverse Function	Log-determinant
Actnorm. See Section 3.1.	$\forall i, j : \mathbf{y}_{i,j} = \mathbf{s} \odot \mathbf{x}_{i,j} + \mathbf{b}$	$\forall i, j : \mathbf{x}_{i,j} = (\mathbf{y}_{i,j} - \mathbf{b})/\mathbf{s}$	$h \cdot w \cdot \texttt{sum}(\log \lvert\mathbf{s}\rvert)$
Invertible 1×1 convolution. $\mathbf{W} : [c \times c]$. See Section 3.2.	$\forall i, j : \mathbf{y}_{i,j} = \mathbf{W}\mathbf{x}_{i,j}$	$\forall i, j : \mathbf{x}_{i,j} = \mathbf{W}^{-1}\mathbf{y}_{i,j}$	$h \cdot w \cdot \log \lvert\det(\mathbf{W})\rvert$ or $h \cdot w \cdot \texttt{sum}(\log \lvert\mathbf{s}\rvert)$ (see eq. (10))
Affine coupling layer. See Section 3.3 and (Dinh et al., 2014)	$\mathbf{x}_a, \mathbf{x}_b = \texttt{split}(\mathbf{x})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{x}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{y}_a = \mathbf{s} \odot \mathbf{x}_a + \mathbf{t}$ $\mathbf{y}_b = \mathbf{x}_b$ $\mathbf{y} = \texttt{concat}(\mathbf{y}_a, \mathbf{y}_b)$	$\mathbf{y}_a, \mathbf{y}_b = \texttt{split}(\mathbf{y})$ $(\log \mathbf{s}, \mathbf{t}) = \texttt{NN}(\mathbf{y}_b)$ $\mathbf{s} = \exp(\log \mathbf{s})$ $\mathbf{x}_a = (\mathbf{y}_a - \mathbf{t})/\mathbf{s}$ $\mathbf{x}_b = \mathbf{y}_b$ $\mathbf{x} = \texttt{concat}(\mathbf{x}_a, \mathbf{x}_b)$	$\texttt{sum}(\log(\lvert\mathbf{s}\rvert))$

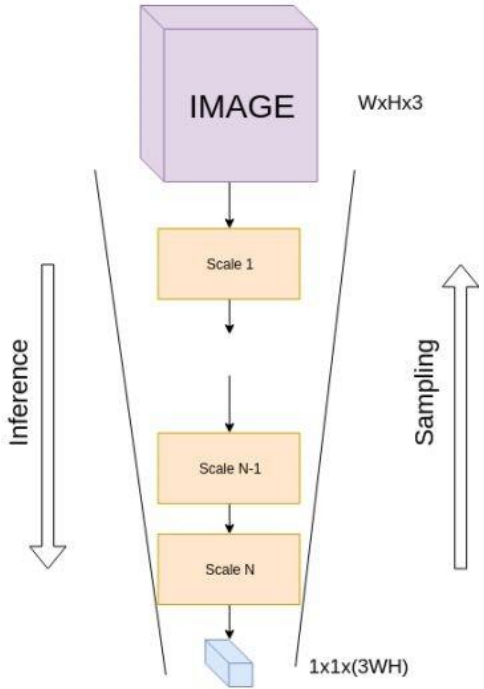
GLOW Multi-scale structure

Squeeze from image dimensions to prior dimensions ("vectorize the tensor") with a simple **reshape**. Each level of squeezing is a level of the multi-scale scheme. Each level contains K flows.



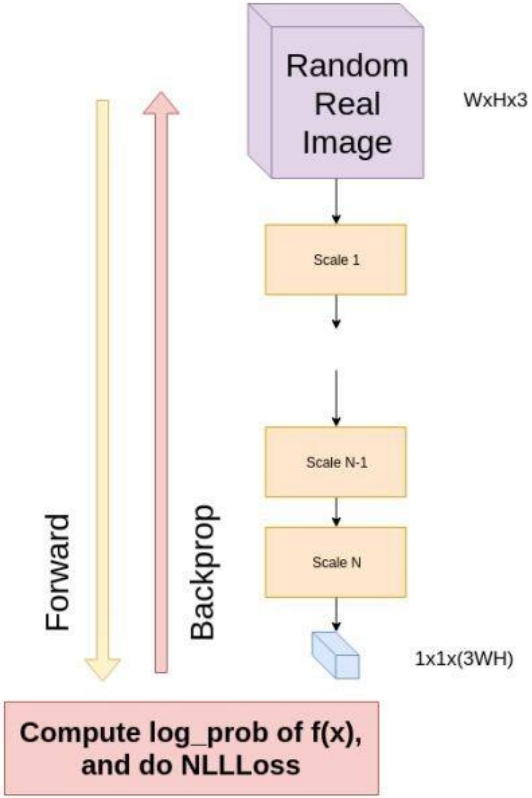
GLOW Multi-scale structure

This is a very powerful structure: we can infer z features from an image and we can sample an image from z features, EITHER WAY!



GLOW Multi-scale structure

Additionally, during training loss is computed in the simplistic Z space!



$$L(D) = \frac{1}{N}\sum_{i=1}^{N} -\log p_\theta(\boldsymbol{x_i})$$

GLOW Results on CelebA

We can sample with a temperature factor T that simply multiplies the scale s in the coupling layers.

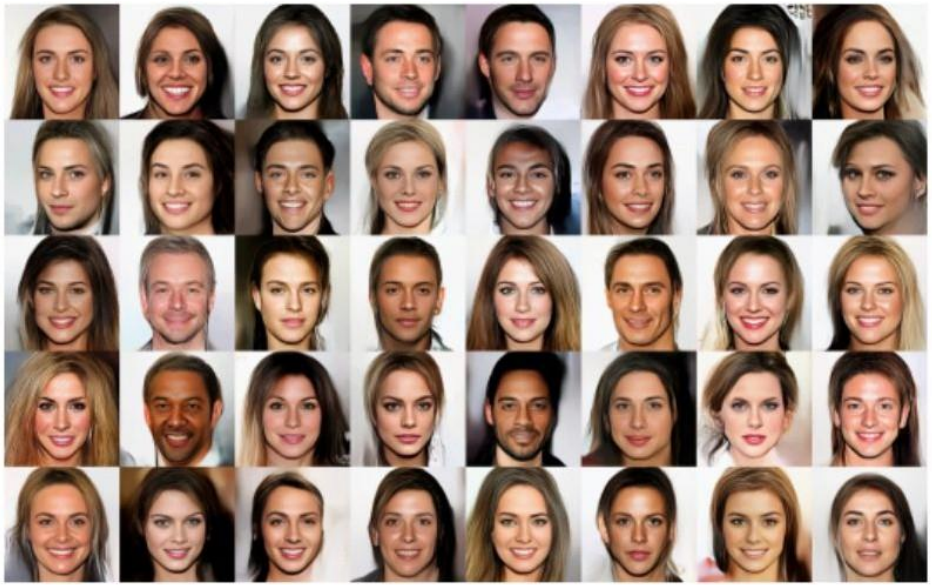


Figure 4: Random samples from the model, with temperature 0.7

GLOW Results with temperature

We can sample with a temperature factor T that simply multiplies the scale s in the coupling layers. Sample quality and diversity varies with T factor, being 0.7 a sweet spot for the authors in this work.

Figure 8: Effect of change of temperature. From left to right, samples obtained at temperatures $0, 0.25, 0.6, 0.7, 0.8, 0.9, 1.0$

GLOW Results interpolating Z

We can infer two real images onto their respective latent representations and sample from their linear interpolations to smoothly mix both images.

Figure 5: Linear interpolation in latent space between real images

GLOW Semantic Manipulation

In celebA there is a binary label per attribute (absence/presence): blonde, smiling, etc. So there are 30.000 binary labels per attritube (from training data).

The z_{pos} average latent vector is calculated for images with the attribute and z_{neg} for images without it. The difference $z_{pos} - z_{neg}$ can be used as a direction for manipulating a certain attribute.

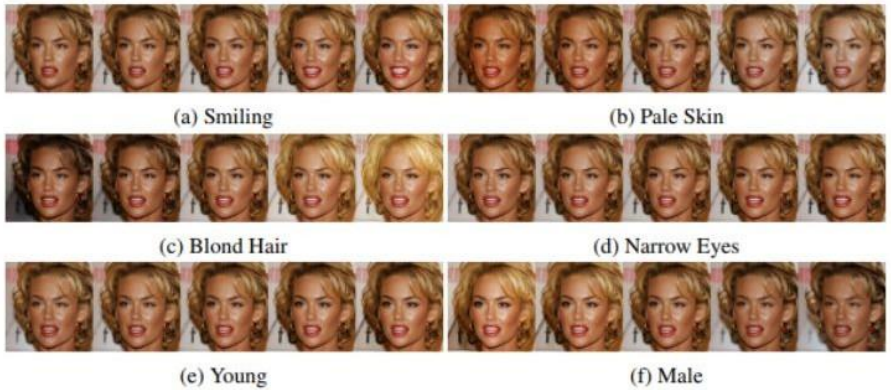


Figure 6: Manipulation of attributes of a face. Each row is made by interpolating the latent code of an image along a vector corresponding to the attribute, with the middle image being the original image

GLOW's depth dependency

Without enough depth, there is a lack of capacity to generate long-range dependencies.

Figure 9: Samples from shallow model on left vs deep model on right. Shallow model has $L = 4$ levels, while deep model has $L = 6$ levels

WaveGLOW

(Prenger et al. 2018)

Modified structure to deal with 1D audio data, giving SoTA results without auto-regressive restriction.

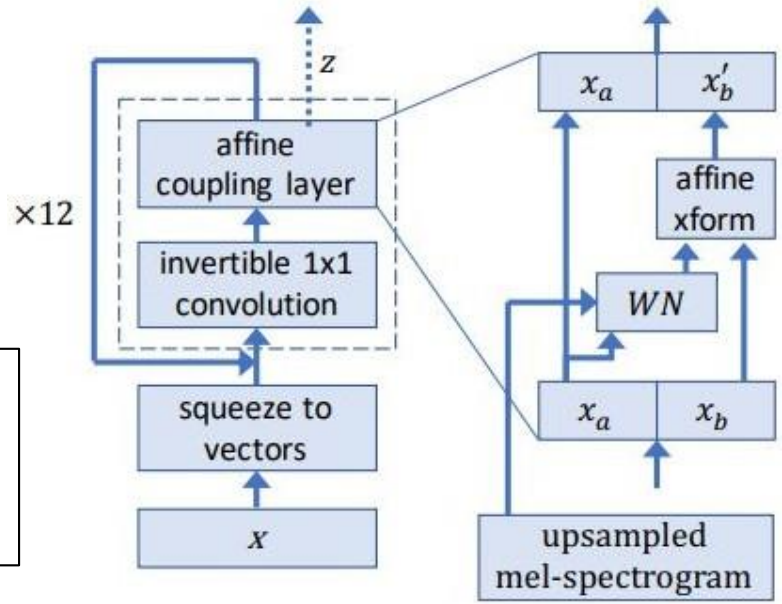
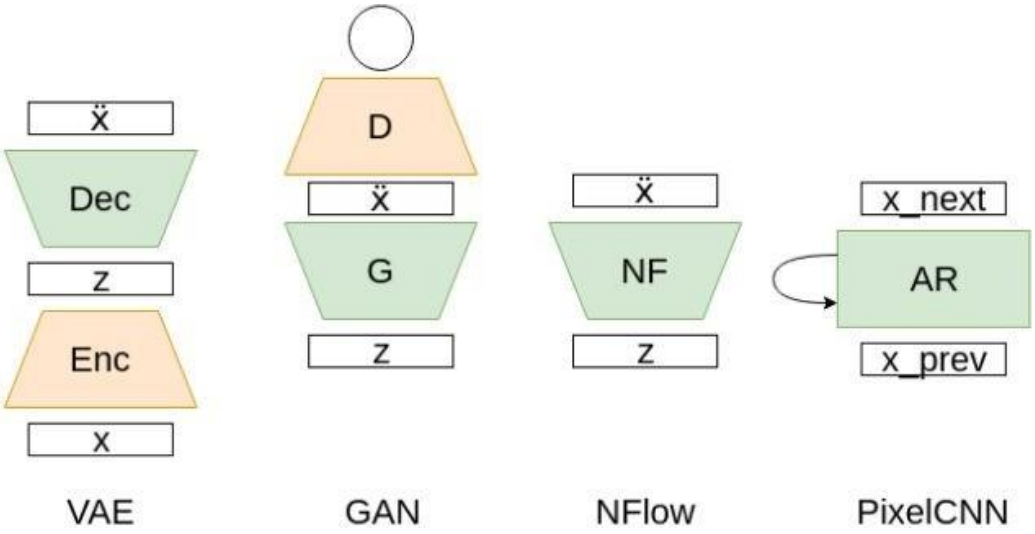


Fig. 1: WaveGlow network

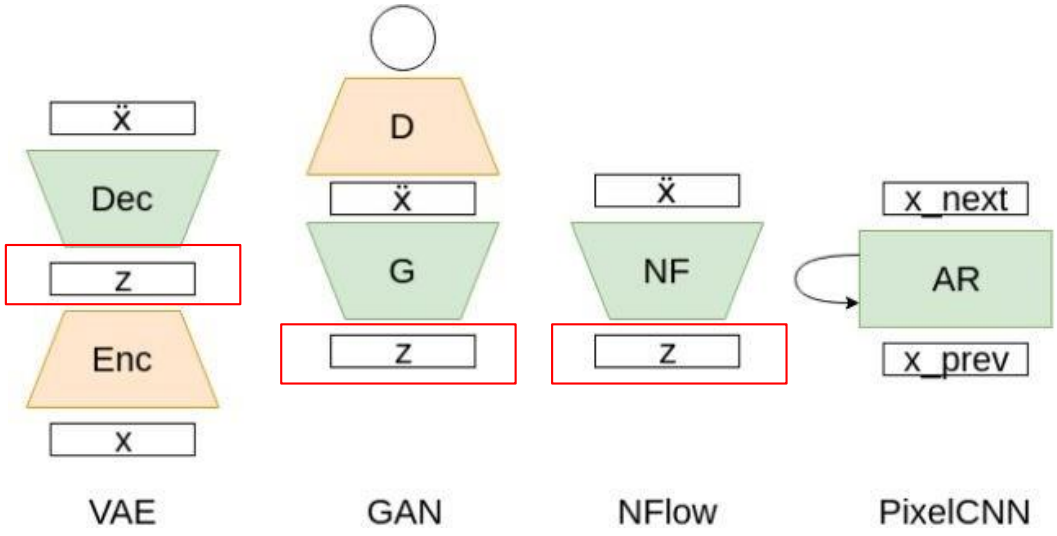
Audio samples available online

Models Comparison

Models comparison

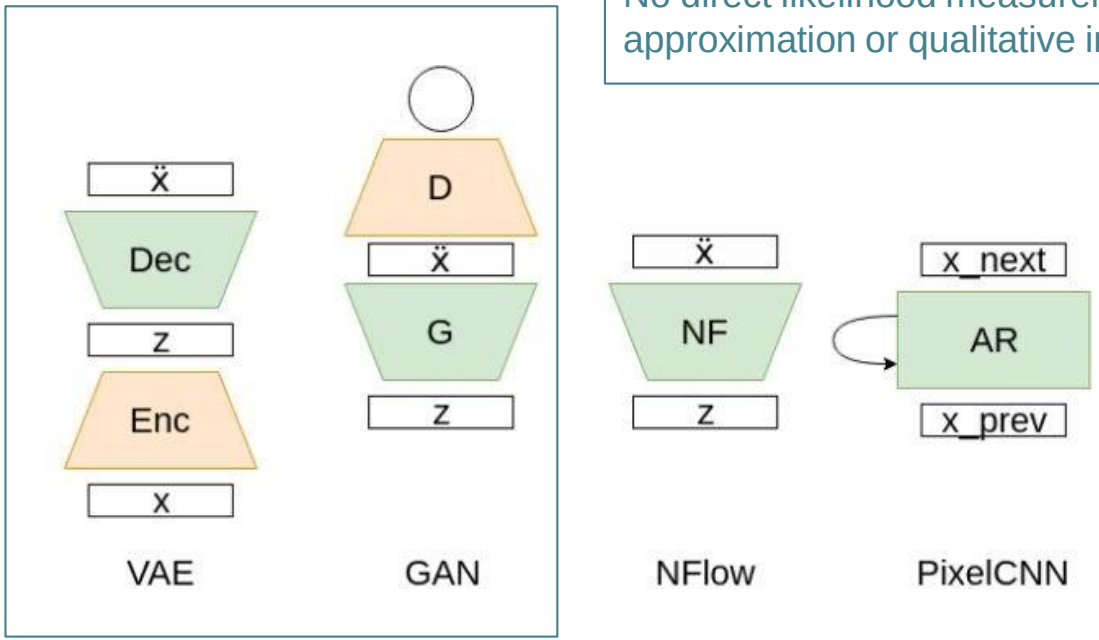


Models comparison



Latent space → feature disentangle, we can navigate and control better what we generate

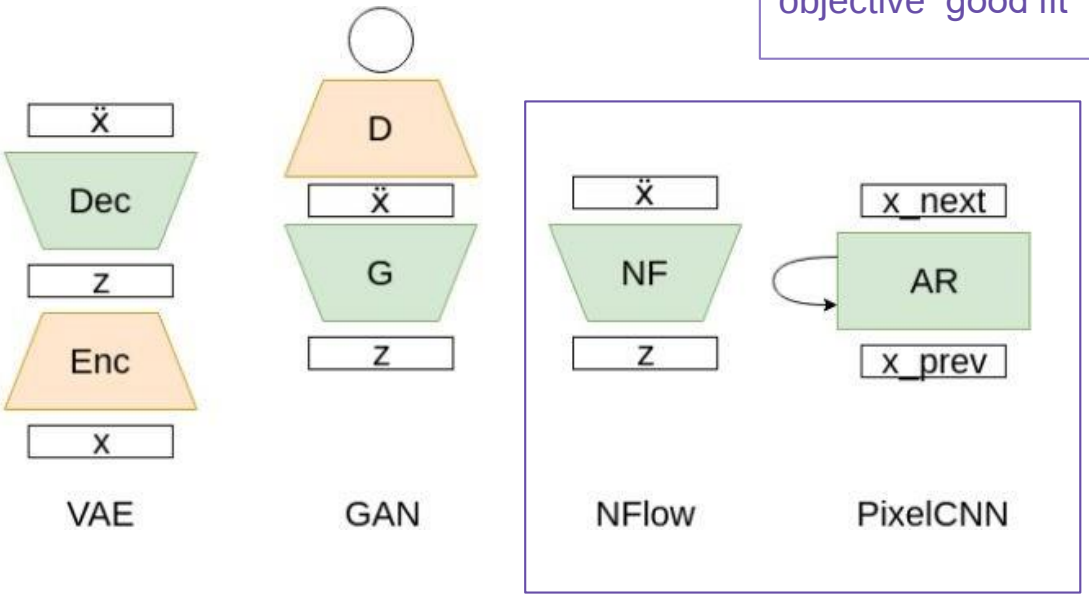
Models comparison



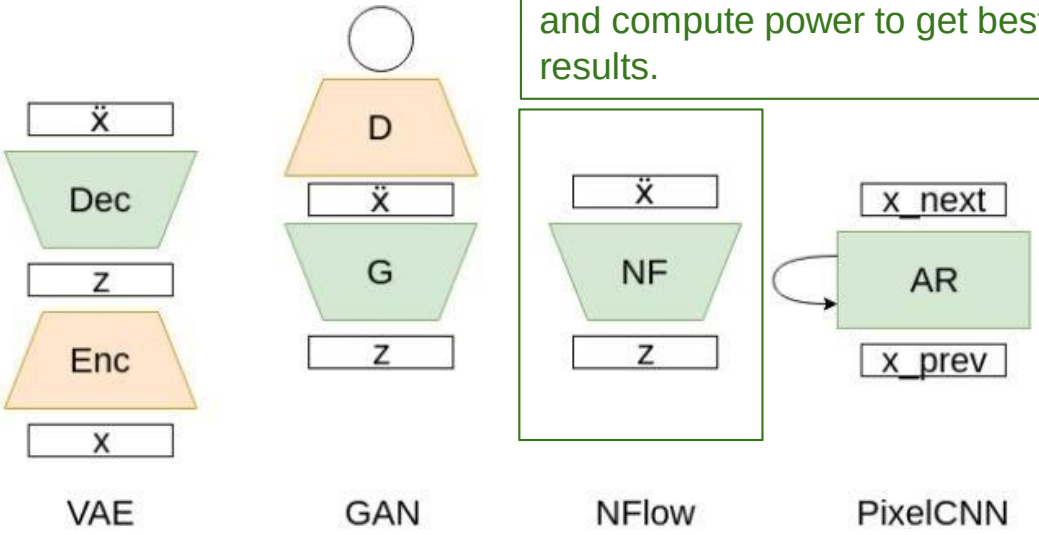
No direct likelihood measurement, just an approximation or qualitative insights.

Models comparison

Direct likelihood values, direct objective “good fit” measurements.



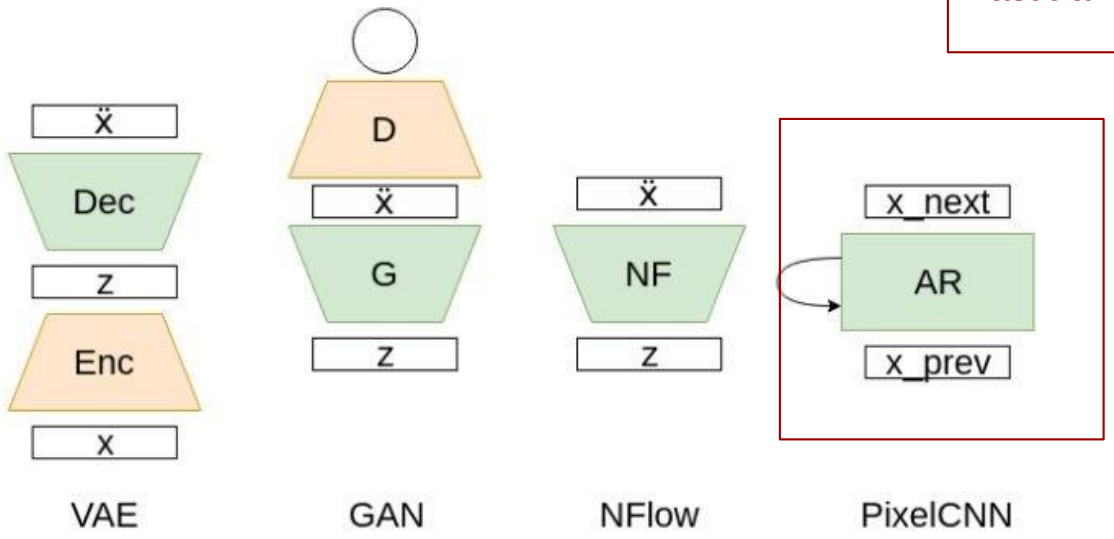
Models comparison



Very deep models (many flow steps), need much much memory and compute power to get best results.

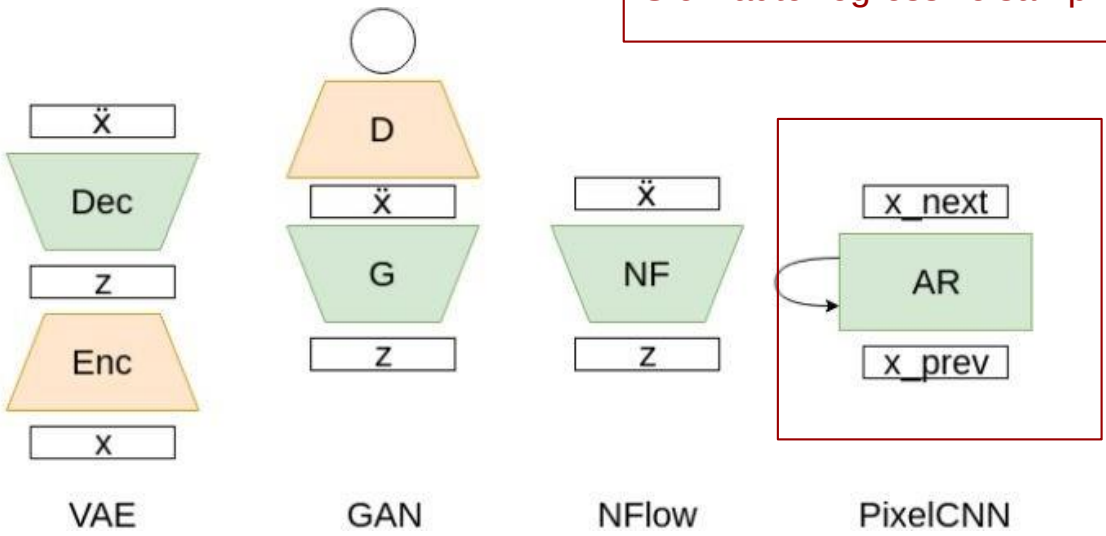
Models comparison

Fast training.

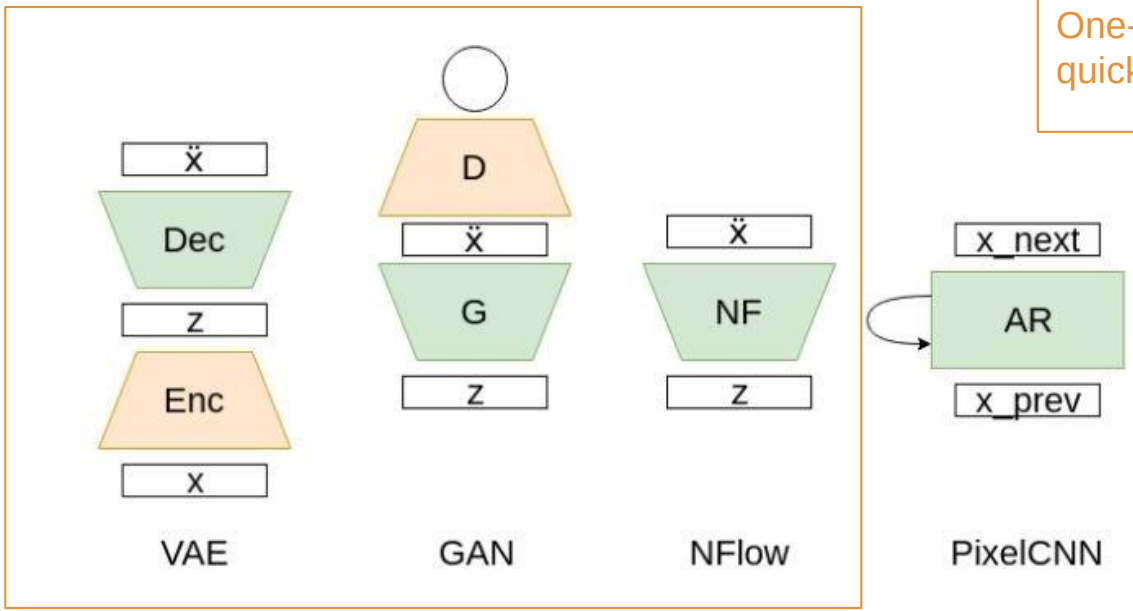


Models comparison

Slow auto-regressive sampling.



Models comparison



One-shot predictions, quickest to sample from.

Denoising Diffusion Probabilistic Models

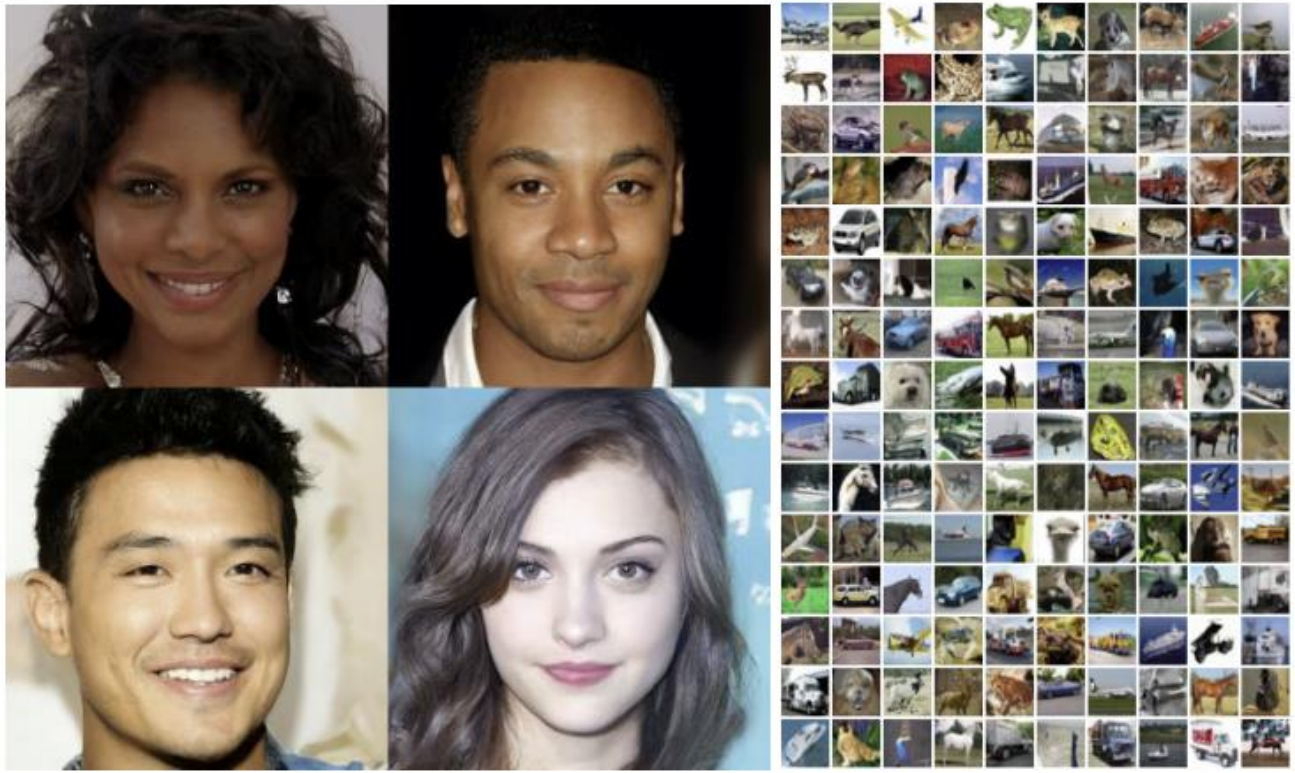


Figure 1: Generated samples on CelebA-HQ 256 × 256 (left) and unconditional CIFAR10 (right)

Denoising Diffusion Probabilistic Models

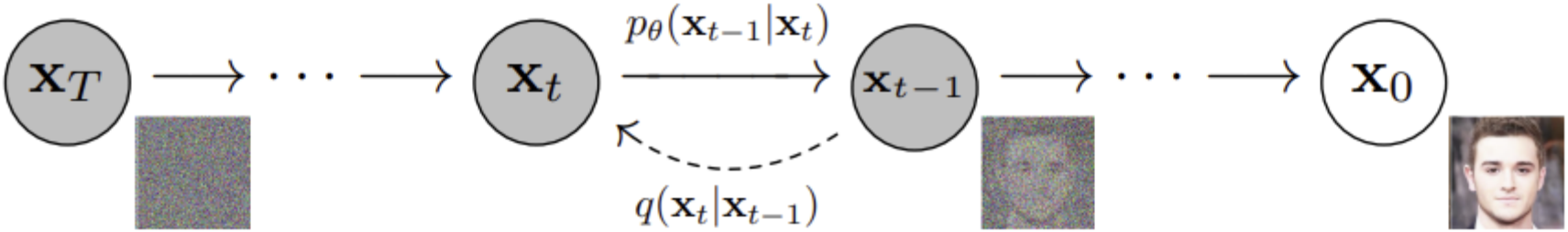


Figure 2: The directed graphical model considered in this work.

$$p_\theta(\mathbf{x}_{0:T}) := p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t), \qquad p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t) := \mathcal{N}(\mathbf{x}_{t-1}; \boldsymbol{\mu}_\theta(\mathbf{x}_t, t), \boldsymbol{\Sigma}_\theta(\mathbf{x}_t, t)) \tag{1}$$

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) := \prod_{t=1}^{T} q(\mathbf{x}_t|\mathbf{x}_{t-1}), \qquad q(\mathbf{x}_t|\mathbf{x}_{t-1}) := \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}) \tag{2}$$

Denoising Diffusion Probabilistic Models

Algorithm 1 Training

1: **repeat**
2: $\mathbf{x}_0 \sim q(\mathbf{x}_0)$
3: $t \sim \text{Uniform}(\{1, \dots, T\})$
4: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
5: Take gradient descent step on
$$\nabla_\theta \left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2$$
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: $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ if $t > 1$, else $\mathbf{z} = \mathbf{0}$
4: $\mathbf{x}_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(\mathbf{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \boldsymbol{\epsilon}_\theta(\mathbf{x}_t, t) \right) + \sigma_t \mathbf{z}$
5: **end for**
6: **return** $\mathbf{x}_0$

$$L_{\text{simple}}(\theta) := \mathbb{E}_{t, \mathbf{x}_0, \boldsymbol{\epsilon}} \left[\left\| \boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t) \right\|^2 \right]$$

$\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$ $\quad \alpha_t := 1 - \beta_t$ and $\bar{\alpha}_t := \prod_{s=1}^{t} \alpha_s$

$\boldsymbol{\epsilon}_\theta$ is a function approximator intended to predict $\boldsymbol{\epsilon}$ from $\mathbf{x}_t$.

Denoising Diffusion Probabilistic Models

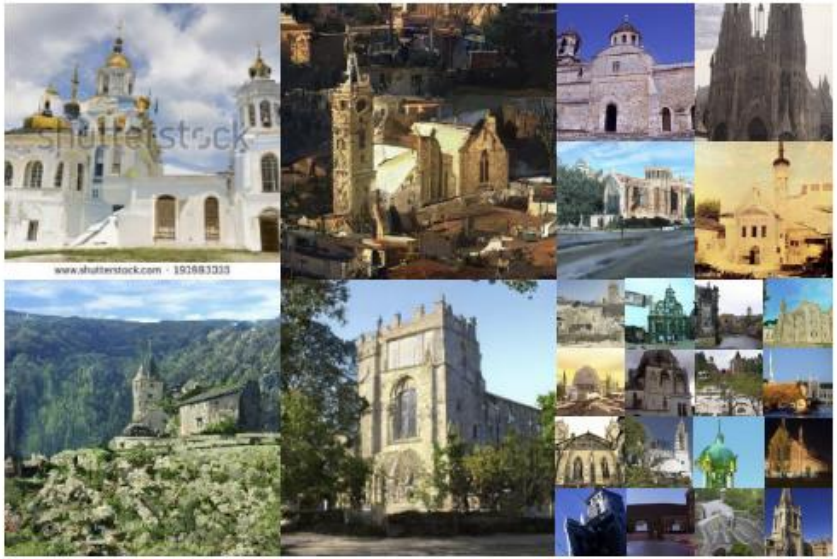


Figure 3: LSUN Church samples. FID=7.89

Figure 4: LSUN Bedroom samples. FID=4.90

Thanks! Questions?