Autoregressive Models, Variational Autoencoders, and … Diffusion Models

Intelligent Visual Computing

Evangelos Kalogerakis

How to generate visual data?

- Encoder-Decoders
- Generative Adversarial Networks
- **Autoregressive models**
  - PixelRNN / PixelCNN
  - VQGAN
  - PolyGen
- Variational Autoencoders
- Diffusion models

Autoregressive Models

Explicitly models data distribution by assuming that **our data consists of individual elements**

$$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 \ldots\}$$

e.g., an image consists of a (flattened) series of pixels, or a mesh consists of a series of triangles ...

Autoregressive Models

Explicitly models data distribution by assuming that **our data consists of individual elements**

$$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 \dots\}$$

e.g., an image consists of a (flattened) series of pixels, or a mesh consists of a series of triangles ...

Data distribution is modeled as:

$$P(\mathbf{X}) = P(\mathbf{x}_1) \cdot P(\mathbf{x}_2 \mid \mathbf{x}_1) \cdot P(\mathbf{x}_3 \mid \mathbf{x}_1, \mathbf{x}_2) \cdot P(\mathbf{x}_4 \mid \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3) \dots$$

Autoregressive Models

Explicitly models data distribution by assuming that **our data consists of individual elements**

$$\mathbf{X} = \{\mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, \mathbf{x_4 ...}\}$$

e.g., an image consists of a (flattened) series of pixels, or a mesh consists of a series of triangles ...

Data distribution is modeled as:

$$P(\mathbf{X}) = \prod_{t=0}^{T} P(\mathbf{x}_{t+1} \mid \mathbf{x_1}, \mathbf{x_2}, \mathbf{x_3}, ..., \mathbf{x}_t)$$

Autoregressive Models

Explicitly models data distribution by assuming that **our data consists of individual elements**

$$\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4 \ldots\}$$

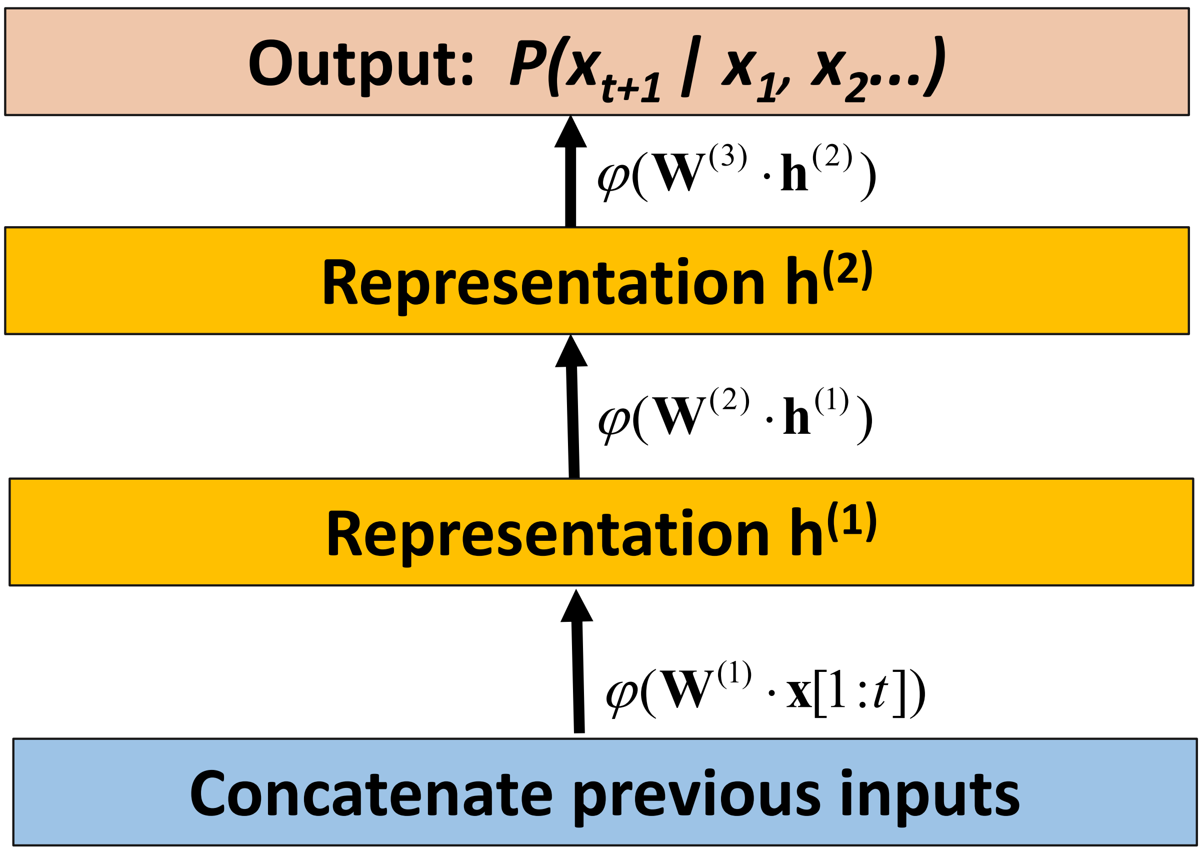
e.g., an image consists of a (flattened) series of pixels, or a mesh consists of a series of triangles ...

Data distribution is modeled as:

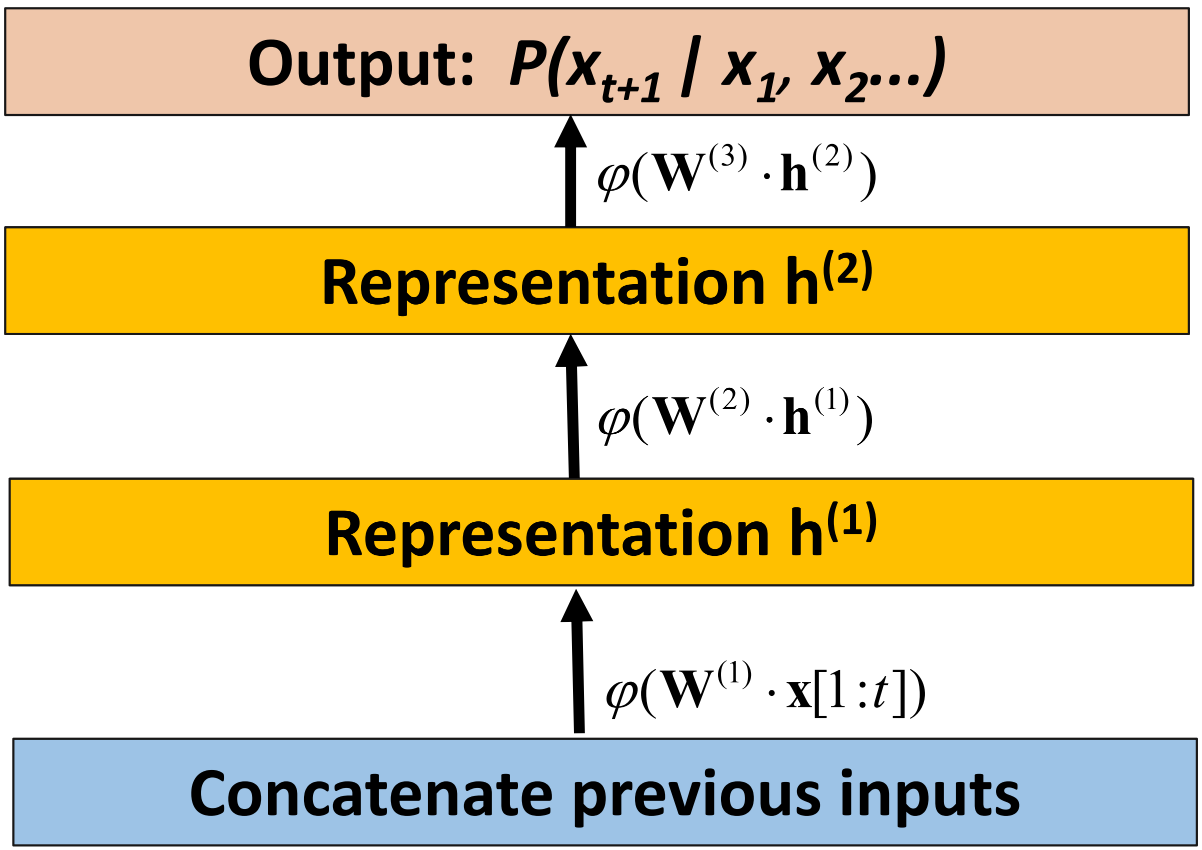
$$P(\mathbf{X}) = \boxed{\prod_{t=0}^{T} P(\mathbf{x}_{t+1} \mid \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_t)}$$

... a generative model conditioned on previous input
model it with a network (what network?)

One idea (?)


Output: $P(x_{t+1} / x_1, x_2 \ldots)$
$\varphi(\mathbf{W}^{(3)} \cdot \mathbf{h}^{(2)})$
Representation $\mathbf{h}^{(2)}$
$\varphi(\mathbf{W}^{(2)} \cdot \mathbf{h}^{(1)})$
Representation $\mathbf{h}^{(1)}$
$\varphi(\mathbf{W}^{(1)} \cdot \mathbf{x}[1:t])$
Concatenate previous inputs

One idea (?)

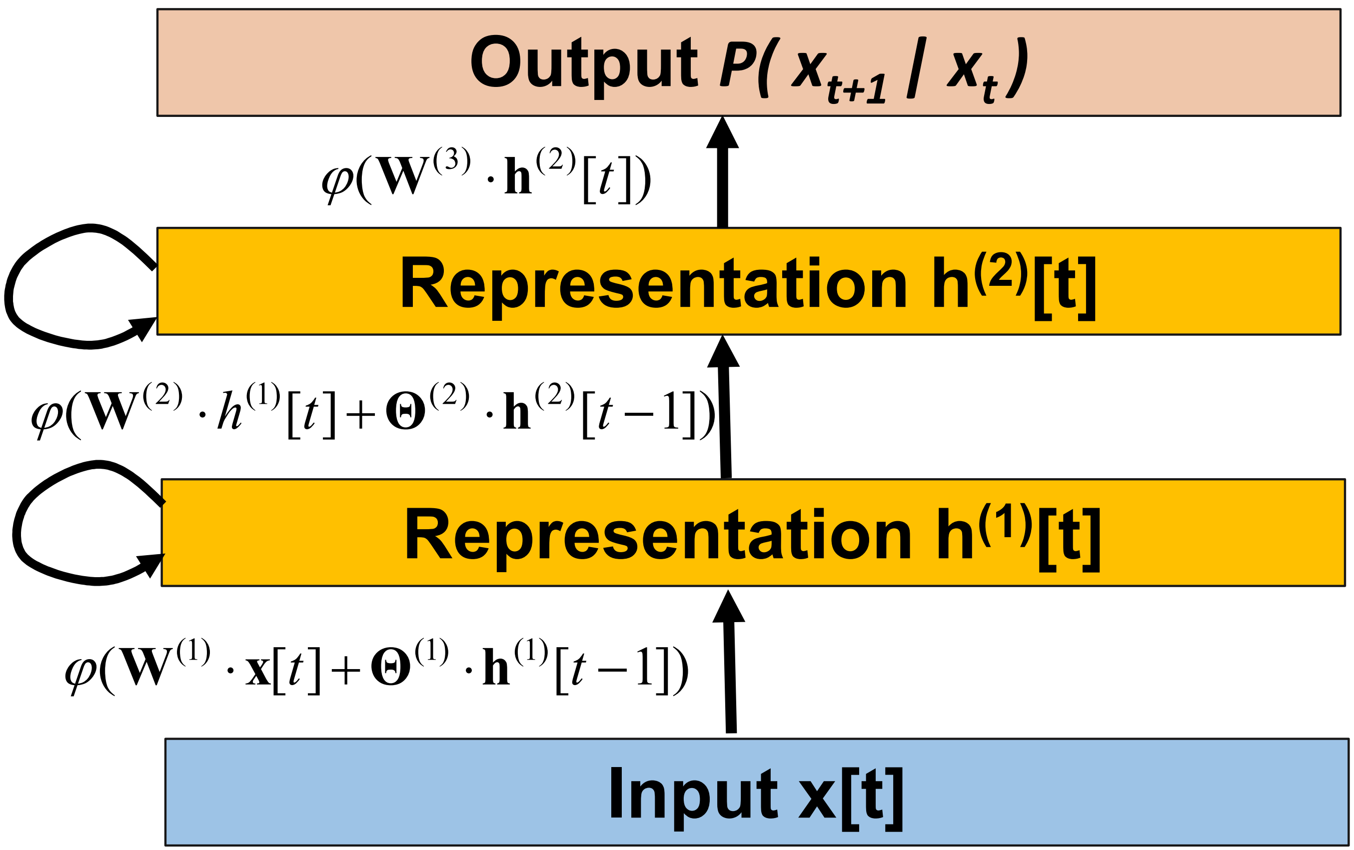


How many previous inputs one should use? Large input => Can be too complex model to learn

Recurrent Net (RNN)

Introduces a loop allowing information to pass from previous inputs

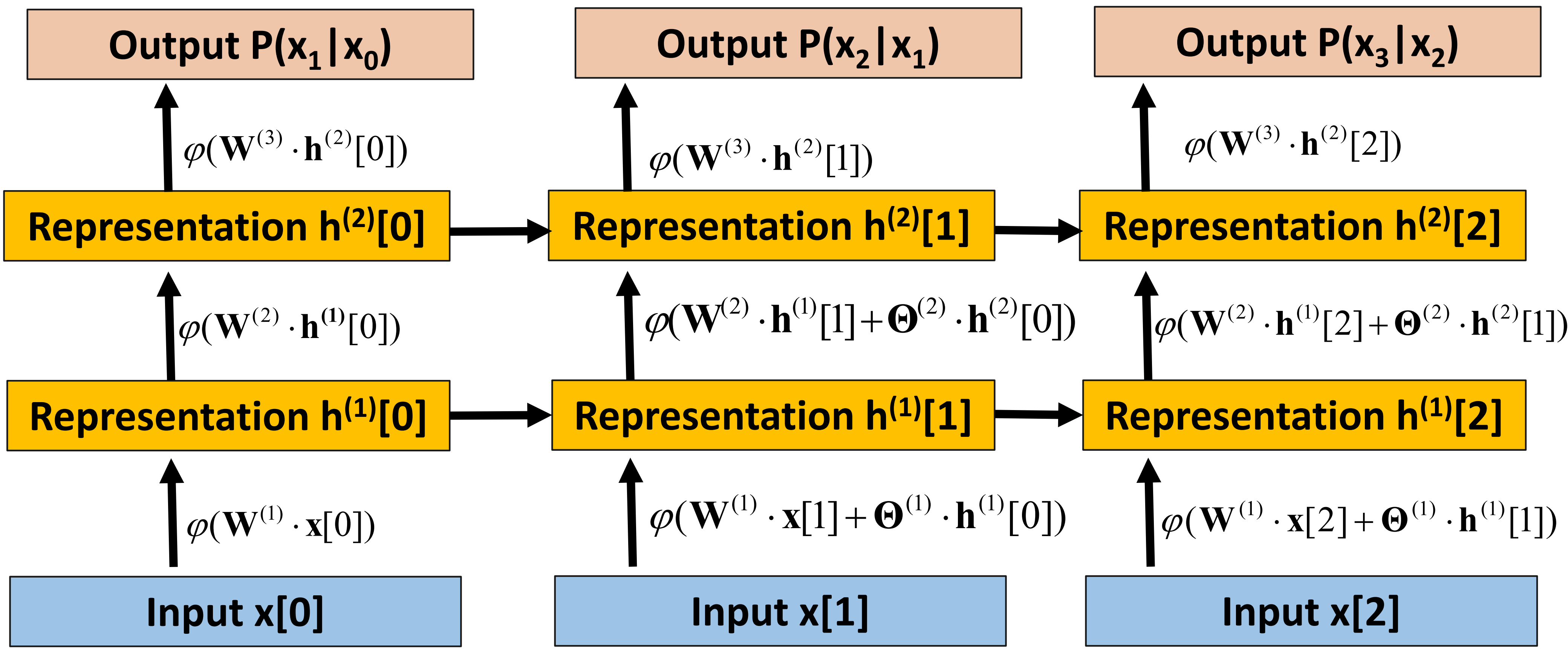
Note: RNNs are not autoregressive since the previous x's are not provided explicitly – instead outputs depend on previous inputs via some hidden state



Recurrent Net (RNN)

Another way to see this network is to unroll it

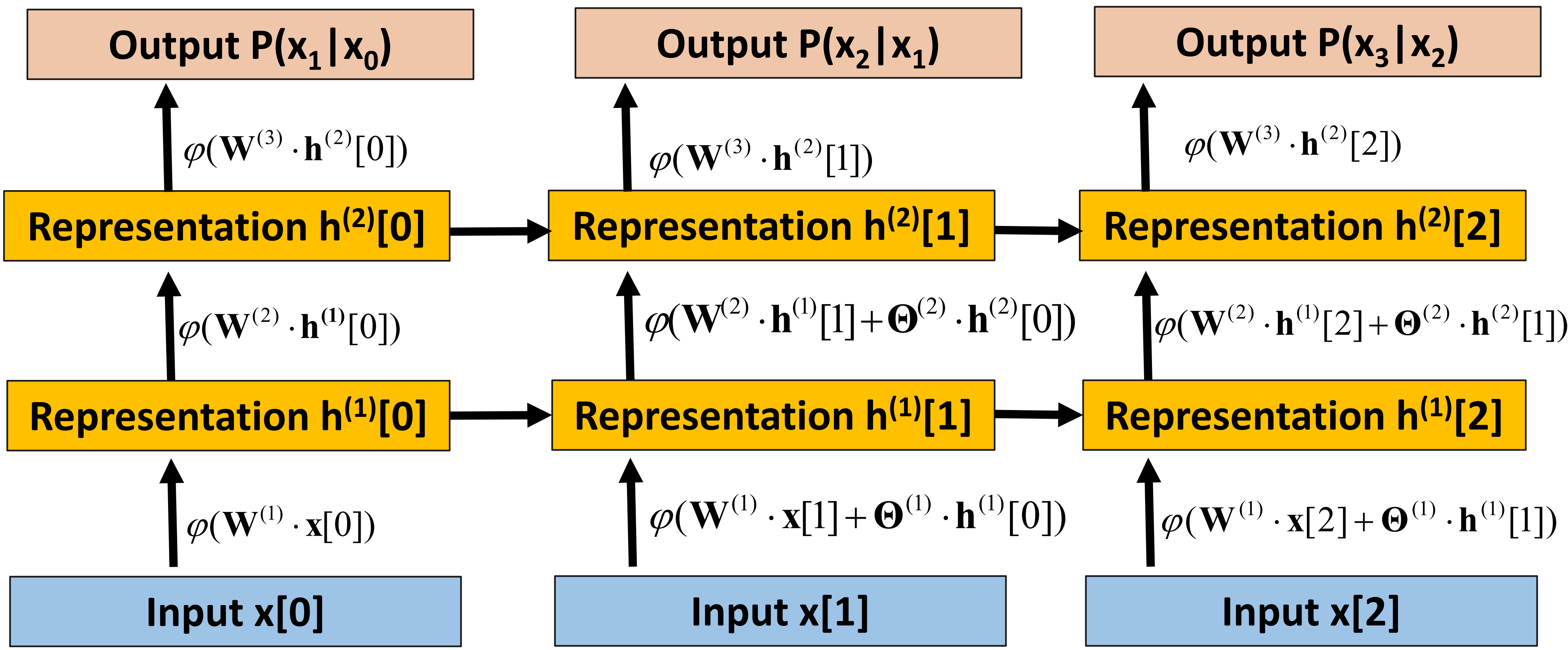
Predictions can now be done like in a typical forward pass!



Recurrent Net (RNN)

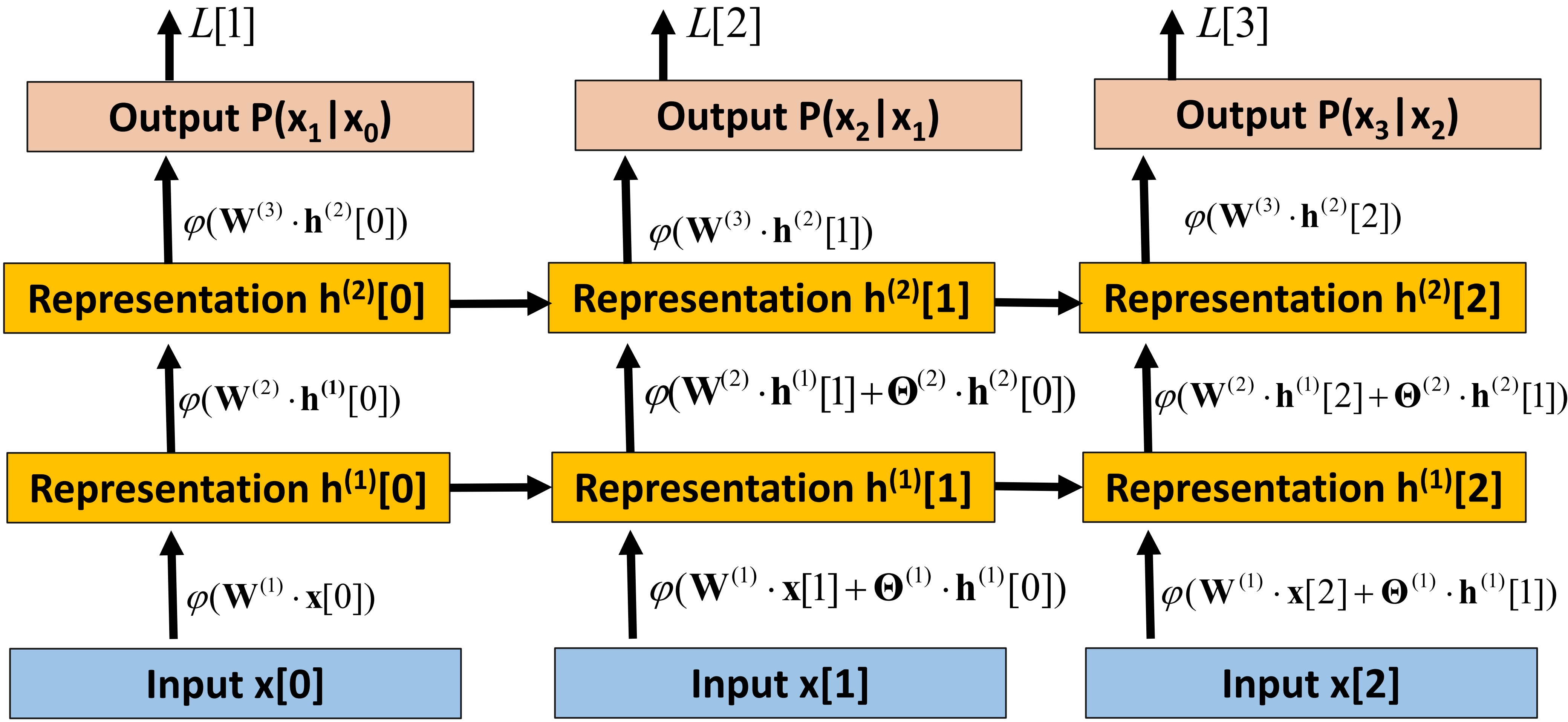
Similarly, all parameters can be learned through backpropagation!

Note: the parameters are shared by all time steps in the network - the gradient of each output depends on the calculations of the current time step + previous steps.



Recurrent Net (RNN)

We can define losses, given ground-truth outputs at each time step



How to generate visual data?

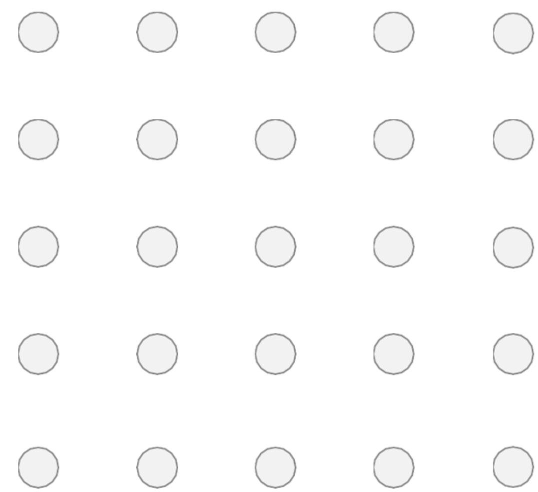
- Encoder-Decoders
- Generative Adversarial Networks
- **Autoregressive models**
  - **PixelRNN / PixelCNN**
  - VQGAN
  - PolyGen
- Variational Autoencoders
- Diffusion models

PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the pixel on the left and from the one above

$$\mathbf{h}_{x,y} = f(\mathbf{h}_{x-1,y}, \mathbf{h}_{x,y-1}; \mathbf{W})$$



At each pixel, decode hidden state to red, then blue, then green
i.e., softmax over each channel intensity value [0, 1, …, 255] (256 categories)

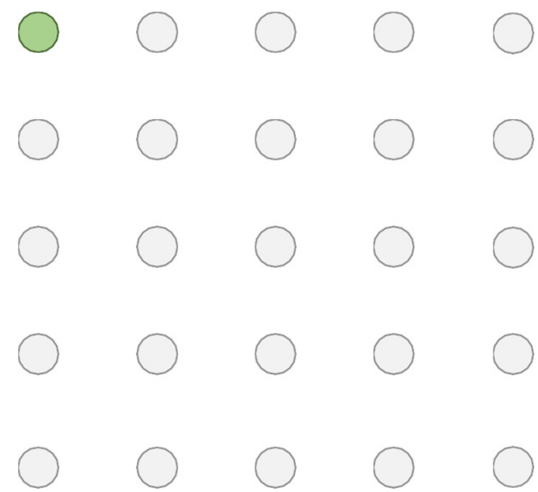
Van den Oord et al, "Pixel Recurrent Neural Networks", 2016

PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the pixel on the left and from the one above

$$\mathbf{h}_{x,y} = f(\mathbf{h}_{x-1,y}, \mathbf{h}_{x,y-1}; \mathbf{W})$$



At each pixel, decode hidden state to red, then blue, then green
i.e., softmax over each channel intensity value [0, 1, …, 255] (256 categories)

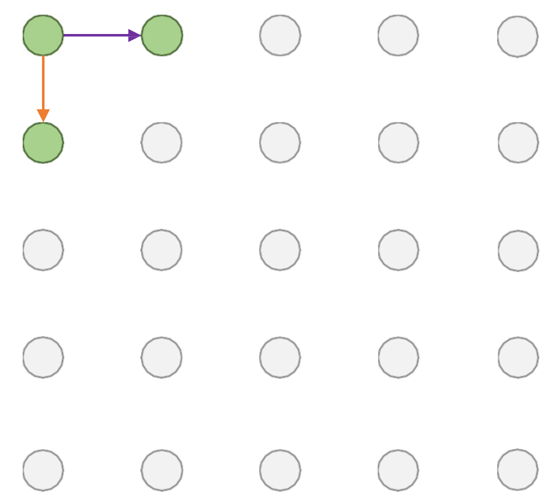
Van den Oord et al, "Pixel Recurrent Neural Networks", 2016

PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the pixel on the left and from the one above

$$\mathbf{h}_{\mathbf{x,y}} = f(\mathbf{h}_{\mathbf{x-1,y}}, \mathbf{h}_{\mathbf{x,y-1}}; \mathbf{W})$$



At each pixel, decode hidden state to red, then blue, then green
i.e., softmax over each channel intensity value [0, 1, …, 255] (256 categories)

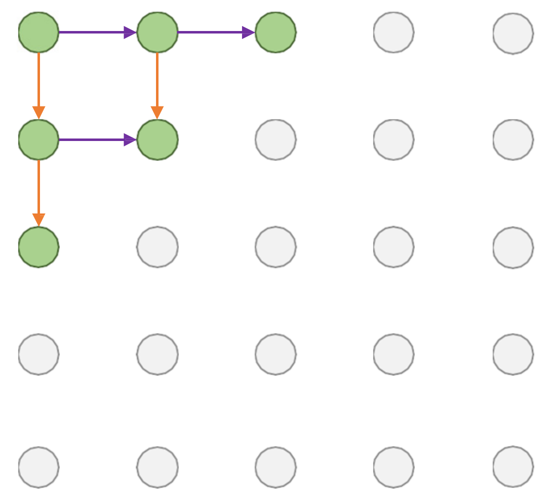
Van den Oord et al, "Pixel Recurrent Neural Networks", 2016

PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the pixel on the left and from the one above

$$\mathbf{h}_{x,y} = f(\mathbf{h}_{x-1,y}, \mathbf{h}_{x,y-1}; \mathbf{W})$$



At each pixel, decode hidden state to red, then blue, then green
i.e., softmax over each channel intensity value [0, 1, …, 255] (256 categories)

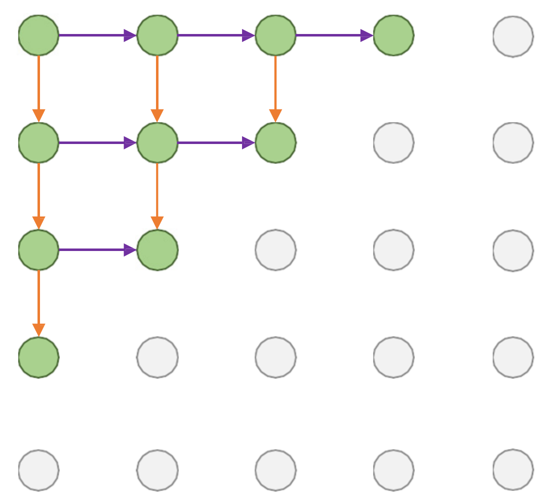
Van den Oord et al, "Pixel Recurrent Neural Networks", 2016

PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the pixel on the left and from the one above

$$\mathbf{h}_{x,y} = f(\mathbf{h}_{x-1,y}, \mathbf{h}_{x,y-1}; \mathbf{W})$$



At each pixel, decode hidden state to red, then blue, then green
i.e., softmax over each channel intensity value [0, 1, …, 255] (256 categories)

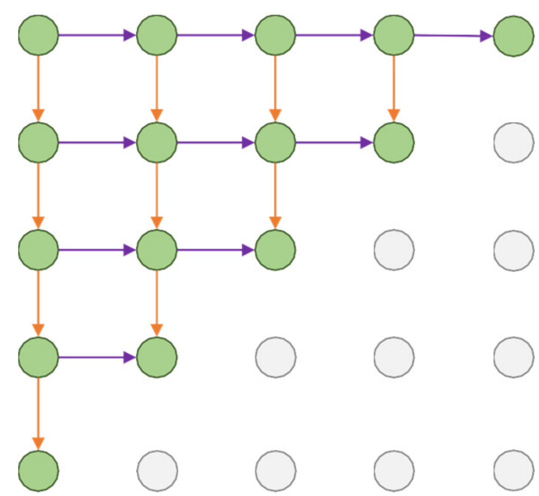
Van den Oord et al, "Pixel Recurrent Neural Networks", 2016

PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the pixel on the left and from the one above

$$\mathbf{h}_{\mathbf{x,y}} = f(\mathbf{h}_{\mathbf{x\text{-}1,y}}, \mathbf{h}_{\mathbf{x,y\text{-}1}}; \mathbf{W})$$



At each pixel, decode hidden state to red, then blue, then green
i.e., softmax over each channel intensity value [0, 1, …, 255] (256 categories)

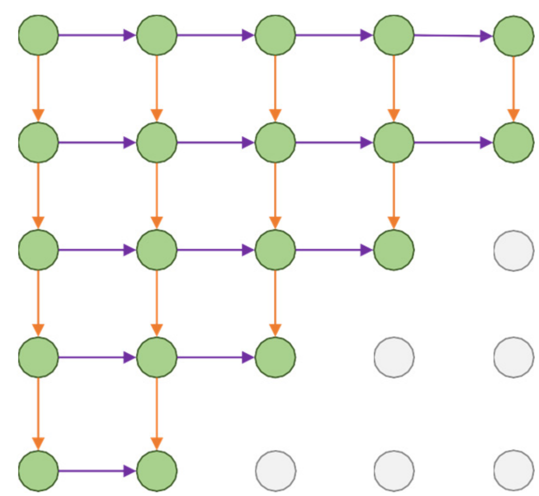
Van den Oord et al, "Pixel Recurrent Neural Networks", 2016

PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the pixel on the left and from the one above

$$\mathbf{h}_{x,y} = f(\mathbf{h}_{x-1,y}, \mathbf{h}_{x,y-1}; \mathbf{W})$$



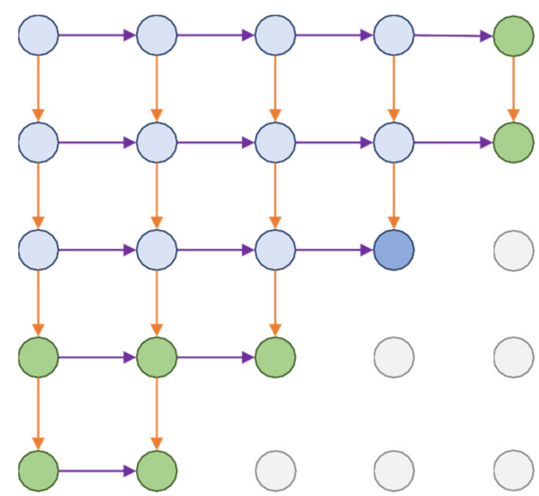
At each pixel, decode hidden state to red, then blue, then green
i.e., softmax over each channel intensity value [0, 1, …, 255] (256 categories)

Van den Oord et al, "Pixel Recurrent Neural Networks", 2016

PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Note that each pixel value is affected from all pixels above and to the left:

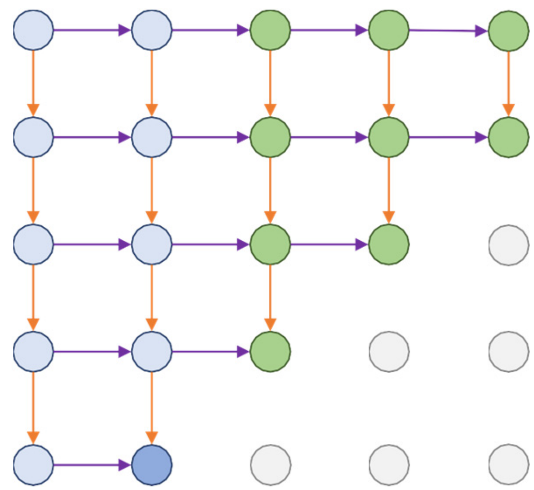


Van den Oord et al, "Pixel Recurrent Neural Networks", 2016

PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Note that each pixel value is affected from all pixels above and to the left:

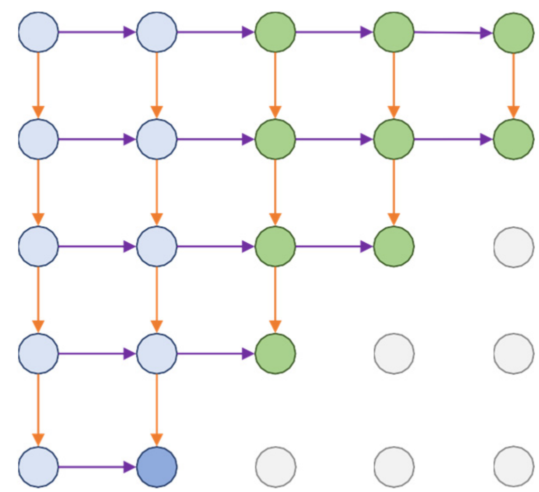


Van den Oord et al, “Pixel Recurrent Neural Networks”, 2016

PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Note that each pixel value is affected from all pixels above and to the left:



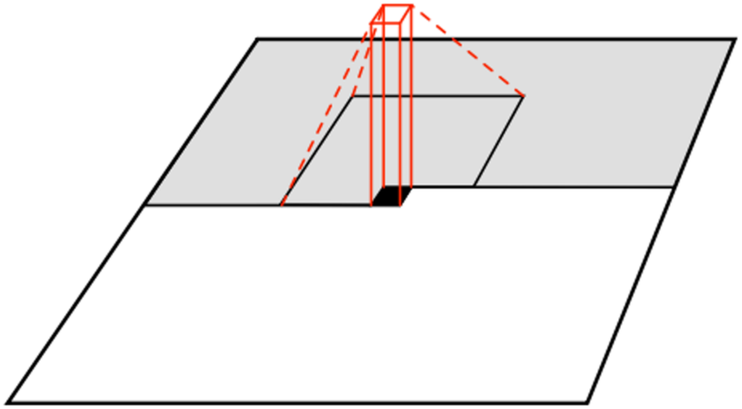
Problem: Very slow during both training and testing; N x N image generation requires lots of sequential steps

Van den Oord et al, "Pixel Recurrent Neural Networks", 2016

PixelCNN

Still generate image pixels starting from corner

Dependency on previous pixels is modeled using a convnet with **masked convolution** filters capturing a context region

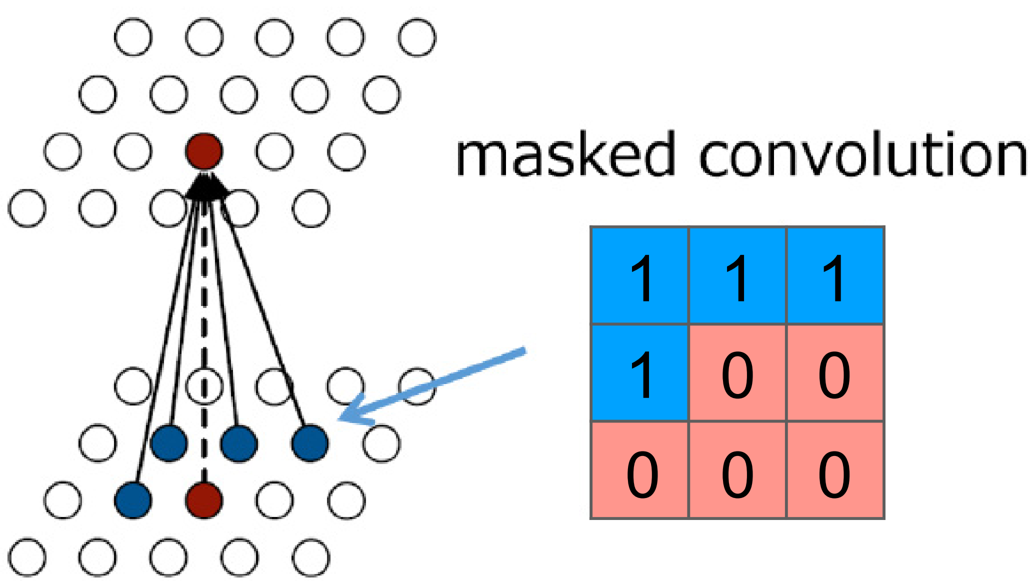


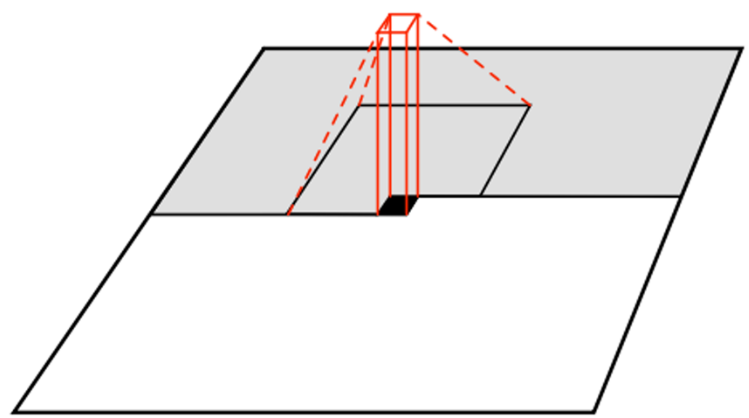
Van den Oord et al, "Conditional Image Generation with PixelCNN Decoders", NeurIPS 2016

PixelCNN

Still generate image pixels starting from corner

Dependency on previous pixels is modeled using a convnet with **masked convolution** filters capturing a context region





Van den Oord et al, “Conditional Image Generation with PixelCNN Decoders”, NeurIPS 2016

PixelCNN

Two types of masks

1	1	1
1	0	0
0	0	0

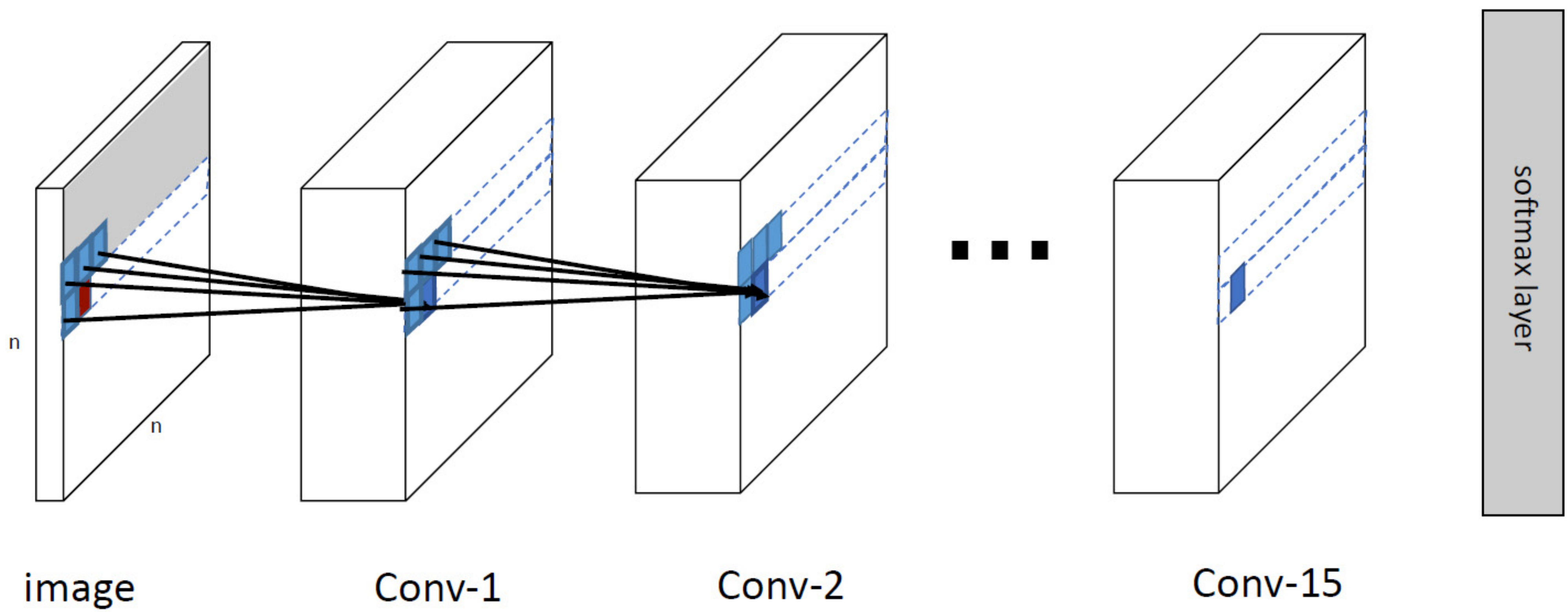
For the first layer
(connected to the input)

1	1	1
1	1	0
0	0	0

All other conv layers

Van den Oord et al, “Conditional Image Generation with PixelCNN Decoders”, NeurIPS 2016

PixelCNN

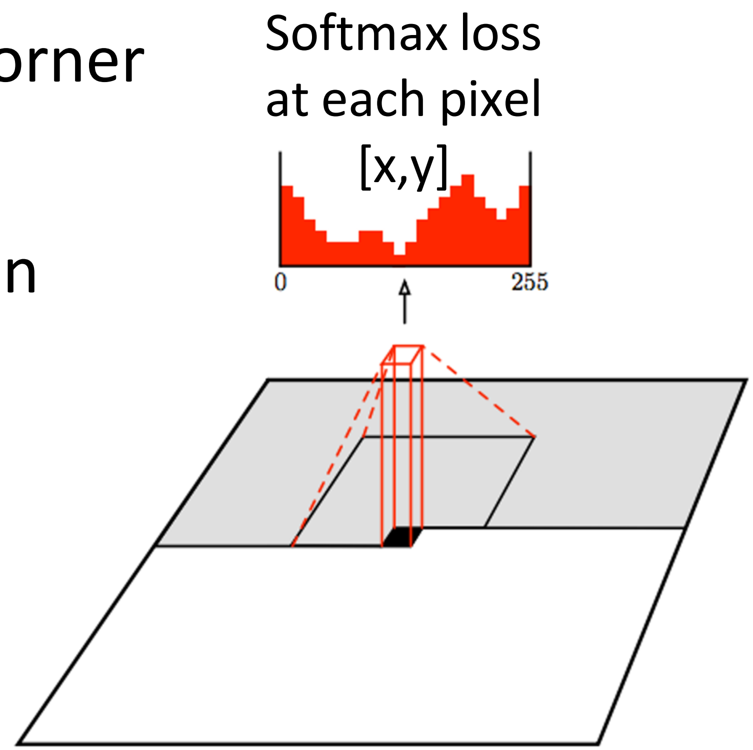


Van den Oord et al, "Conditional Image Generation with PixelCNN Decoders", NeurIPS 2016

PixelCNN

Still generate image pixels starting from corner

Output generates a probability distribution over pixel intensities [0,1,…,255]

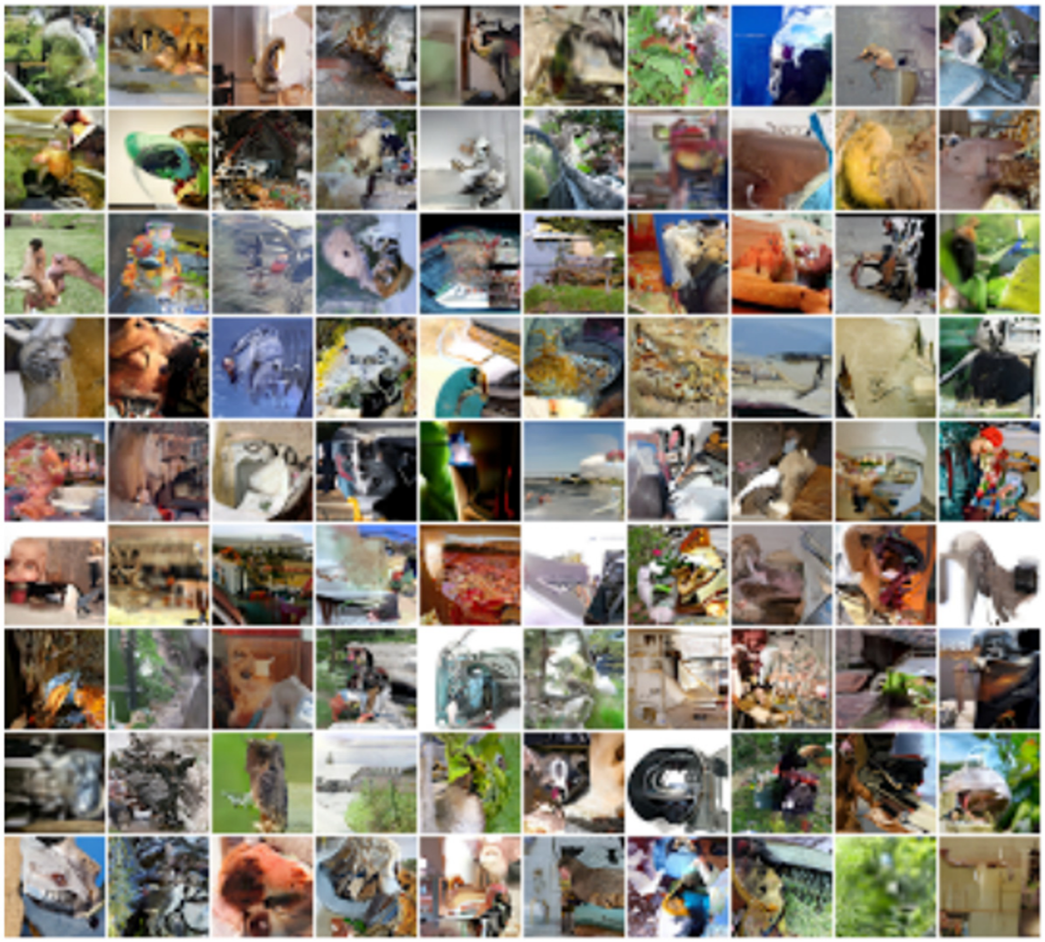


Van den Oord et al, “Conditional Image Generation with PixelCNN Decoders”, NeurIPS 2016

PixelCNN

Training is faster than PixelRNN (parallelize convolutions)

Generation must still proceed sequentially (slow) starting from top left

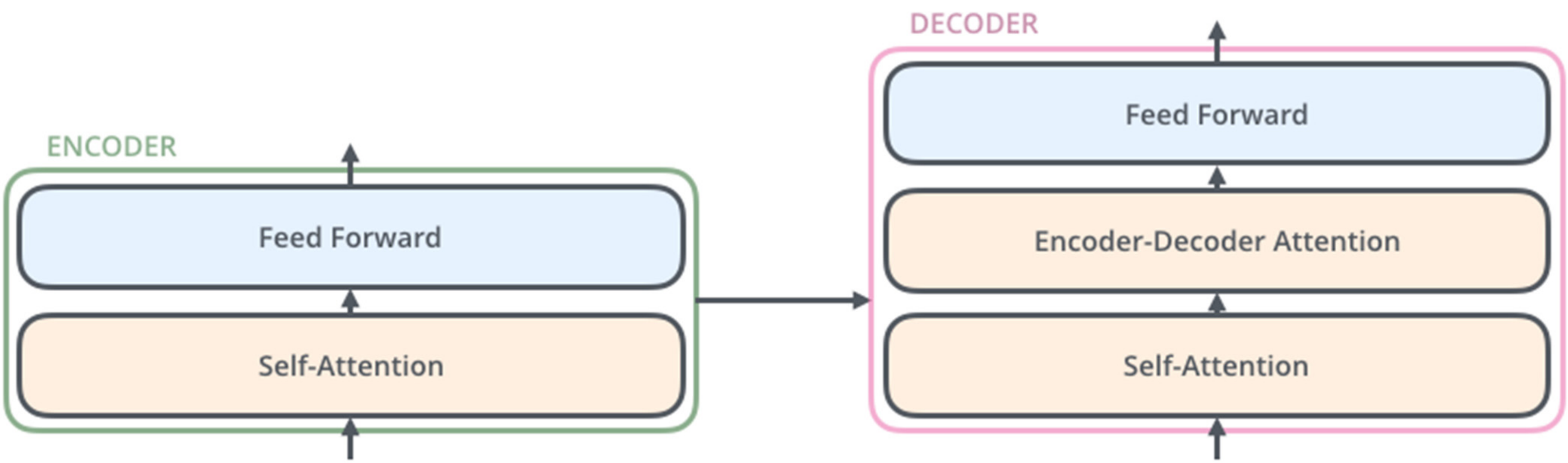


32x32 ImageNet

Van den Oord et al, "Conditional Image Generation with PixelCNN Decoders", NeurIPS 2016

Transformer Decoders

The decoder transformer can alternatively be used for auto-regressive prediction.
Its self-attention layer is only allowed to attend to earlier positions in the output sequence (also done by masking inputs)

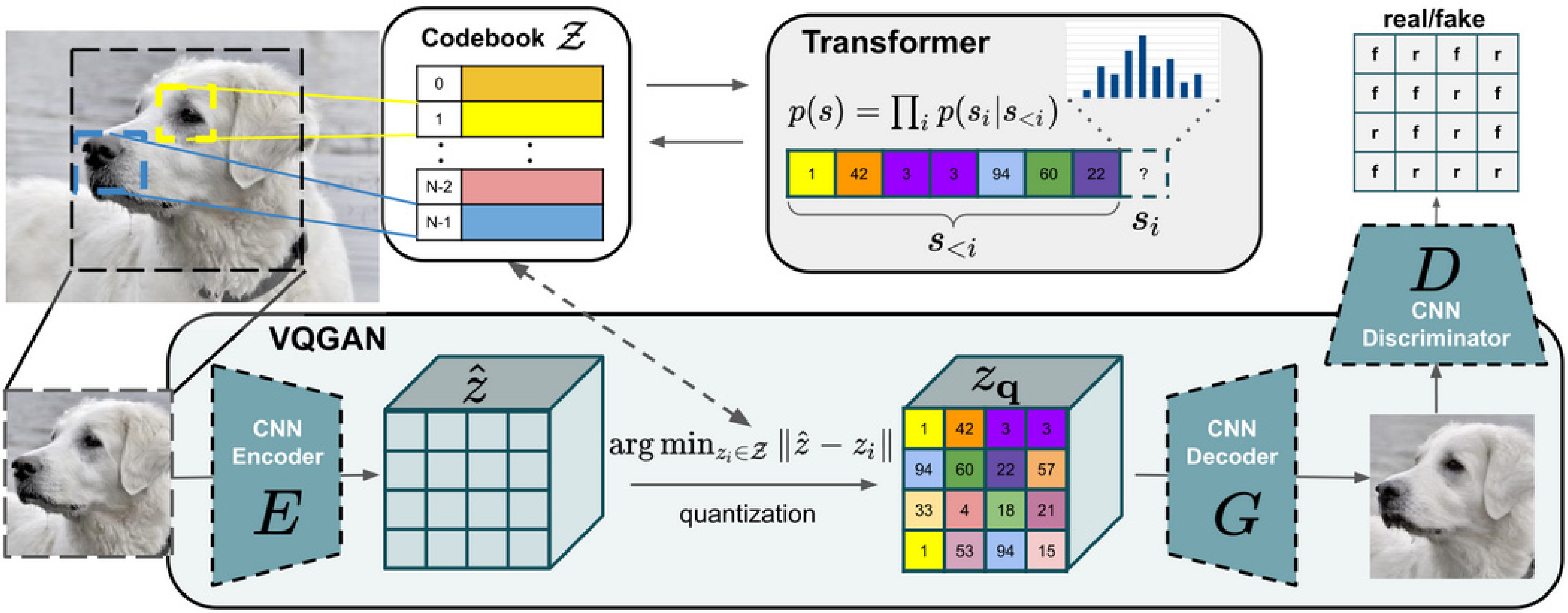


http://jalammar.github.io/illustrated-transformer/

How to generate visual data?

- Encoder-Decoders
- Generative Adversarial Networks
- **Autoregressive models**
  - PixelRNN / PixelCNN
  - **VQGAN**
  - PolyGen
- Variational Autoencoders
- Diffusion models

Autoregressive predictions of latents



Taming Transformers for High-Resolution Image Synthesis (VQGAN), 2021

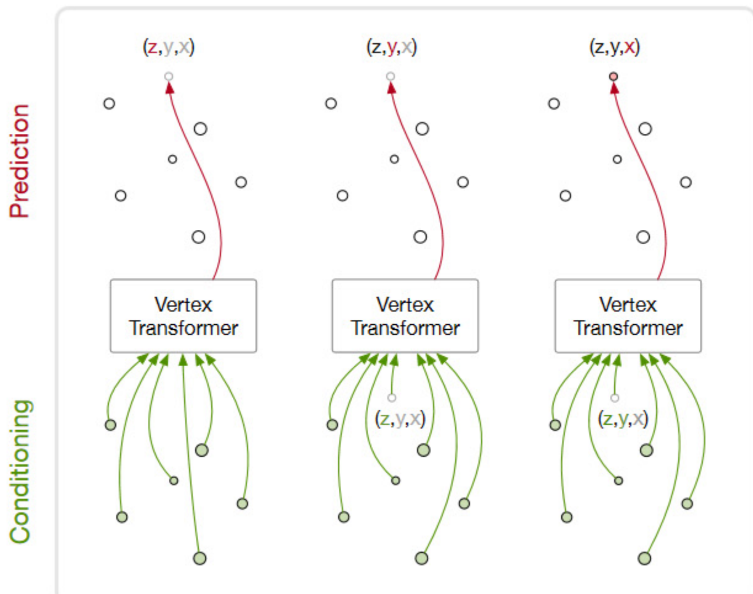
VQGAN result

How to generate visual data?

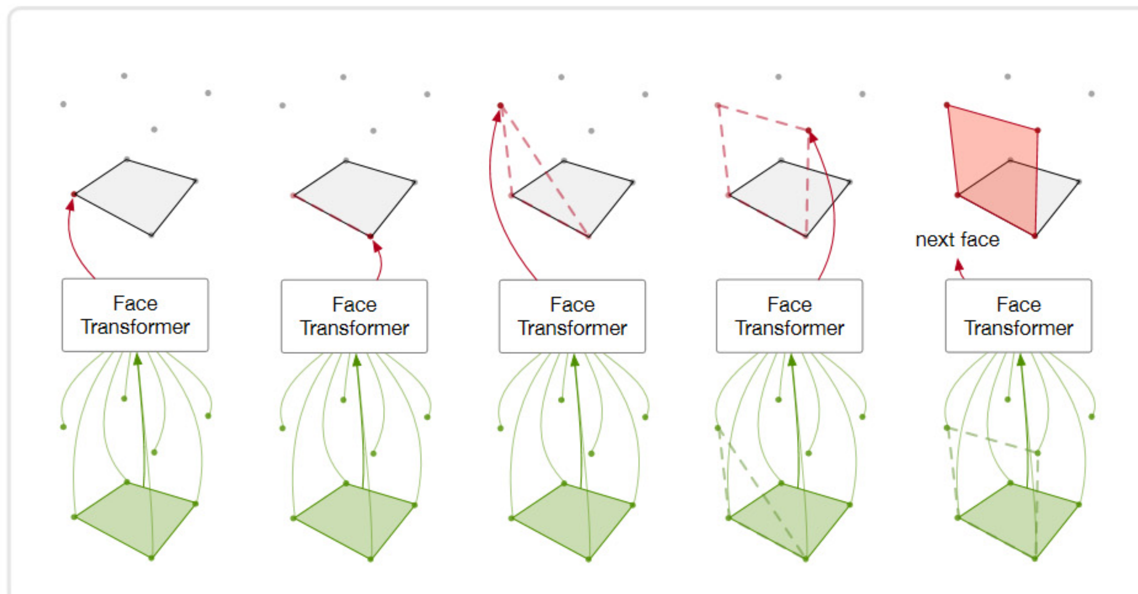
- Encoder-Decoders
- Generative Adversarial Networks
- **Autoregressive models**
  - PixelRNN / PixelCNN
  - VQGAN
  - **PolyGen**
- Variational Autoencoders
- Diffusion models

Transformers for 3D mesh generation

Generated vertices as an ordered list (ordered by lowest to highest z-coordinate), then generates faces conditioned on the generated points and previous faces



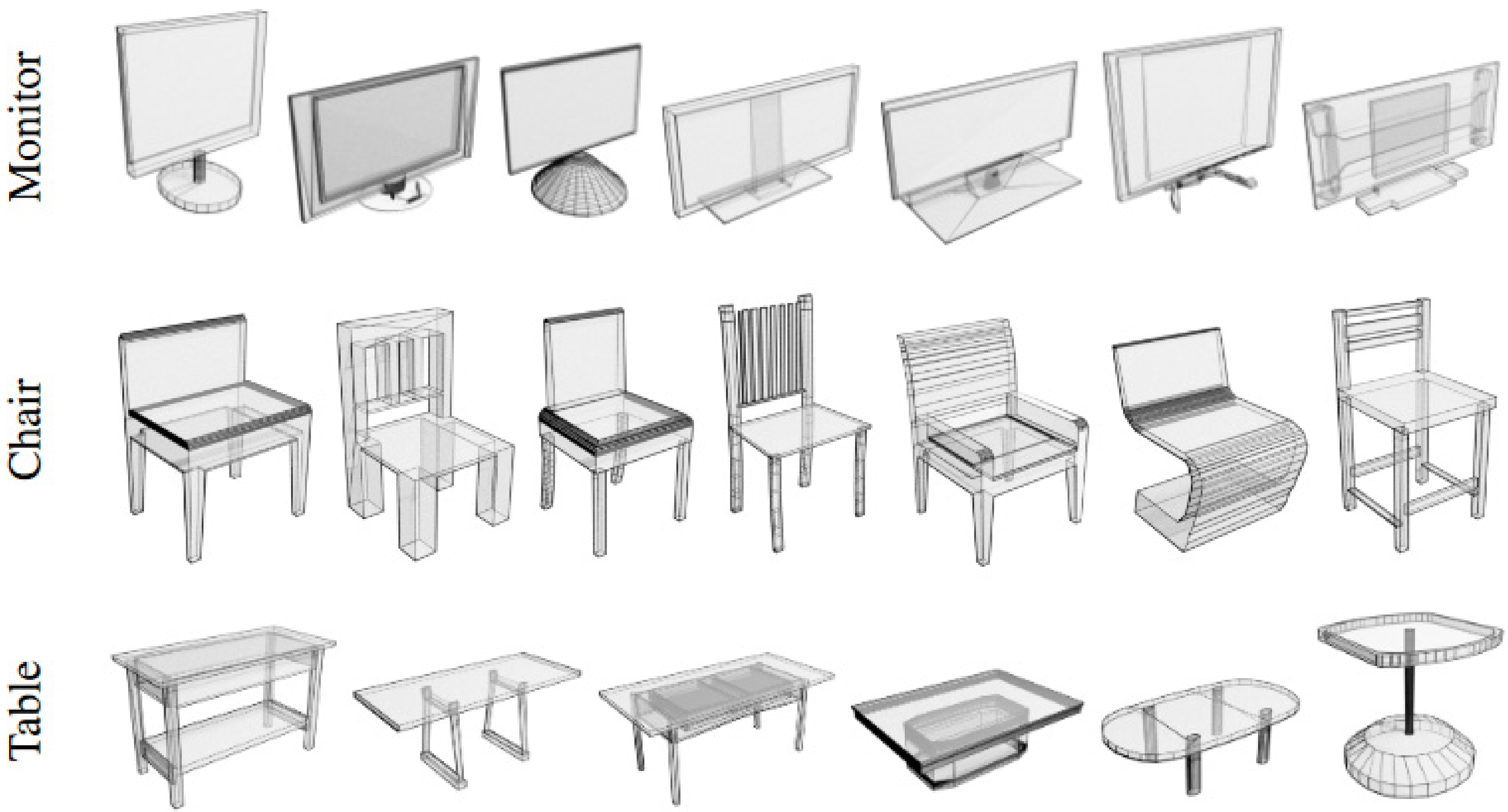
Vertex Model



Face Model

PolyGen: An Autoregressive Generative Model of 3D Meshes, ICML 2020

Generated Meshes



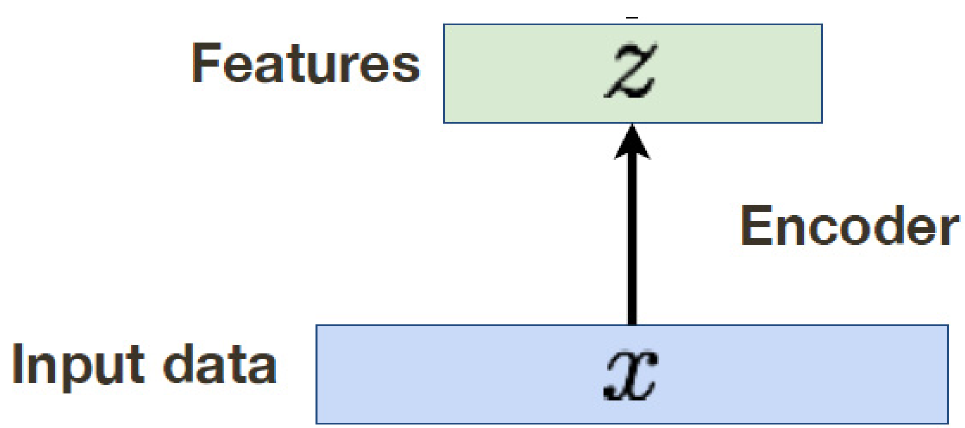
PolyGen: An Autoregressive Generative Model of 3D Meshes, ICML 2020

How to generate visual data?

- Encoder-Decoders
- Generative Adversarial Networks
- Autoregressive models
- **Variational Autoencoders**
- Diffusion models

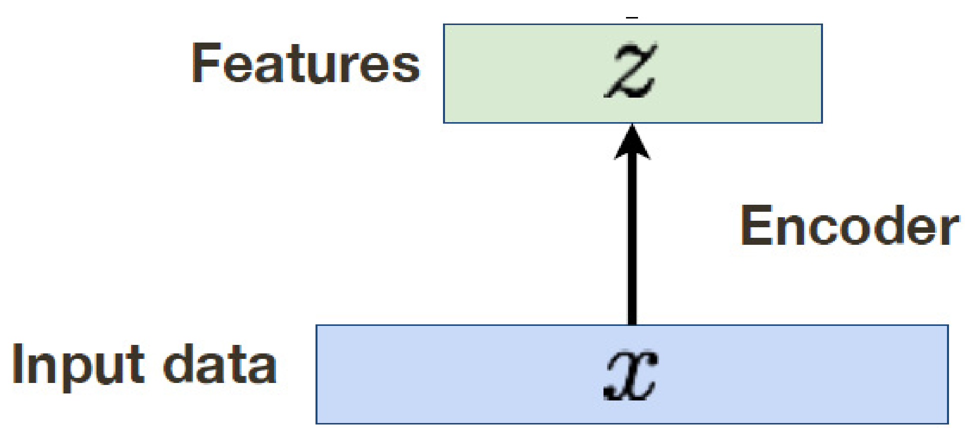
~~Variational~~ Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from **unlabeled** training data



~~Variational~~ Autoencoders

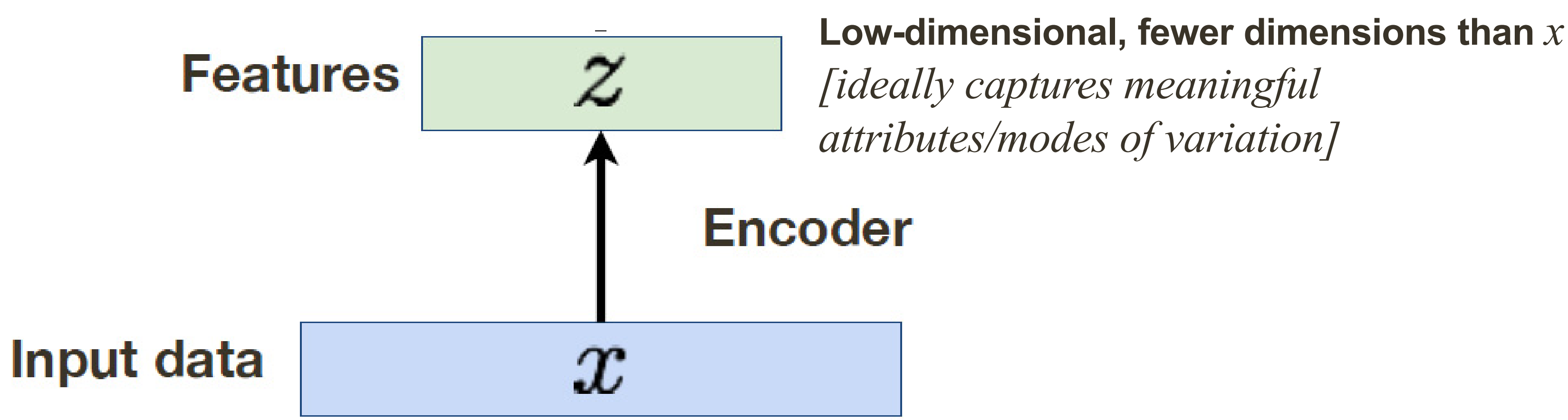
Unsupervised approach for learning a lower-dimensional feature representation from **unlabeled** training data



Originally: Linear + nonlinearity (sigmoid)
Later: Convnets, Transformers

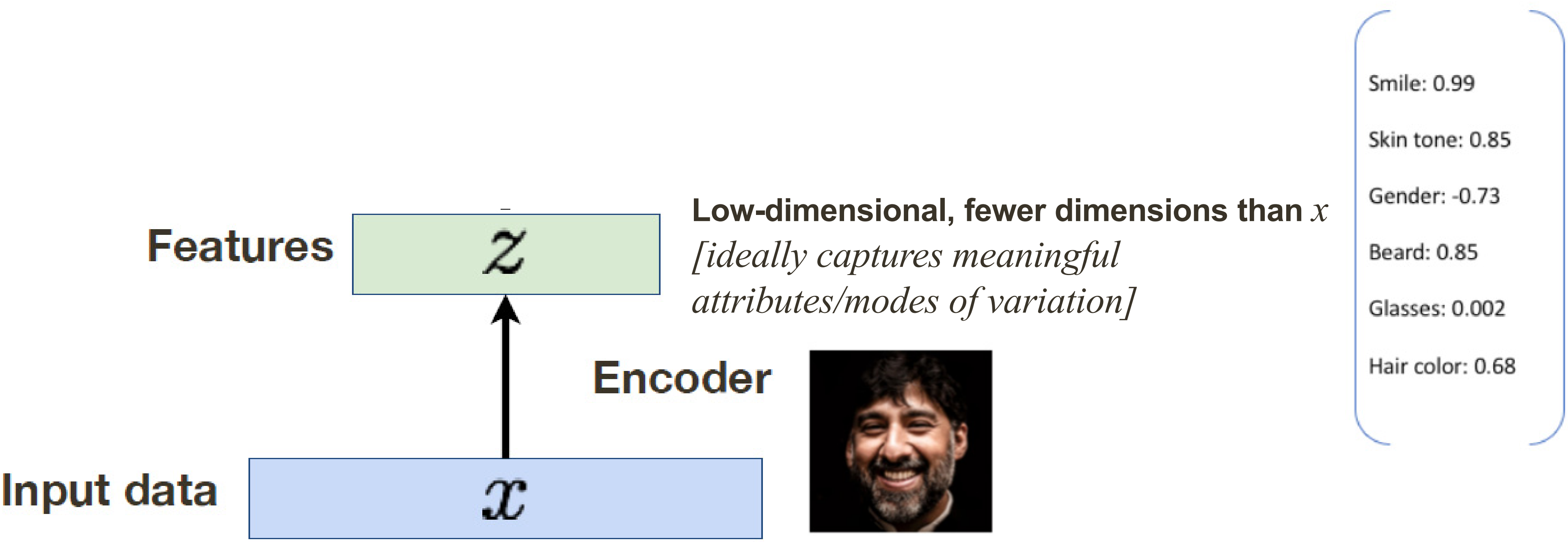
~~Variational~~ Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from **unlabeled** training data



~~Variational~~ Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from **unlabeled** training data

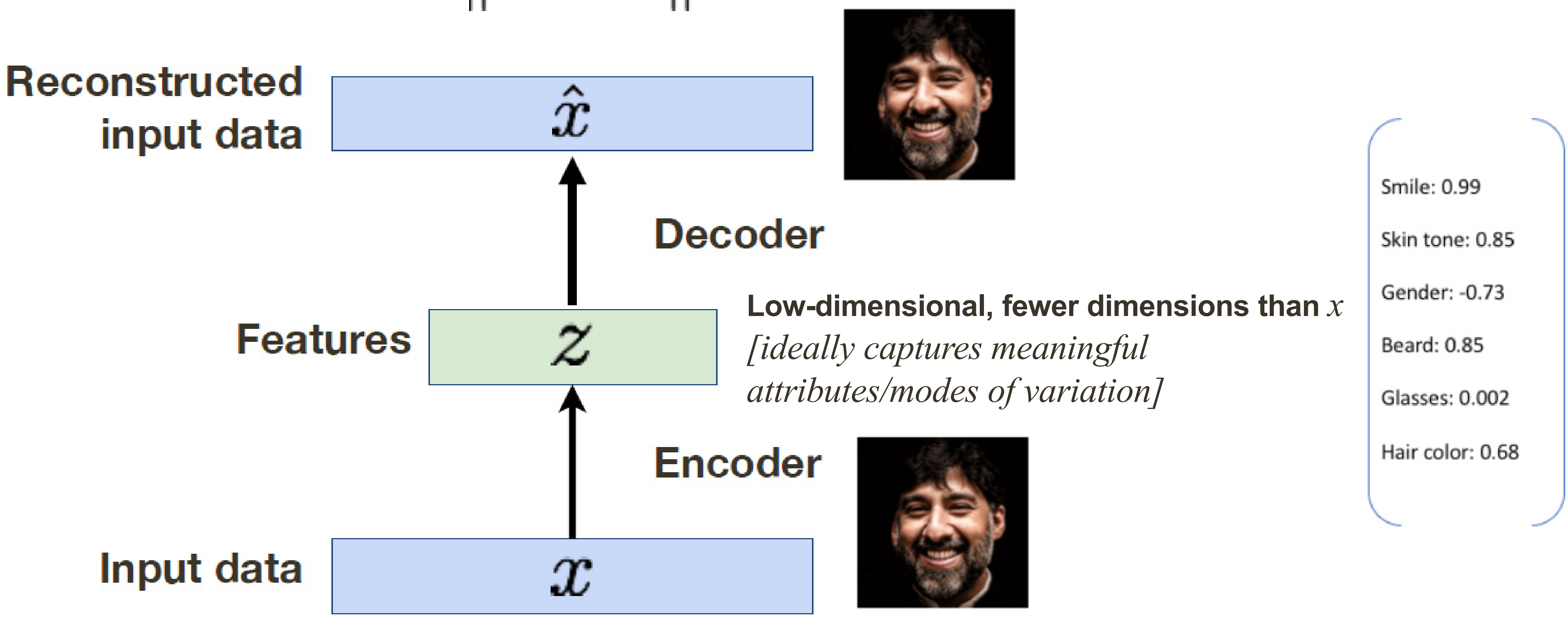


~~Variational~~ Autoencoders

Train such that features can reconstruct original data best they can!

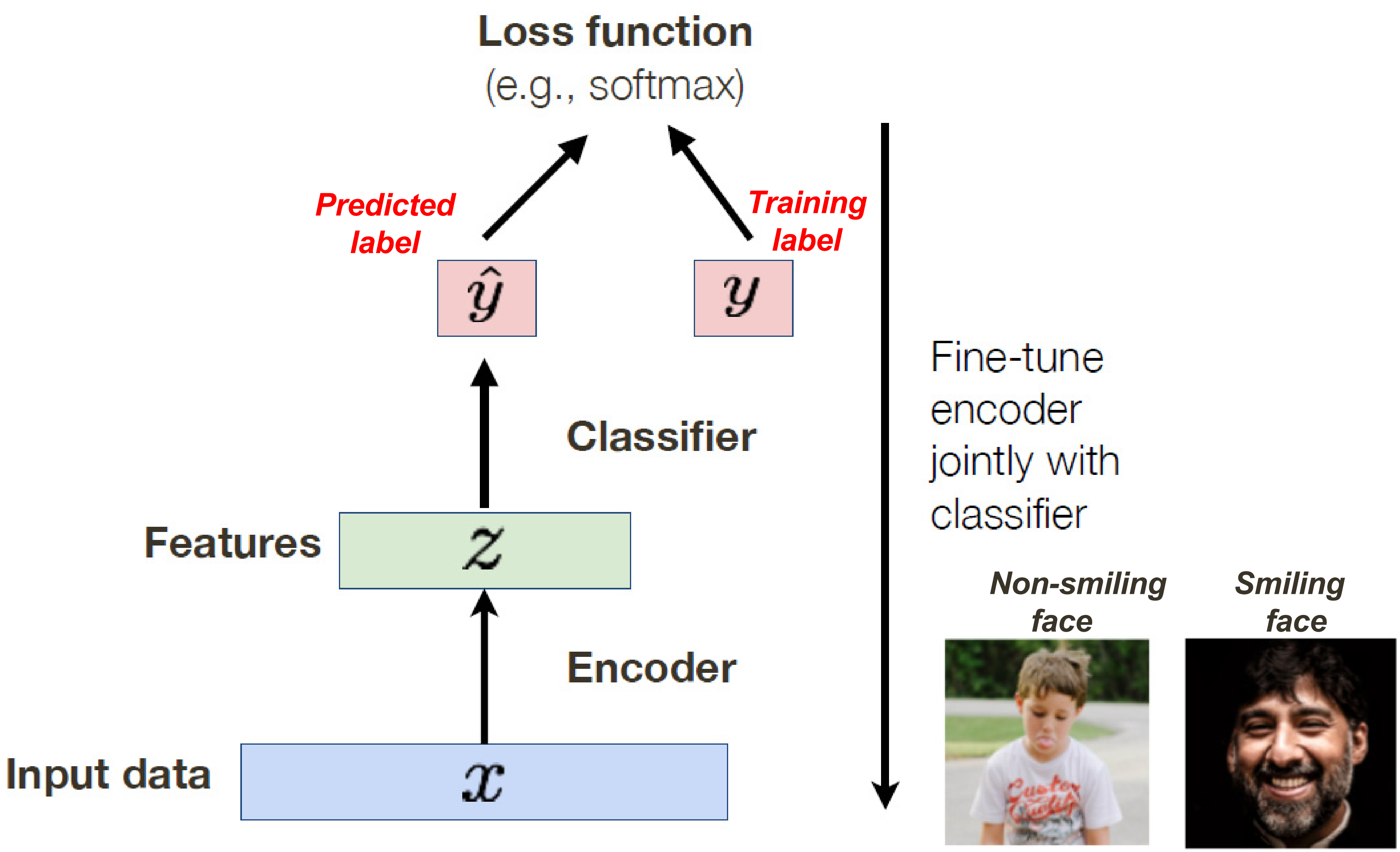
L2 Loss function:

$$\|x - \hat{x}\|^2$$

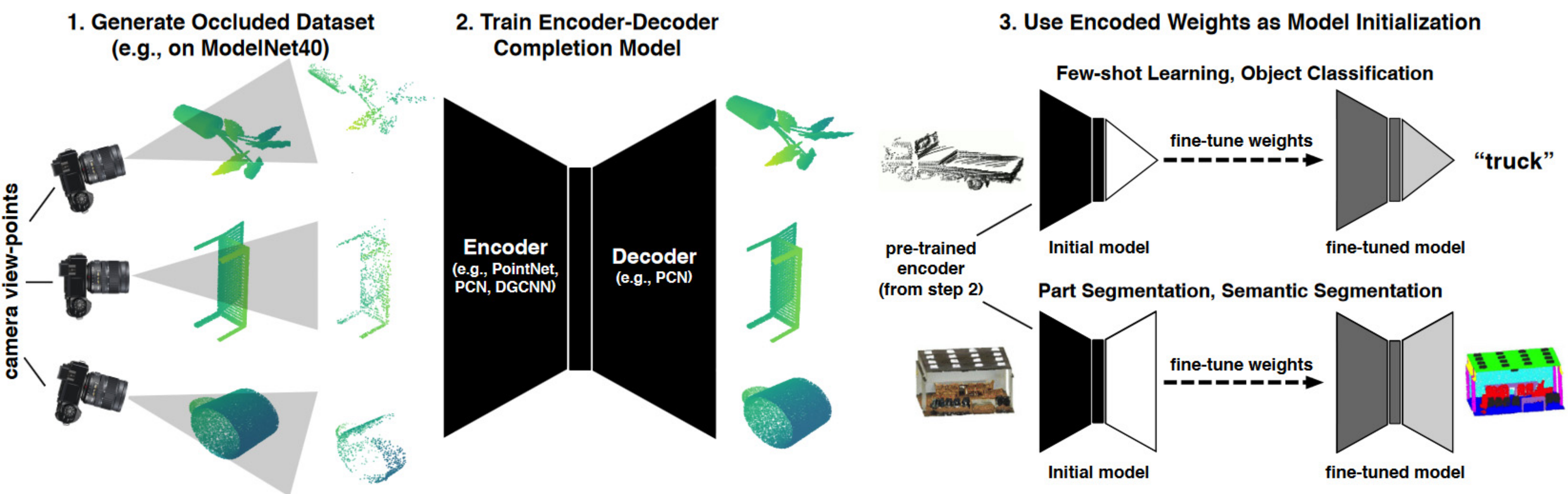


~~Variational~~ Autoencoders

After pre-training with a reconstruction loss, fine-tune encoder for a supervised task with **few amounts of data!**



Example: 3D point cloud pre-training



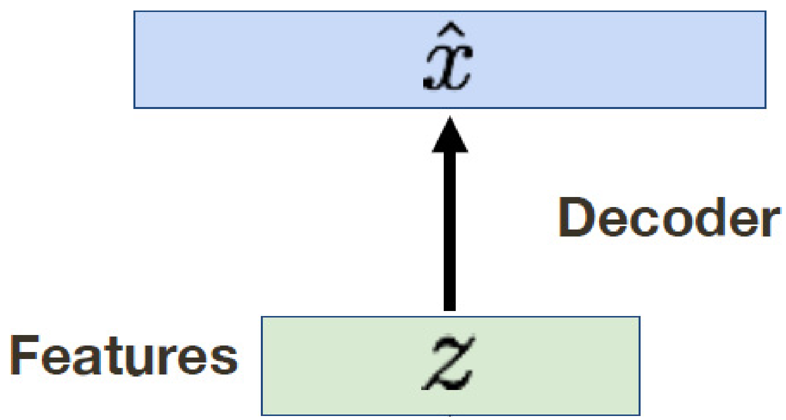
Unsupervised Point Cloud Pre-training via Occlusion Completion, ICCV 2021

Variational Autoencoders

Allow us to generate data!

Assume training data is generated from underlying unobserved **latent representation z**

At test time:



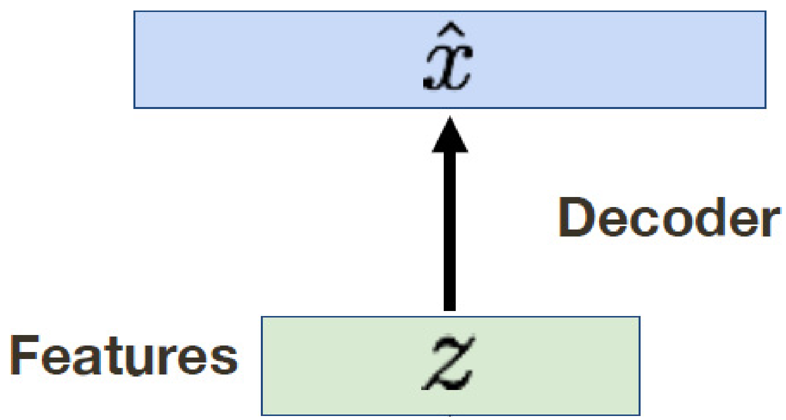
Sample from simple distribution
P(z)
(a Gaussian)

Variational Autoencoders

Allow us to generate data!

Assume training data is generated from underlying unobserved **latent representation z**

At test time:



Sample from complex cond. distribution
$P(x \mid z)$
(a neural network with learned param θ)

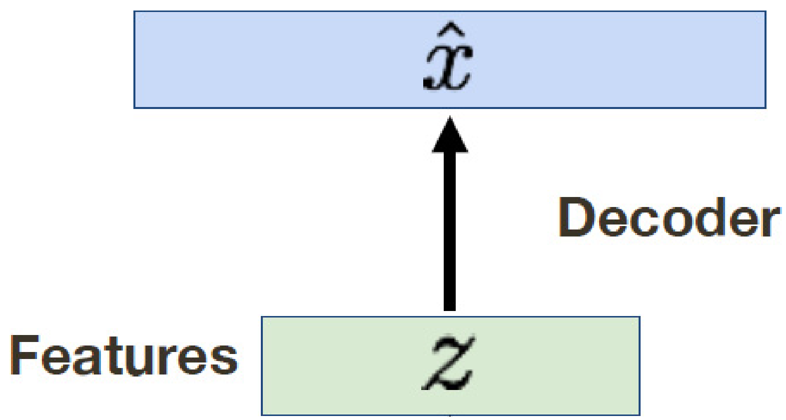
Sample from simple distribution
$P(z)$
(a Gaussian)

Variational Autoencoders

How to train this model?

Maximum likelihood:

$$p_\theta(x) = \int_z p_\theta(z) p_\theta(x|z) dz$$



Sample from complex cond. distribution
$P(x \mid z)$
(a neural network with learned param θ)

Sample from simple distribution
$P(z)$
(a Gaussian)

Variational Autoencoders

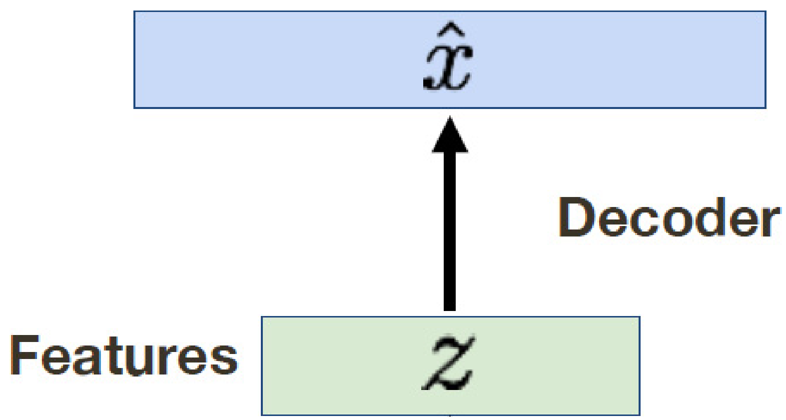
How to train this model?

Maximum likelihood:

$$p_\theta(x) = \int_z \boxed{p_\theta(z)} p_\theta(x|z)dz$$

🙂

Simple **Gaussian** Prior



Sample from complex cond. distribution
$P(x \mid z)$
(a neural network with learned param θ)

Sample from simple distribution
$P(z)$
(a Gaussian)

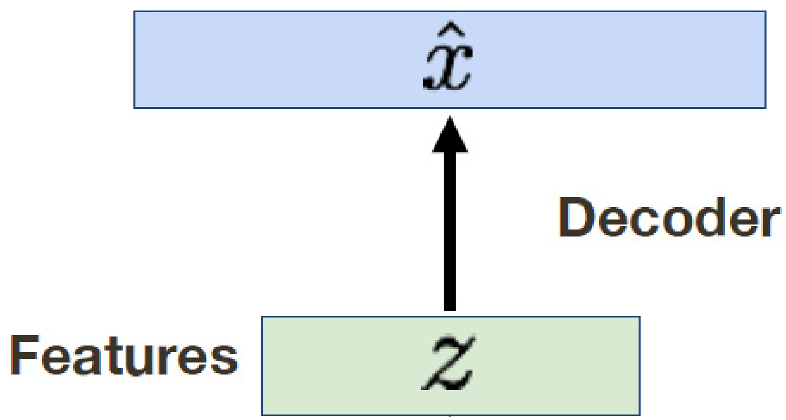
Variational Autoencoders

Decoder Neural Network

How to train this model?

Maximum likelihood:

$$p_\theta(x) = \int_z p_\theta(z)\boxed{p_\theta(x|z)}dz$$



Sample from complex cond. distribution
$P(x \mid z)$
(a neural network with learned param θ)

Sample from simple distribution
$P(z)$
(a Gaussian)

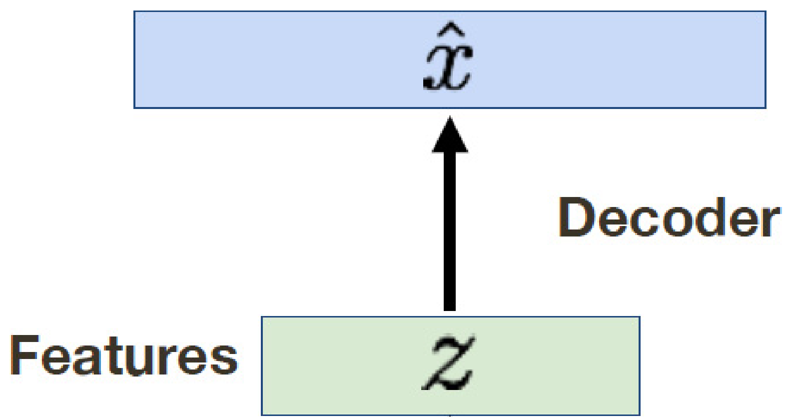
Variational Autoencoders

How to train this model?

Maximum likelihood:

$$p_\theta(x) = \boxed{\int_z} p_\theta(z) p_\theta(x|z) dz$$

Intractable to compute for every z ☹



Sample from complex cond. distribution
$P(x \mid z)$
(a neural network with learned param θ)

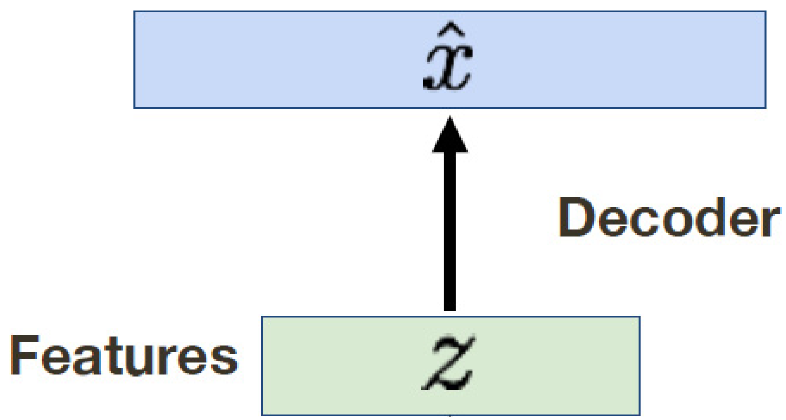
Sample from simple distribution
$P(z)$
(a Gaussian)

Variational Autoencoders

How to train this model?

Maximum likelihood: $$p_\theta(x) = \boxed{\int_z} p_\theta(z) p_\theta(x|z) dz$$

Intractable to compute for every z



Sample from complex cond. distribution
$P(x \mid z)$
(a neural network with learned param θ)

Sample from simple distribution
$P(z)$
(a Gaussian)

Posterior density is also intractable: $p_\theta(z|x) = p_\theta(x|z)p_\theta(z) / \boxed{p_\theta(x)}$

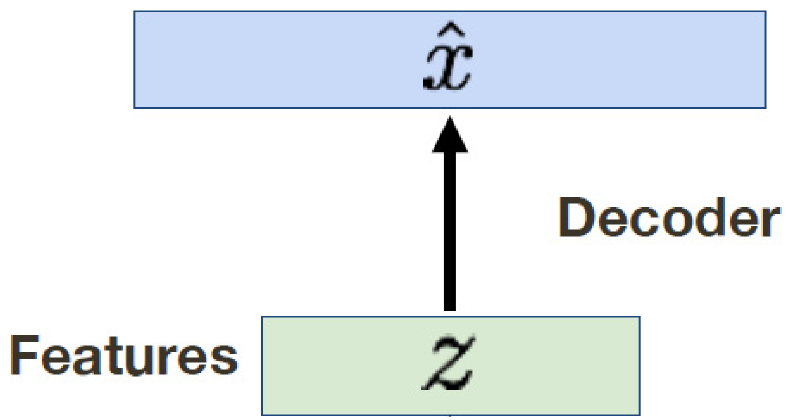
Variational Autoencoders

How to train this model?

Maximum likelihood:

$$p_\theta(x) = \boxed{\int_z} p_\theta(z)p_\theta(x|z)dz$$

Intractable to compute for every z ☹



Sample from complex cond. distribution
$P(x \mid z)$
(a neural network with learned param θ)

Sample from simple distribution
$P(z)$
(a Gaussian)

Solution: approximate $p_\theta(z \mid x)$ with a tractable distribution $q_\varphi(z \mid x)$

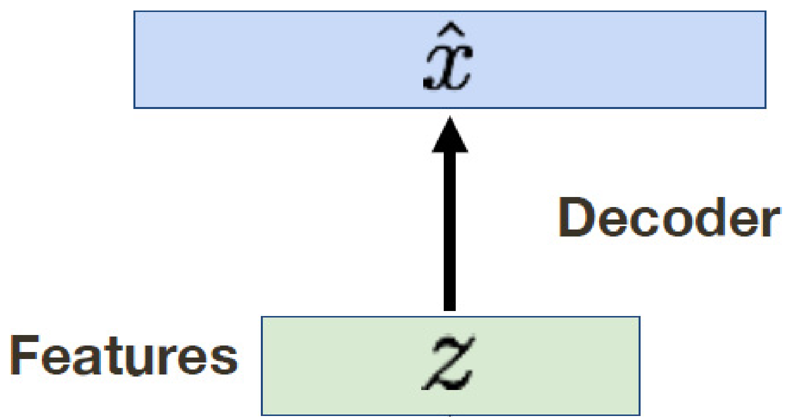
Posterior density is also intractable: $p_\theta(z|x) = p_\theta(x|z)p_\theta(z)/p_\theta(x)$

Variational Autoencoders

How to train this model?

Maximum likelihood:
$$p_\theta(x) = \boxed{\int_z} p_\theta(z)p_\theta(x|z)dz$$

Intractable to compute for every z



Sample from complex cond. distribution
$P(x \mid z)$
(a neural network with learned param θ)

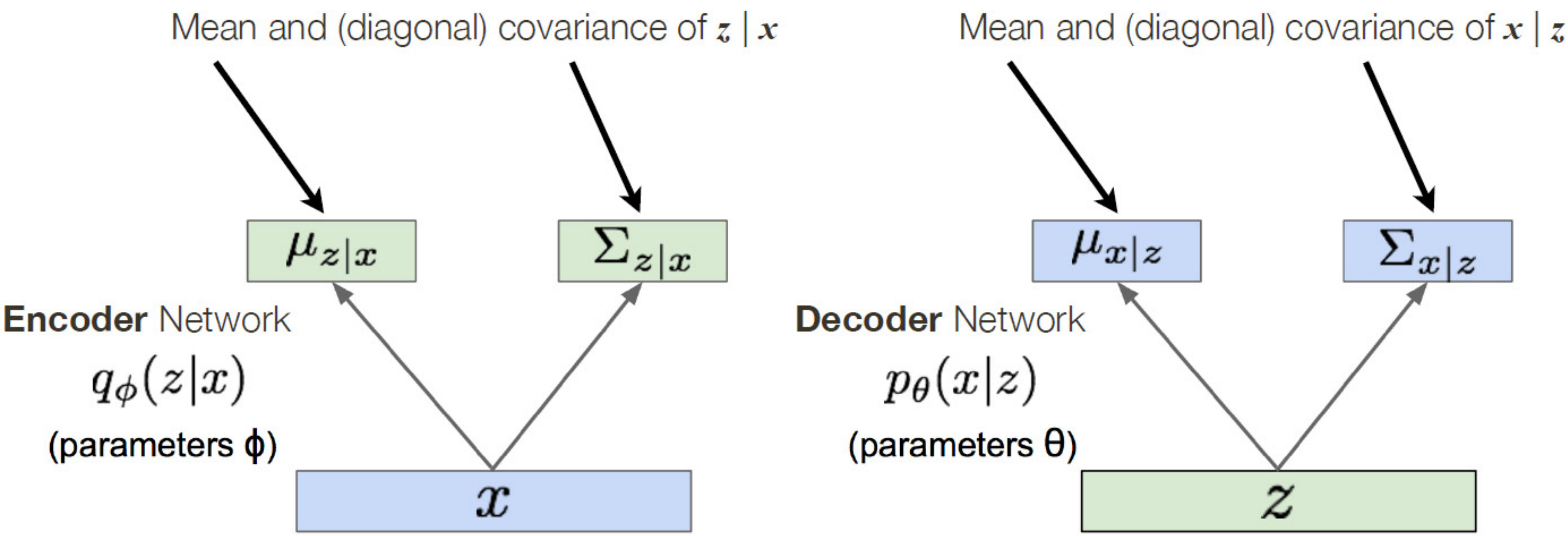
Sample from simple distribution
$P(z)$
(a Gaussian)

Solution: approximate $p_\theta(z \mid x)$ with a **neural network** $q_\varphi(z \mid x)$ **[encoder]**

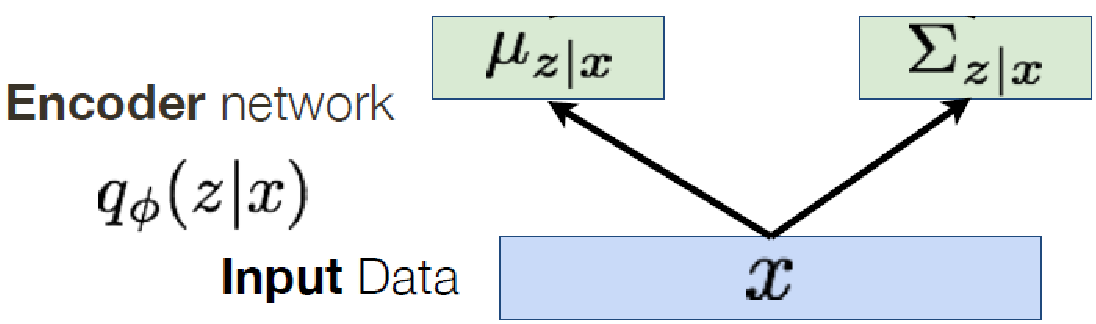
Posterior density is also intractable: $p_\theta(z|x) = p_\theta(x|z)p_\theta(z)/p_\theta(x)$

Variational Autoencoders

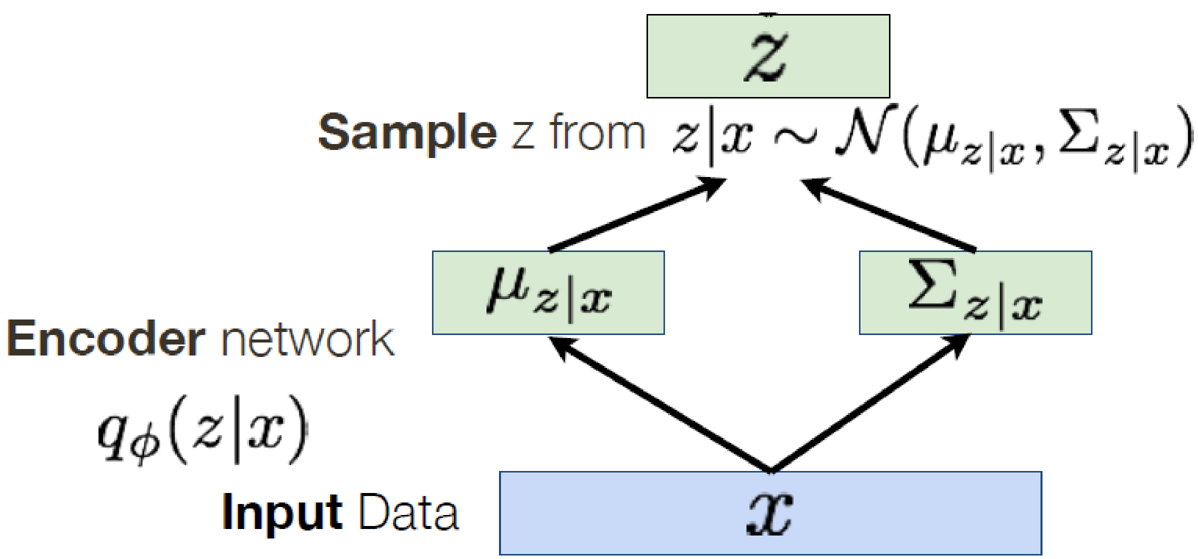
Since we are modeling probabilistic generation of data, encoder and decoder networks are probabilistic (they model Gaussian distributions)



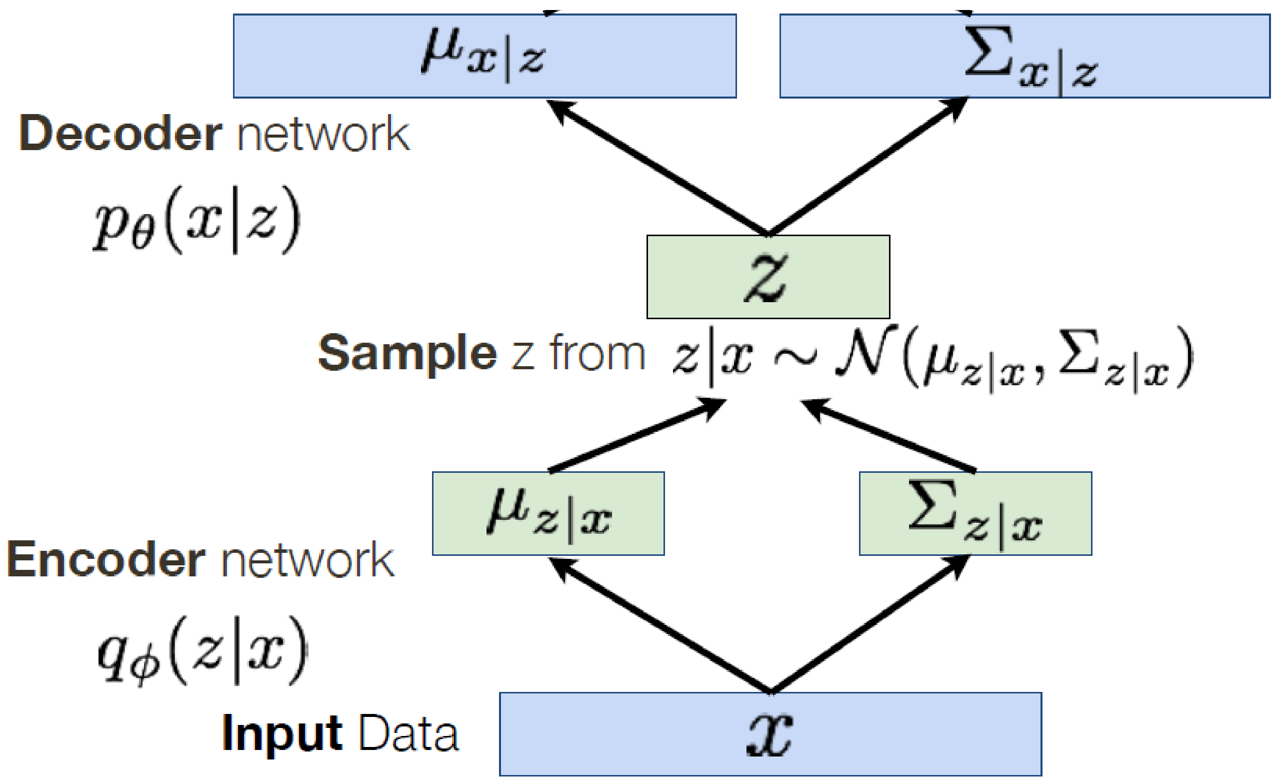
Forward pass during training



Forward pass during training

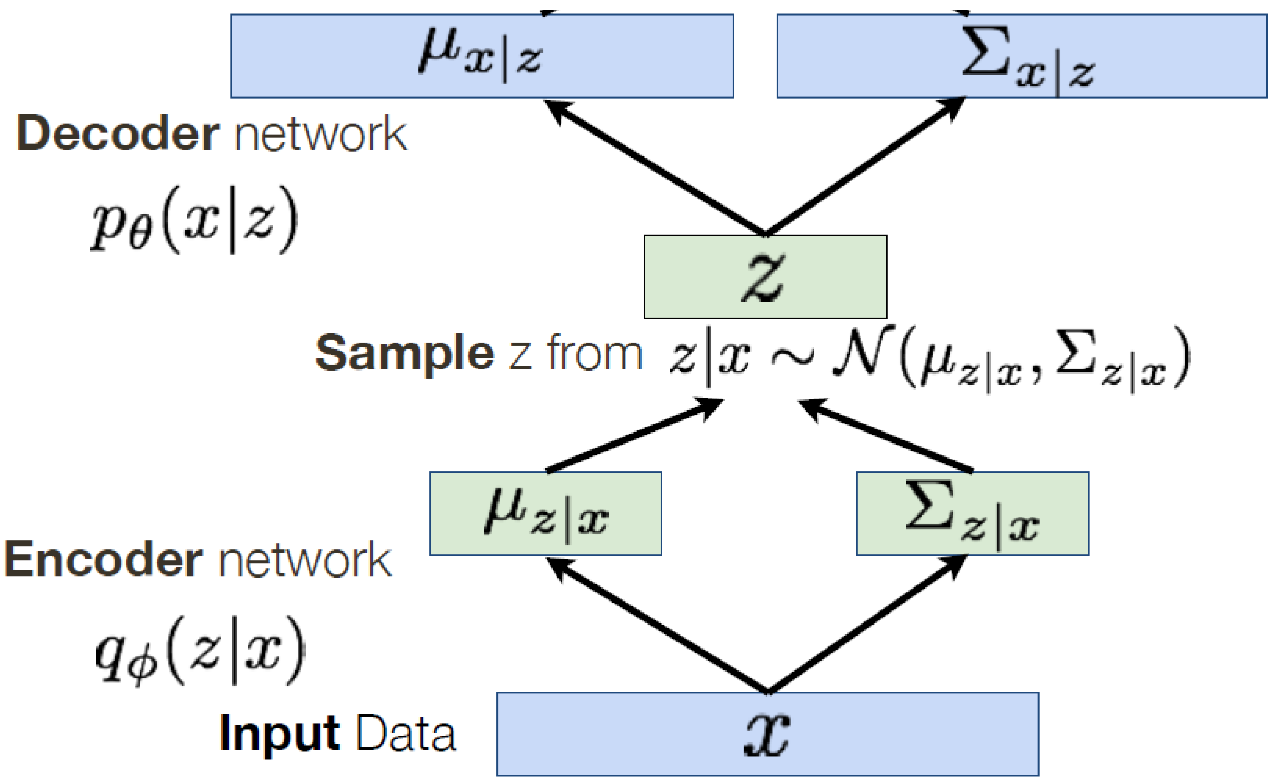


Forward pass during training



VAE Loss

$$-\mathbf{E}_{z\sim q_\phi(z|x)} \log\left(p_\theta\left(x|z\right)\right) + KL\left(q_\phi\left(z|x\right) \| p_\theta(z)\right)$$



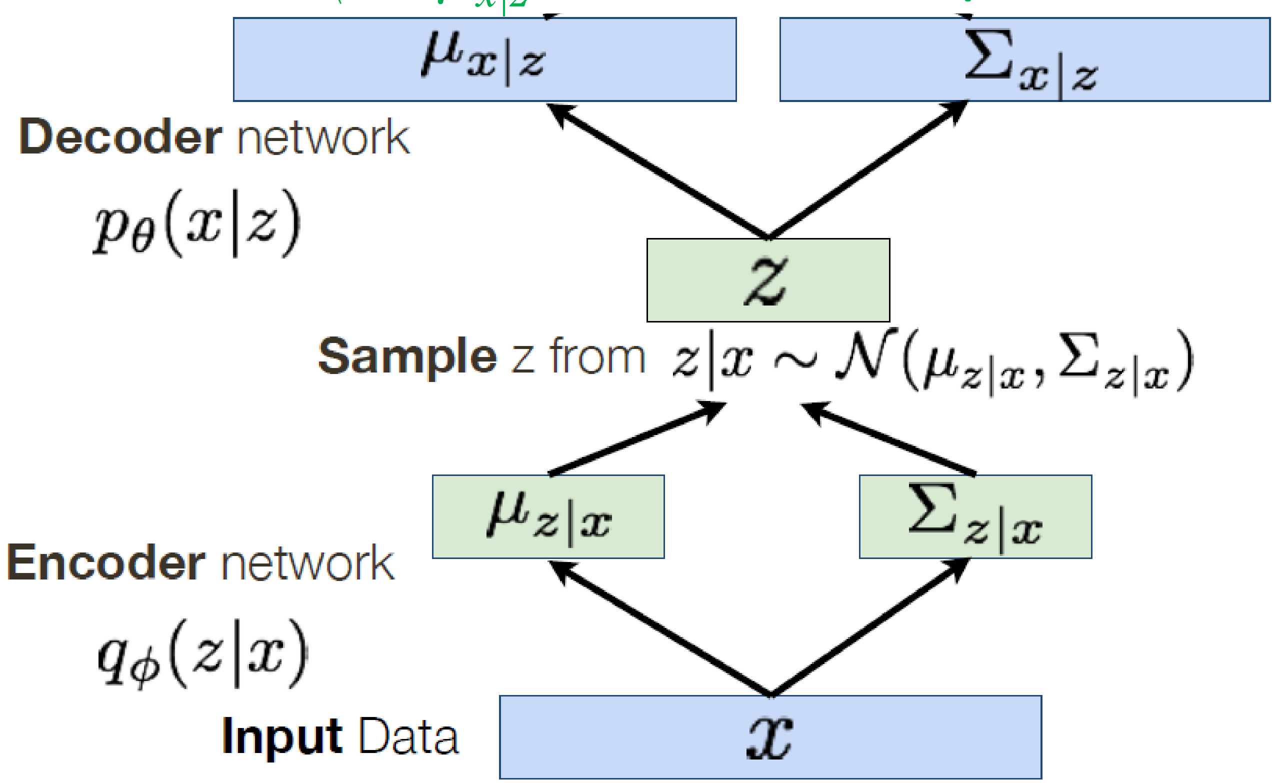
VAE Loss

$$\boxed{-\mathbf{E}_{z\sim q_\phi(z|x)} \log\left(p_\theta\left(x|z\right)\right)} + KL\left(q_\phi\left(z|x\right)\|\,p_\theta(z)\right)$$

Reconstruction Loss

(outputs should be as close as possible to input)

reduces to $(x - \mu_{x|z})^2$ for fixed output covariance

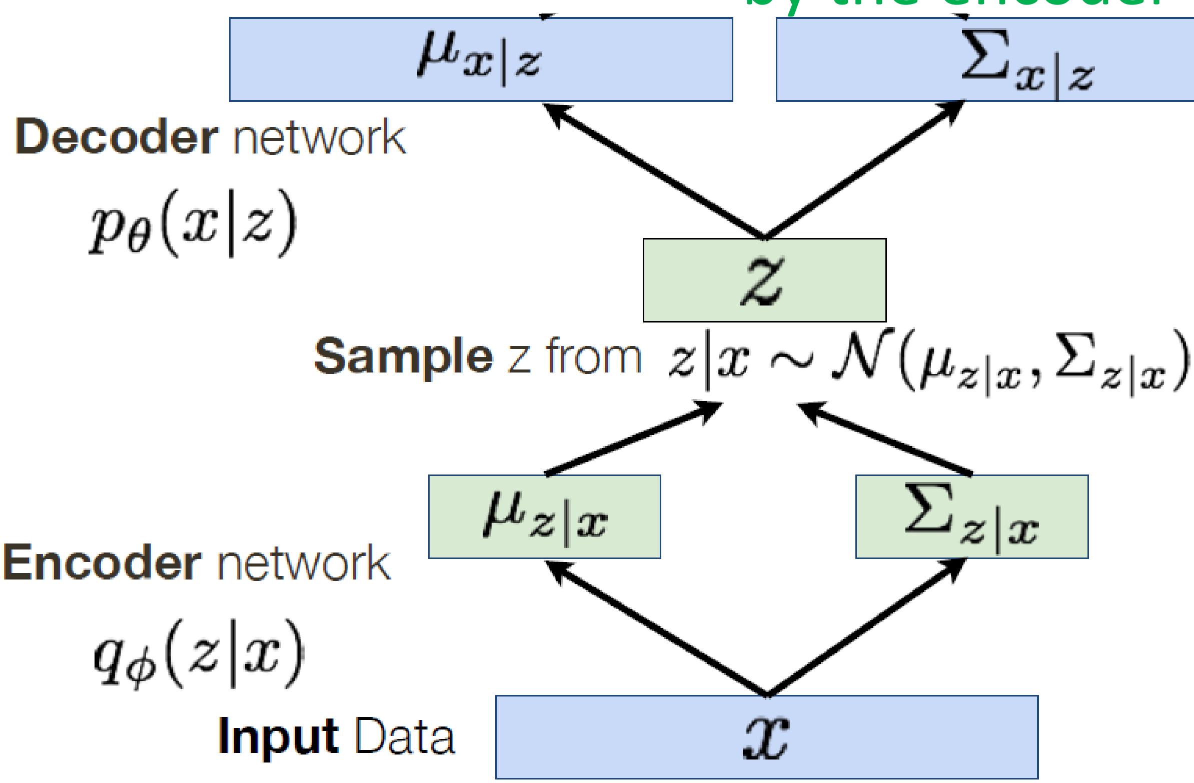


VAE Loss

$$-\mathbf{E}_{z\sim q_\phi(z|x)}\log\left(p_\theta\left(x|z\right)\right)+\boxed{KL\left(q_\phi\left(z|x\right)\|p_\theta(z)\right)}$$

Regularization term

Make distribution of the latent space produced by the encoder close to a standard Gaussian.



KL divergence

A measure of how one probability distribution is different from a second:

$$KL\left(q_\phi\left(z|x\right)\|\,p_\theta(z)\right)=\int_z q_\phi\left(z|x\right)\log\frac{q_\phi\left(z|x\right)}{p_\theta(z)}$$

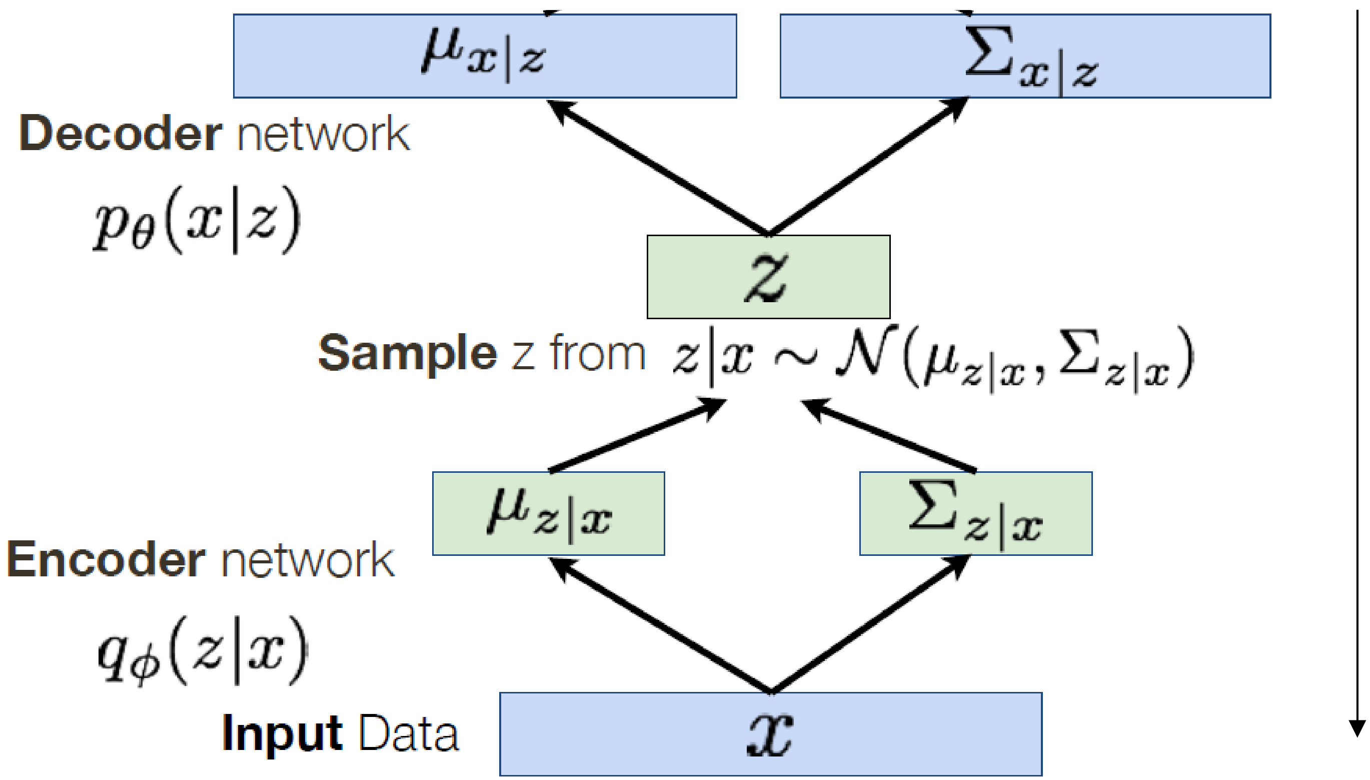
In our case, we want our latent space $p_\theta(z)$ to be $N(0, I)$

VAE Loss

$$-\mathbf{E}_{z\sim q_\phi(z|x)}\log\left(p_\theta\left(x|z\right)\right) + KL\left(q_\phi\left(z|x\right)\|p_\theta(z)\right)$$

$$\lambda\,(x-\mu_{x|z})^2 + \sum_{d=1}^{D}(\sigma^2_{z|x}[d]+\mu^2_{z|x}[d]-\log\sigma_{z|x}[d]-1)$$

(where λ is a weighting term)



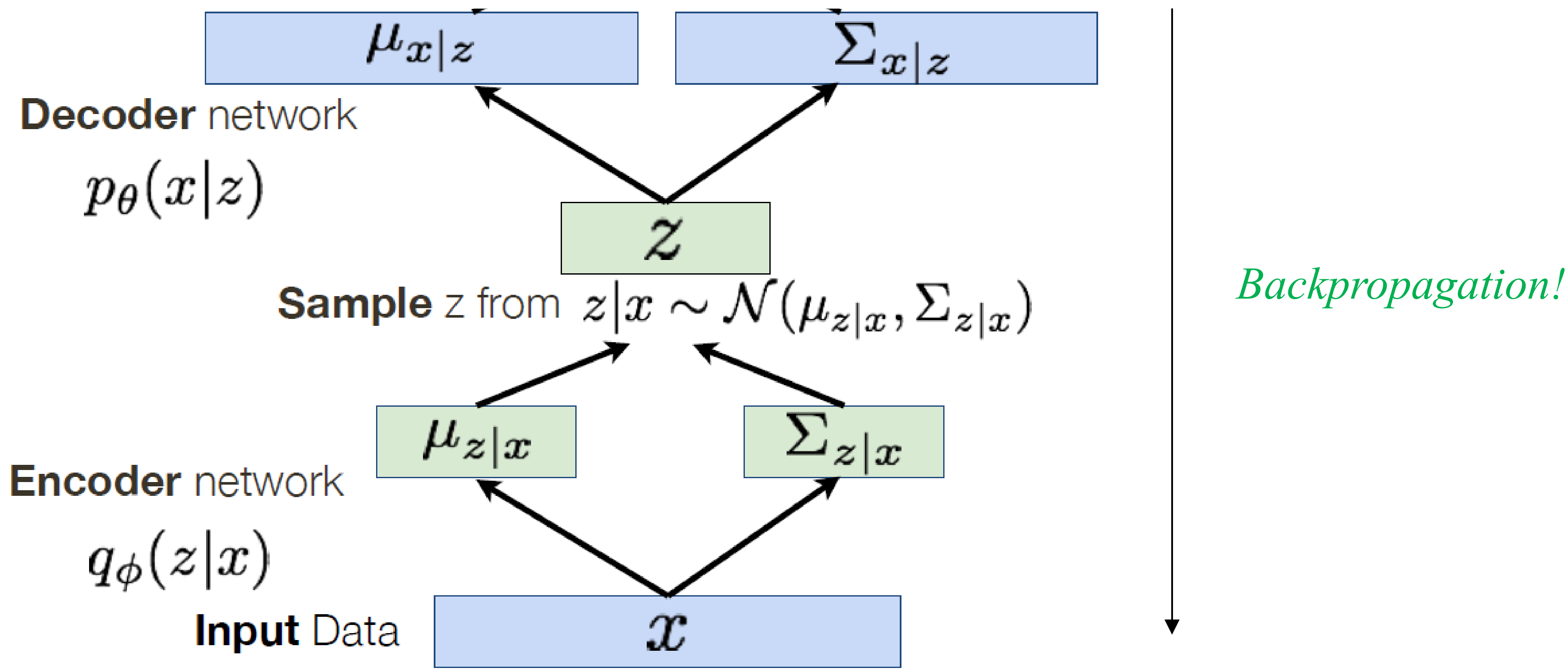
Backpropagation!

VAE Loss (skipping proofs...)

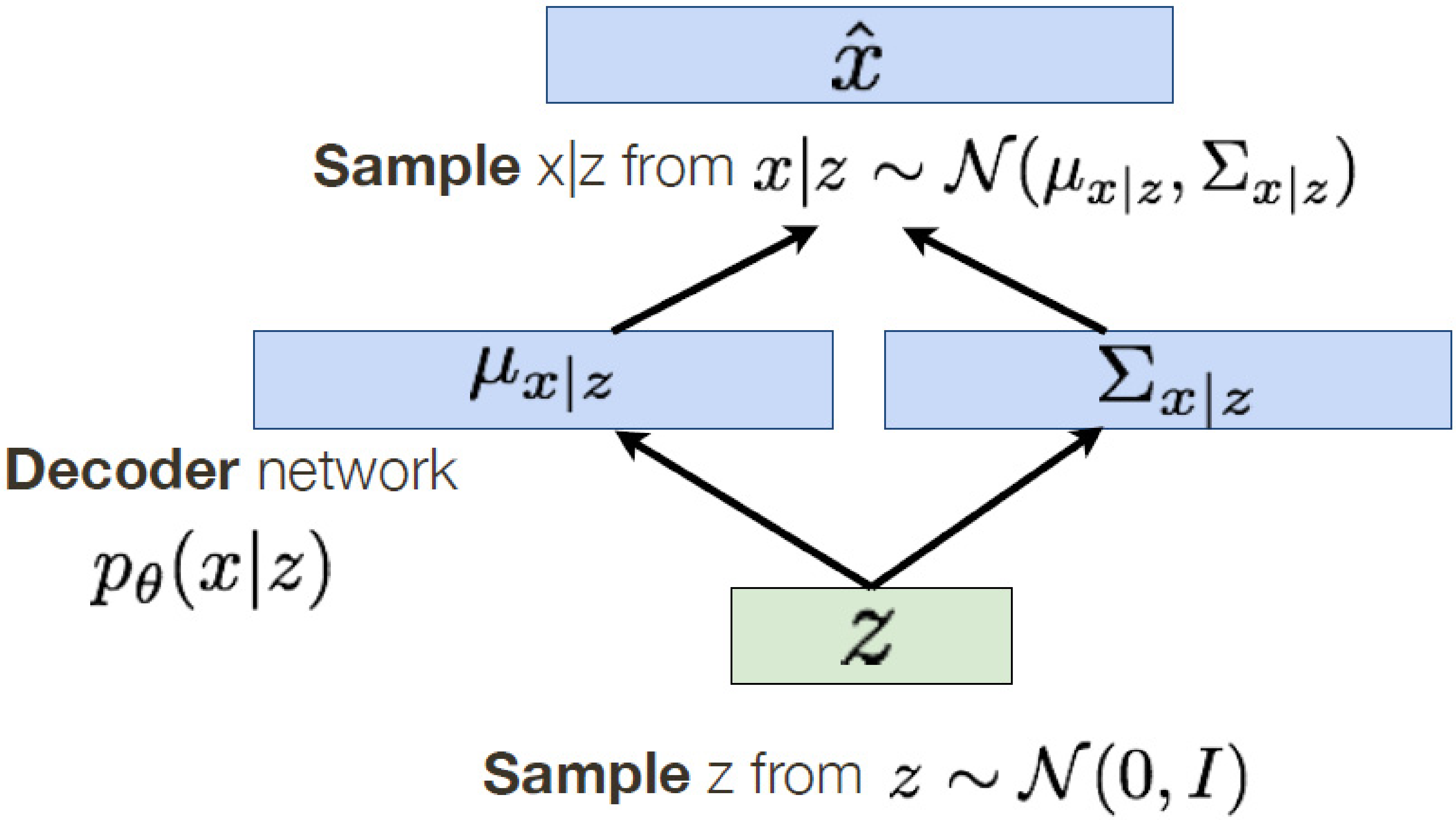
$$-\mathbf{E}_{z\sim q_\phi(z|x)}\log\left(p_\theta\left(x|z\right)\right) + KL\left(q_\phi\left(z|x\right)\|p_\theta(z)\right)$$

Minimize upper bound on loss we care about!

$$\geq -\log p_\theta(x)$$



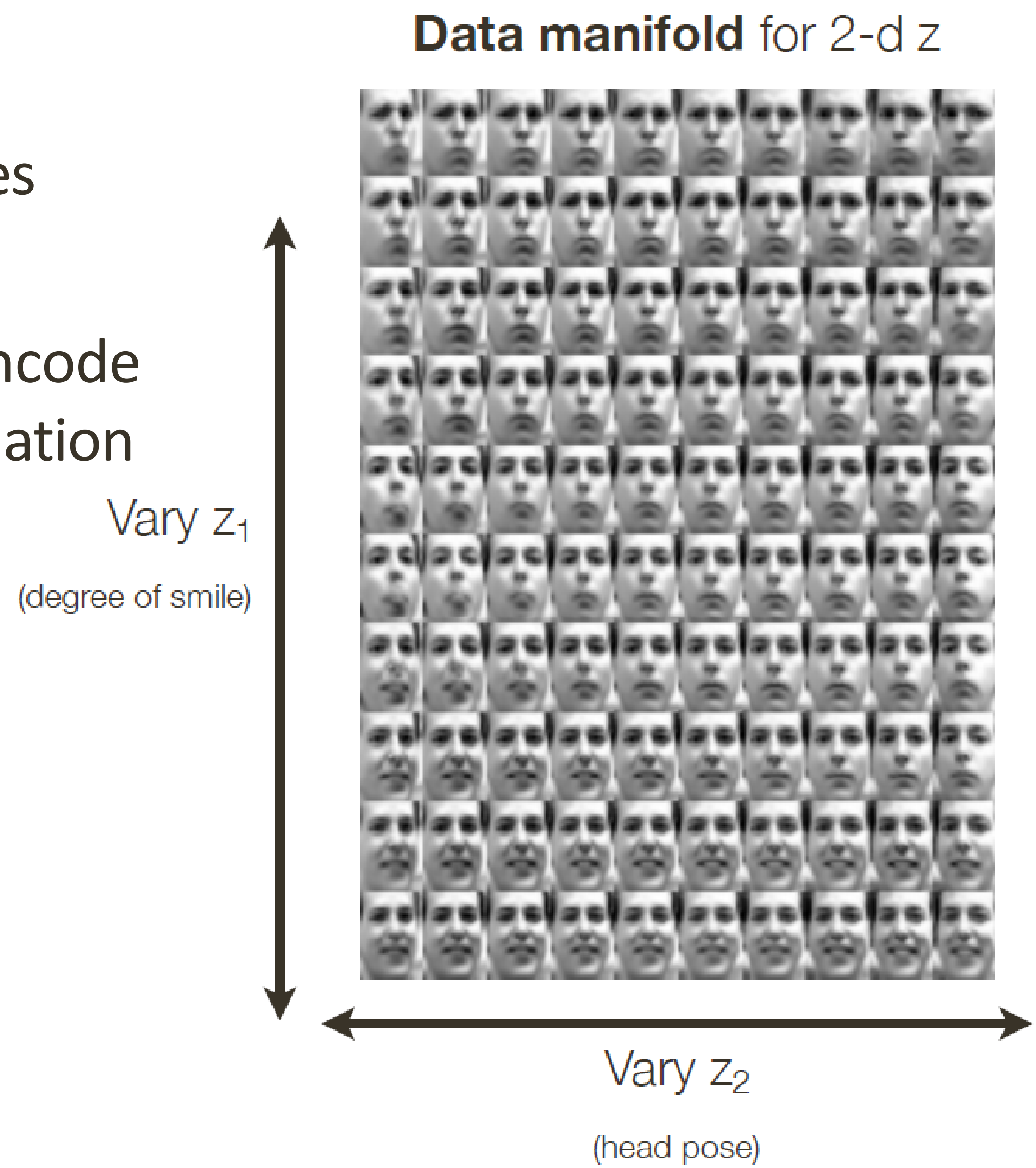
Test time


$\hat{x}$
Sample x|z from $x|z \sim \mathcal{N}(\mu_{x|z}, \Sigma_{x|z})$
$\mu_{x|z}$
$\Sigma_{x|z}$
Decoder network
$p_\theta(x|z)$
z
Sample z from $z \sim \mathcal{N}(0, I)$

VAE Latent space

- Diagonal prior on z => independent latent variables

- Different dimensions of z encode interpretable factors of variation



VAE useful literature

- **Understanding Variational Autoencoders: https://towardsdatascience.com/understanding-variational-autoencoders-vaes-f70510919f73**

- **Tutorial on Variational Autoencoders**

  https://arxiv.org/pdf/1606.05908.pdf

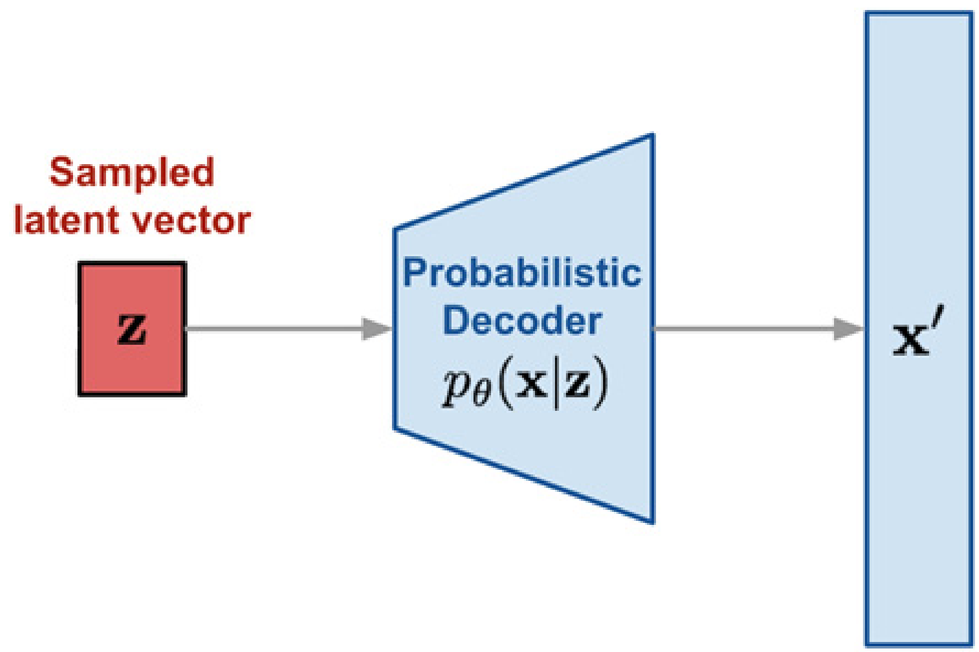
- **Today VAEs are mostly used to produce a low-dimensional latent space of data – latent diffusion models operate on this space…**

How to generate visual data?

- Encoder-Decoders
- Generative Adversarial Networks
- Variational Autoencoders
- Autoregressive Models
- **Diffusion models**

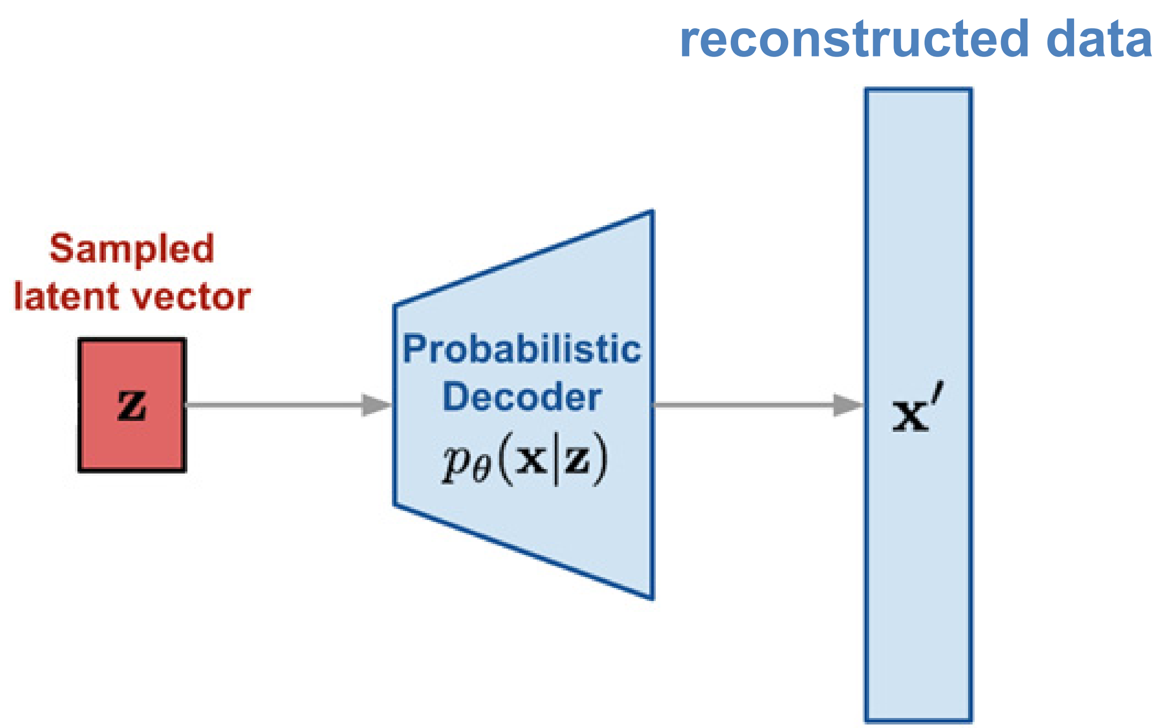
Review: **VAEs**

Explicit generative model i.e., parameterizes data distribution:
$\mathbf{P(\ x\ ) = P(z)\ P(x \mid z)}$, where $\mathbf{P(z)}$ and $\mathbf{P(x \mid z)}$ are Gaussians



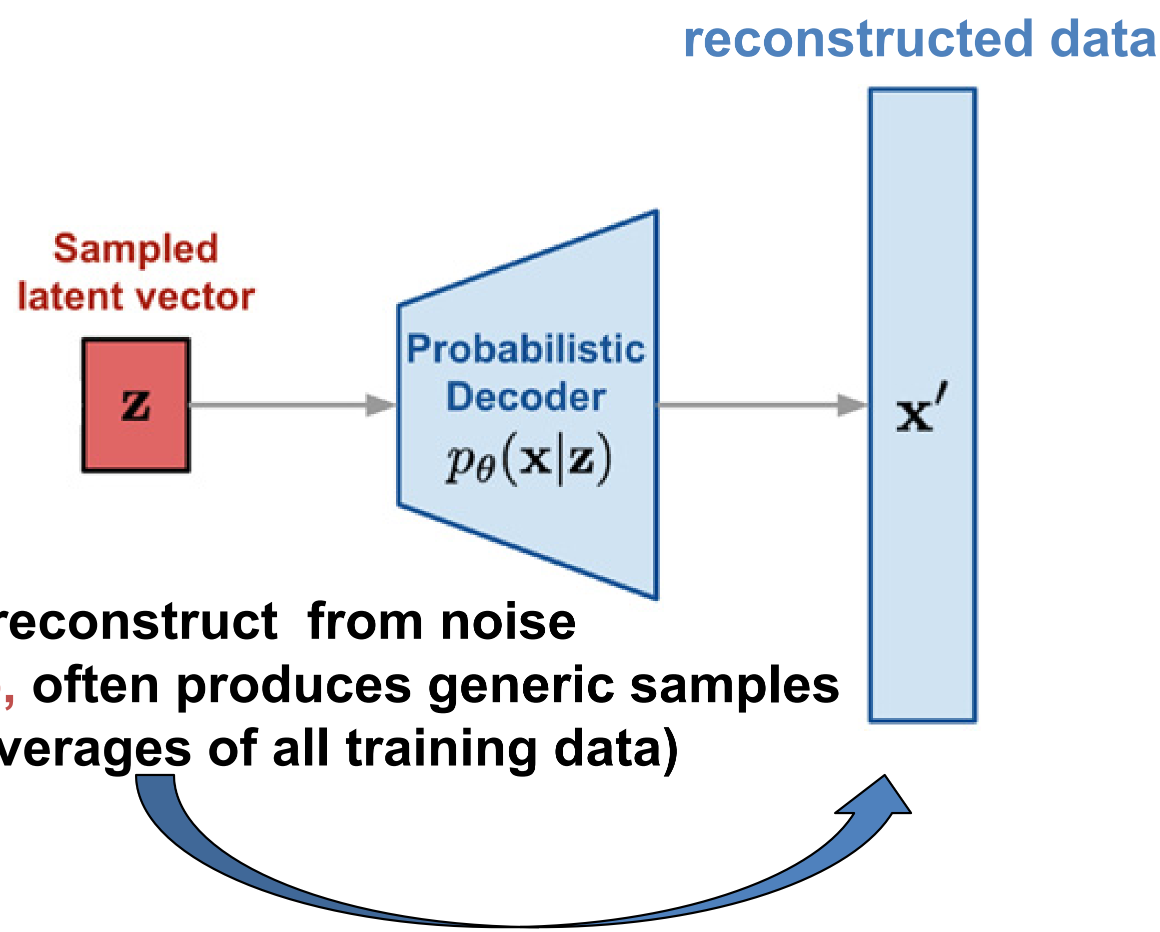
Review: **VAEs**

Many advantages e.g., fast sampling, no mode collapse, effective compression of input data, yet poor quality in generated samples



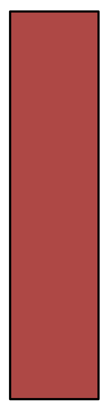
Review: **VAEs**

Many advantages e.g., fast sampling, no mode collapse, effective compression of input data, yet poor quality in generated samples



Diffusion models

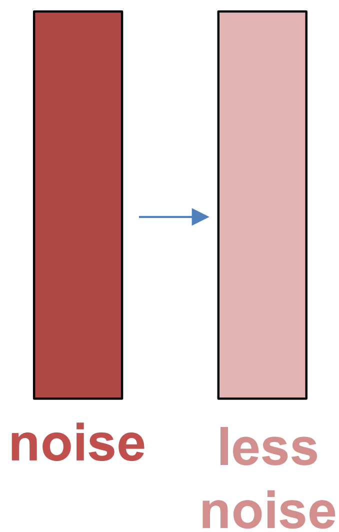
Follow a more gradual, multi-step reconstruction approach



https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models

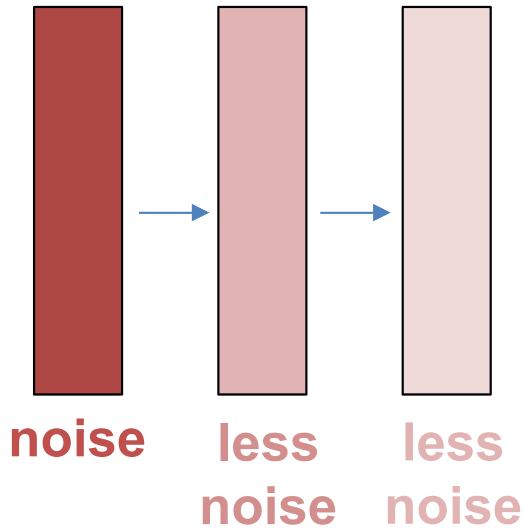
Follow a more gradual, multi-step reconstruction approach



https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models

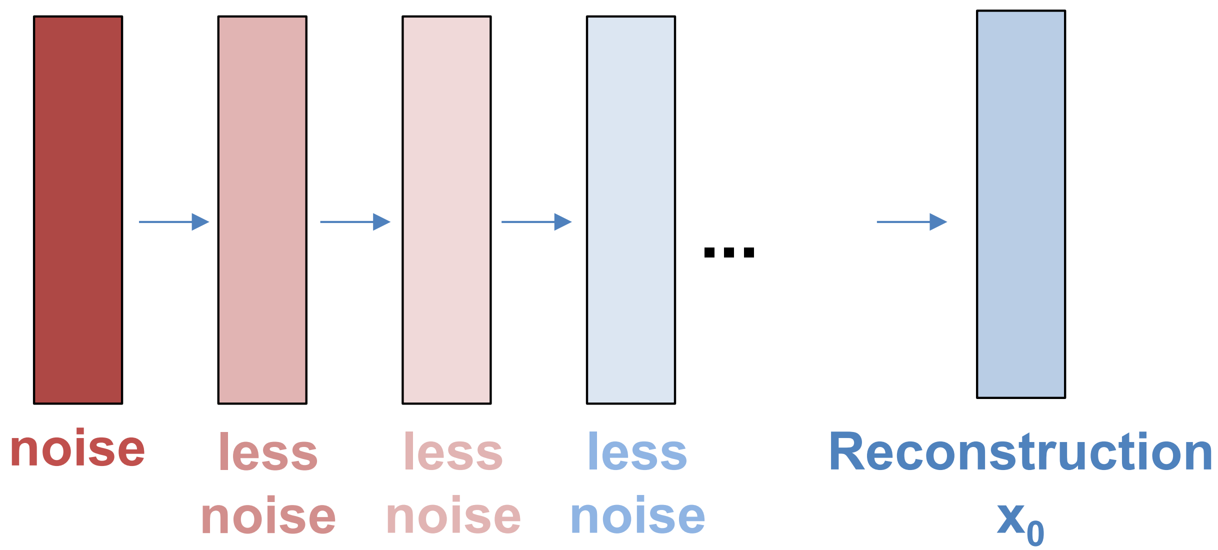
Follow a more gradual, multi-step reconstruction approach



https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models

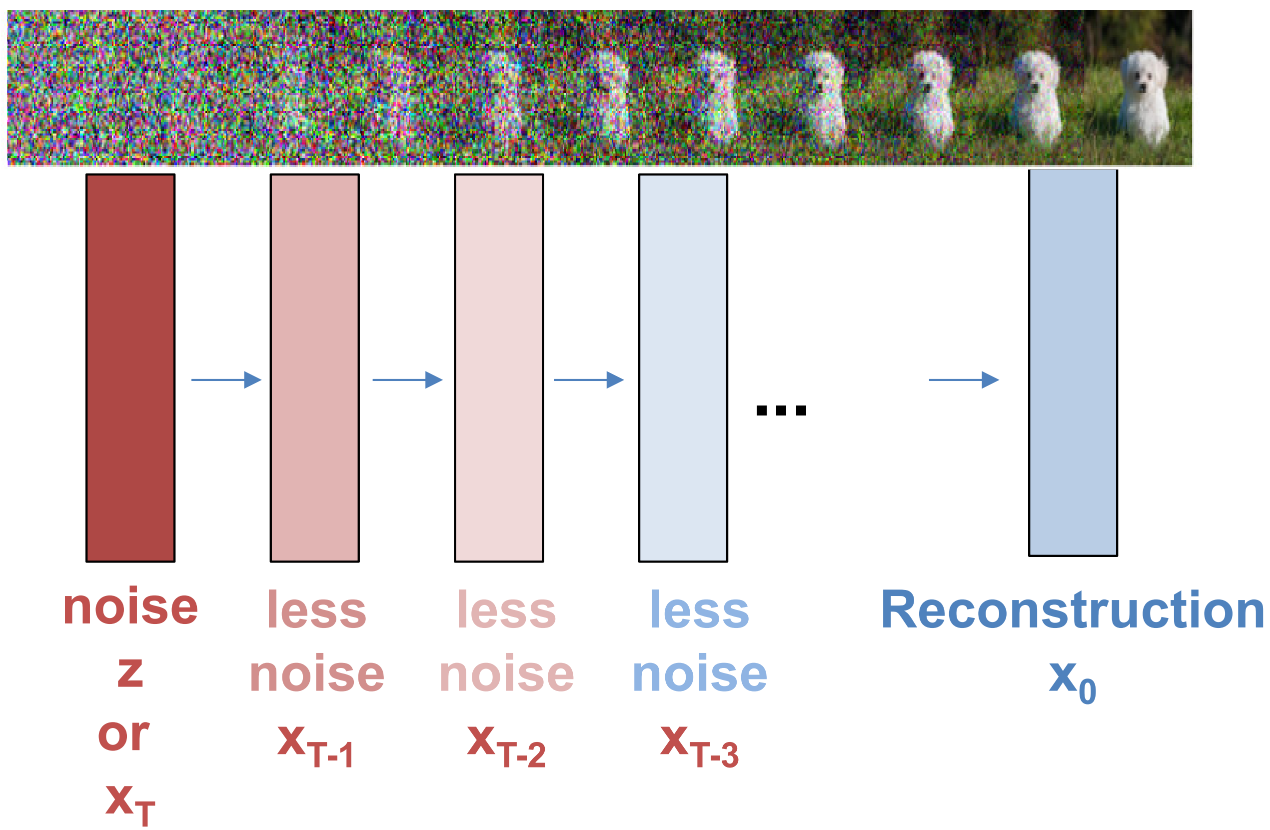
Follow a more gradual, multi-step reconstruction approach



https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models

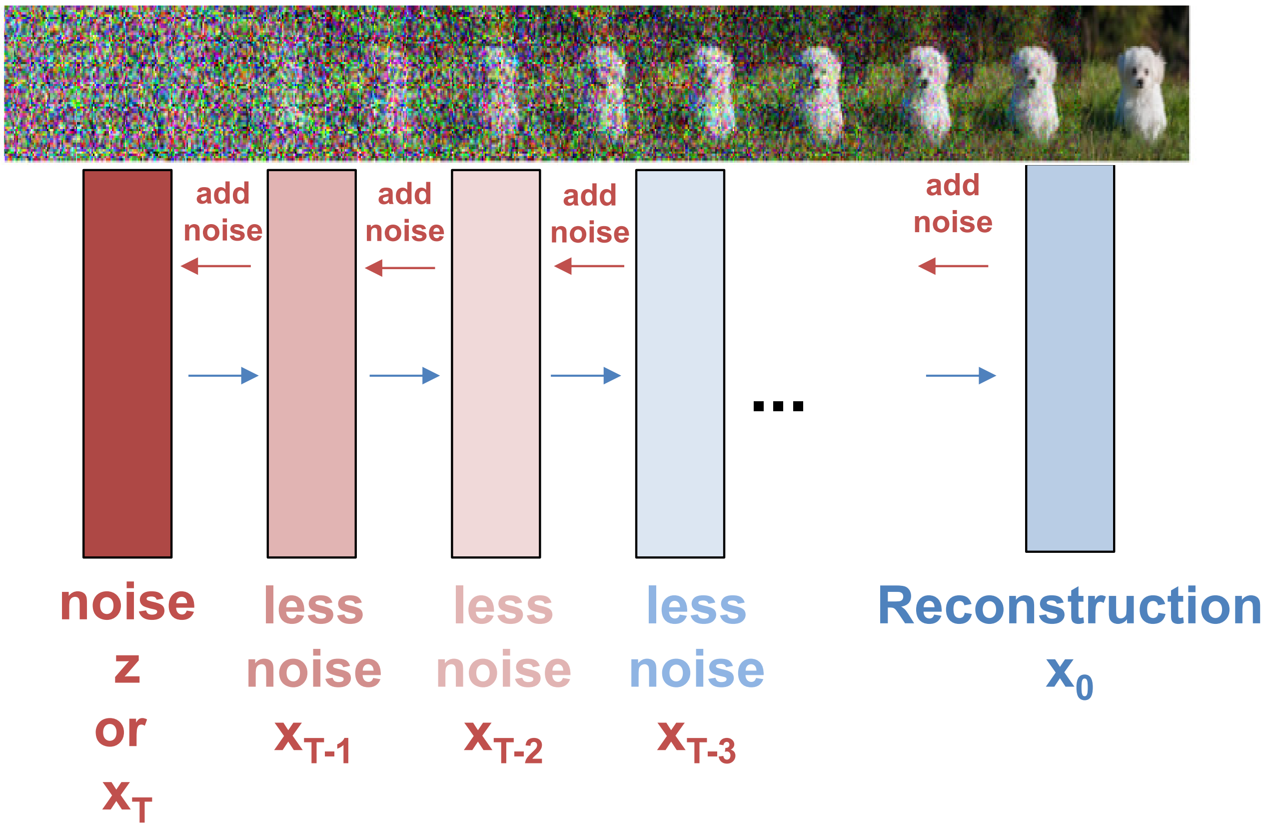
Follow a more gradual, multi-step reconstruction approach



https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models - “Forward” process

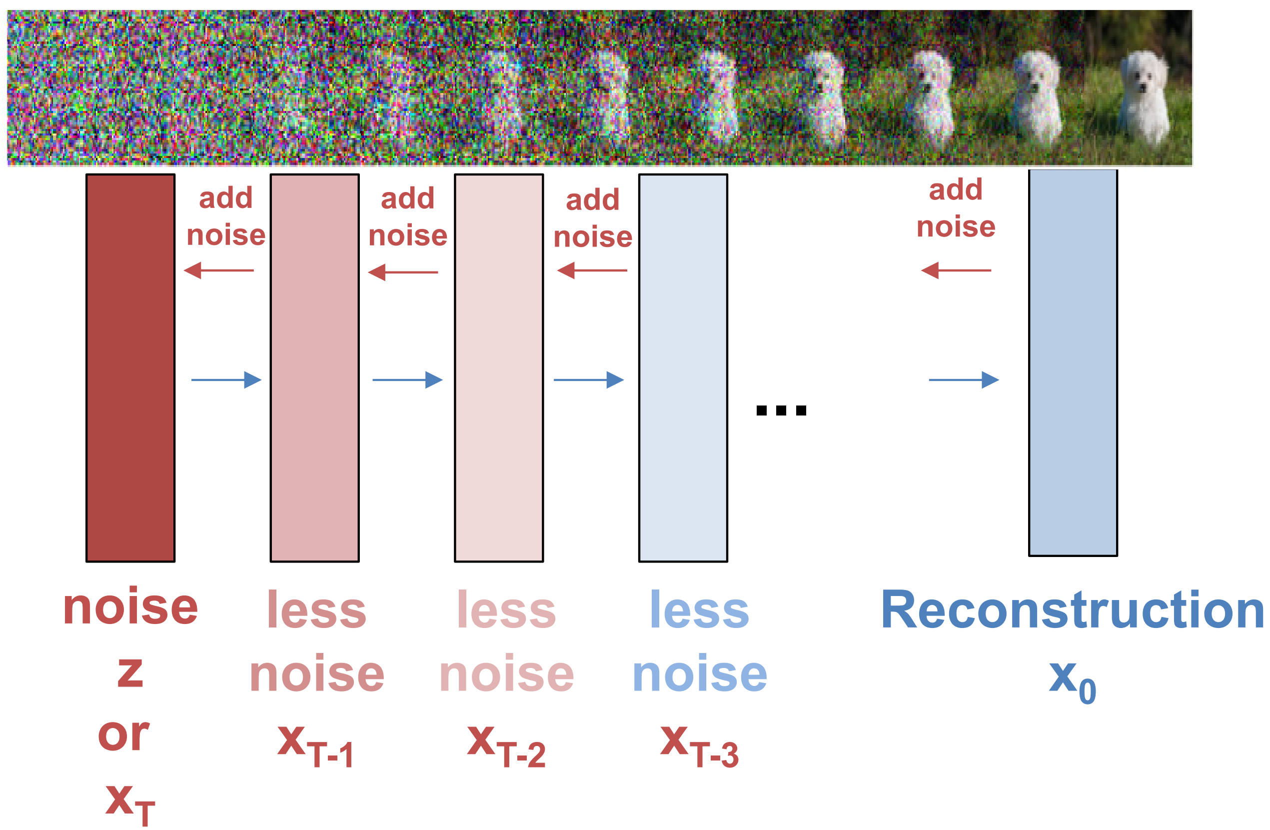
Let’s go from data x_0 to noise **gradually, step-by-step with a simple process: add standard Gaussian noise ε at each step**



https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models - “Forward” process

q(x_t | x_{t-1}) = ***gaussian****(previous image, some variance)*

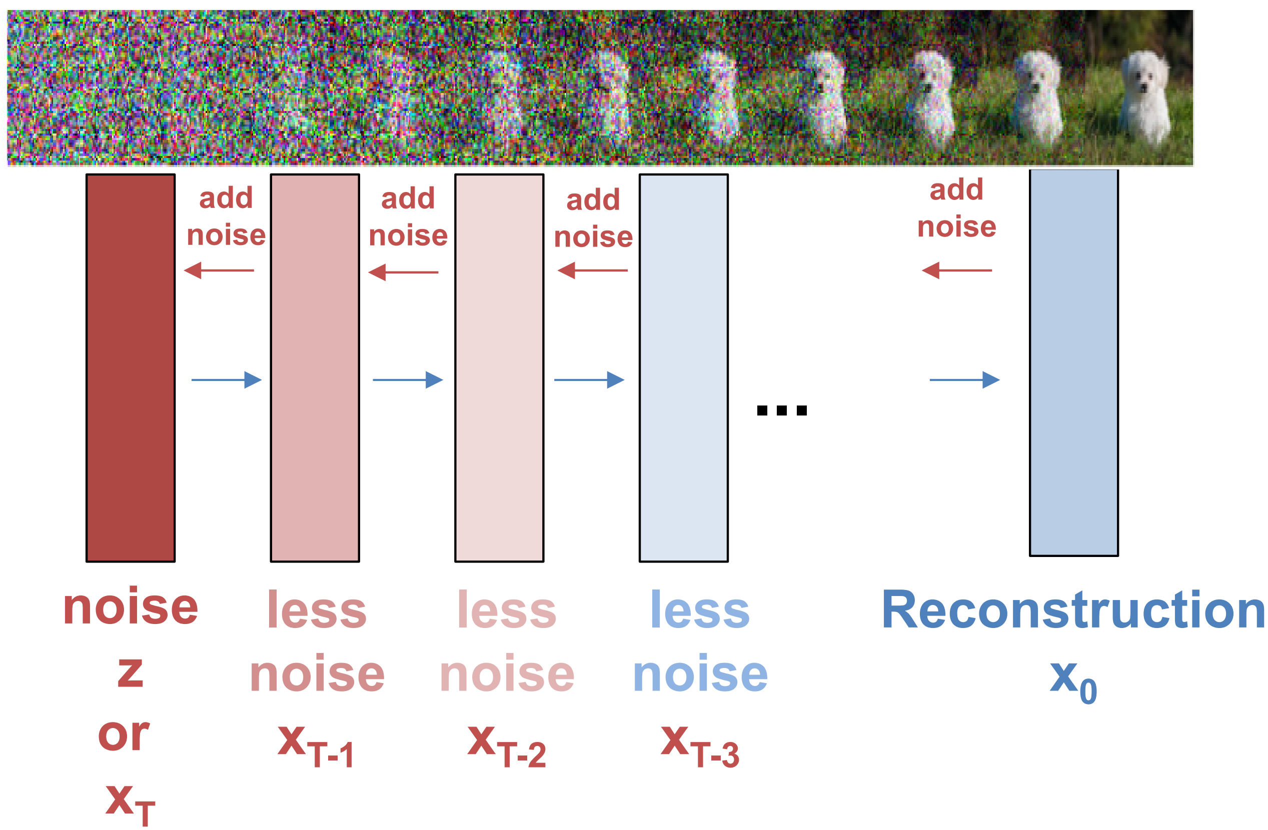


https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models - "Forward" process

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = N(\mathbf{x}_{t-1}, \beta_t I)$$

(where I is the diagonal matrix, i.e., add noise with diagonal covariance scaled by β_t)

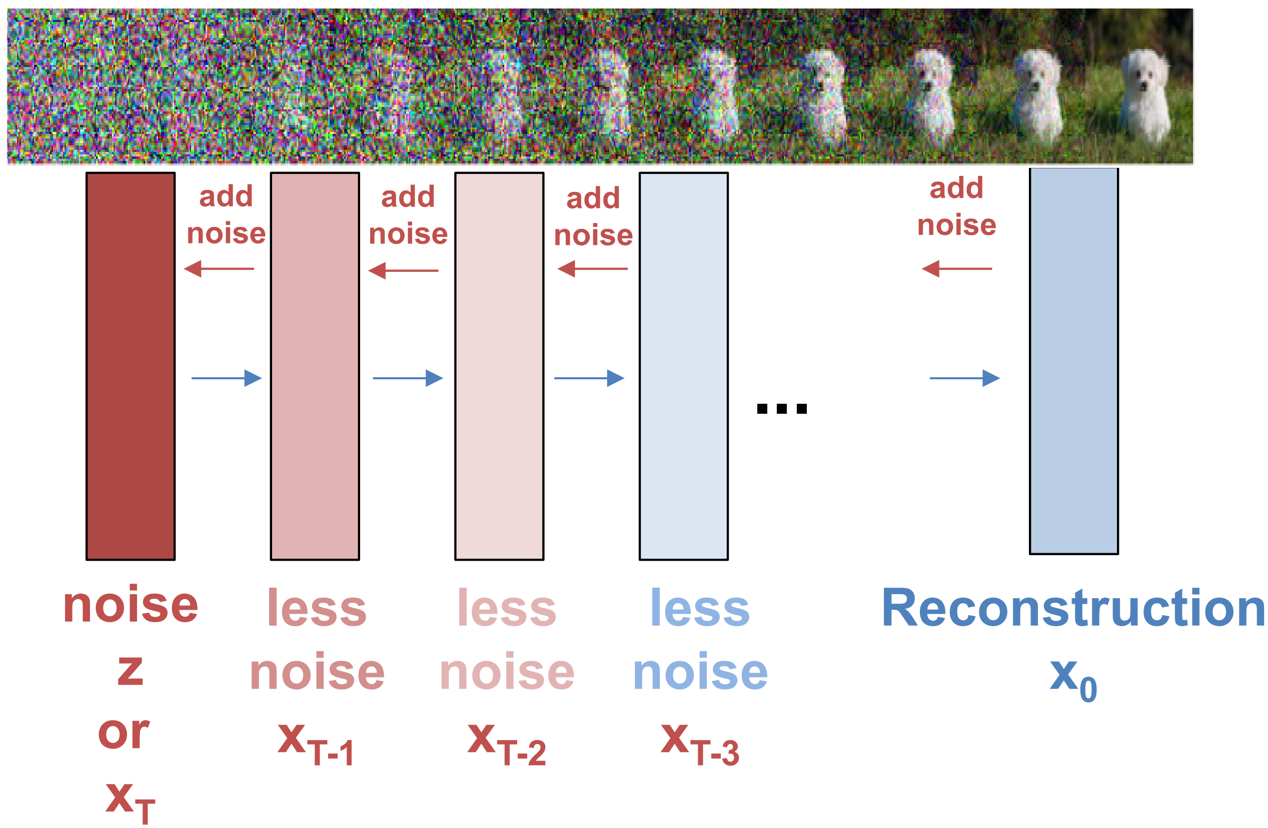


https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models - “Forward” process

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = N(\sqrt{a_t}\,\mathbf{x}_{t-1}, \beta_t \boldsymbol{I})$$

Scale down input and set: $a_t = 1 - \beta_t$... **Why?**

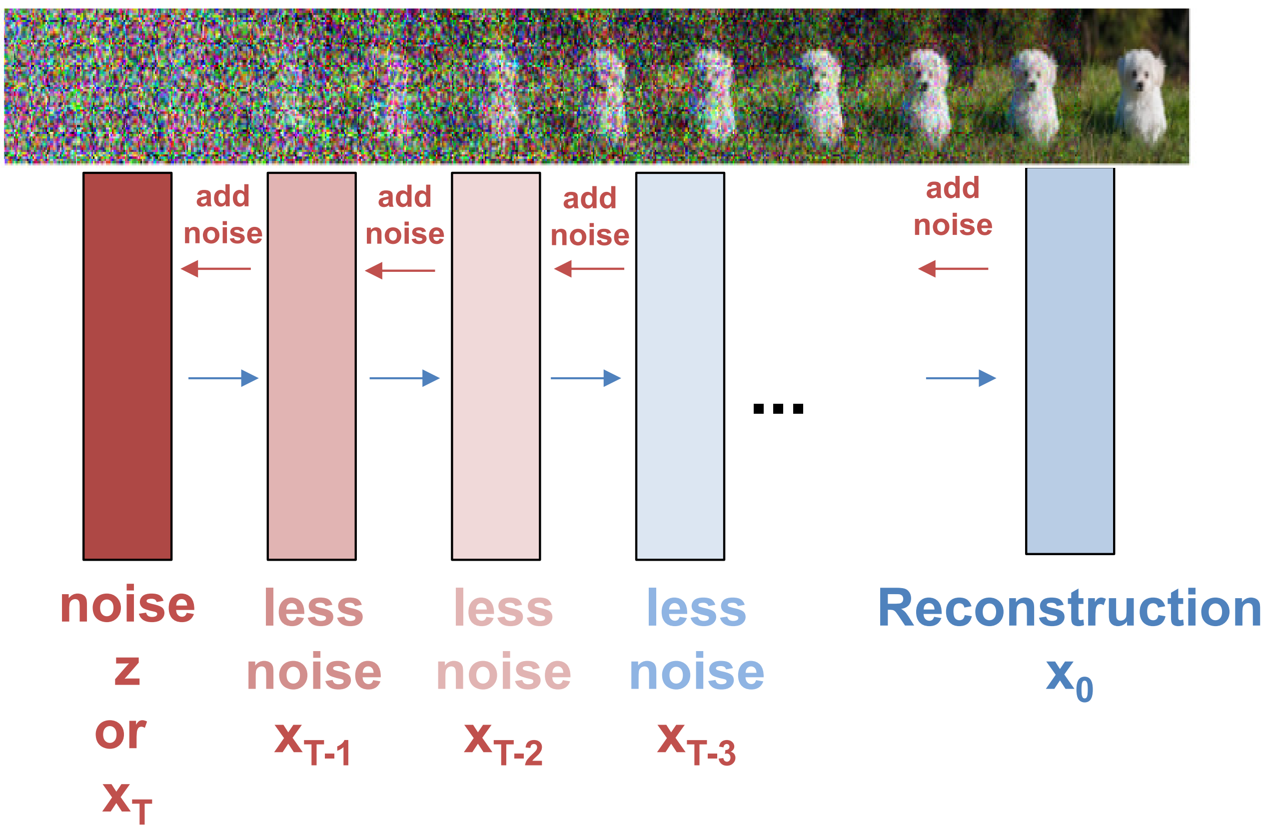


https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models - “Forward” process

In the final step: $z = x_T \sim N(0, I)$

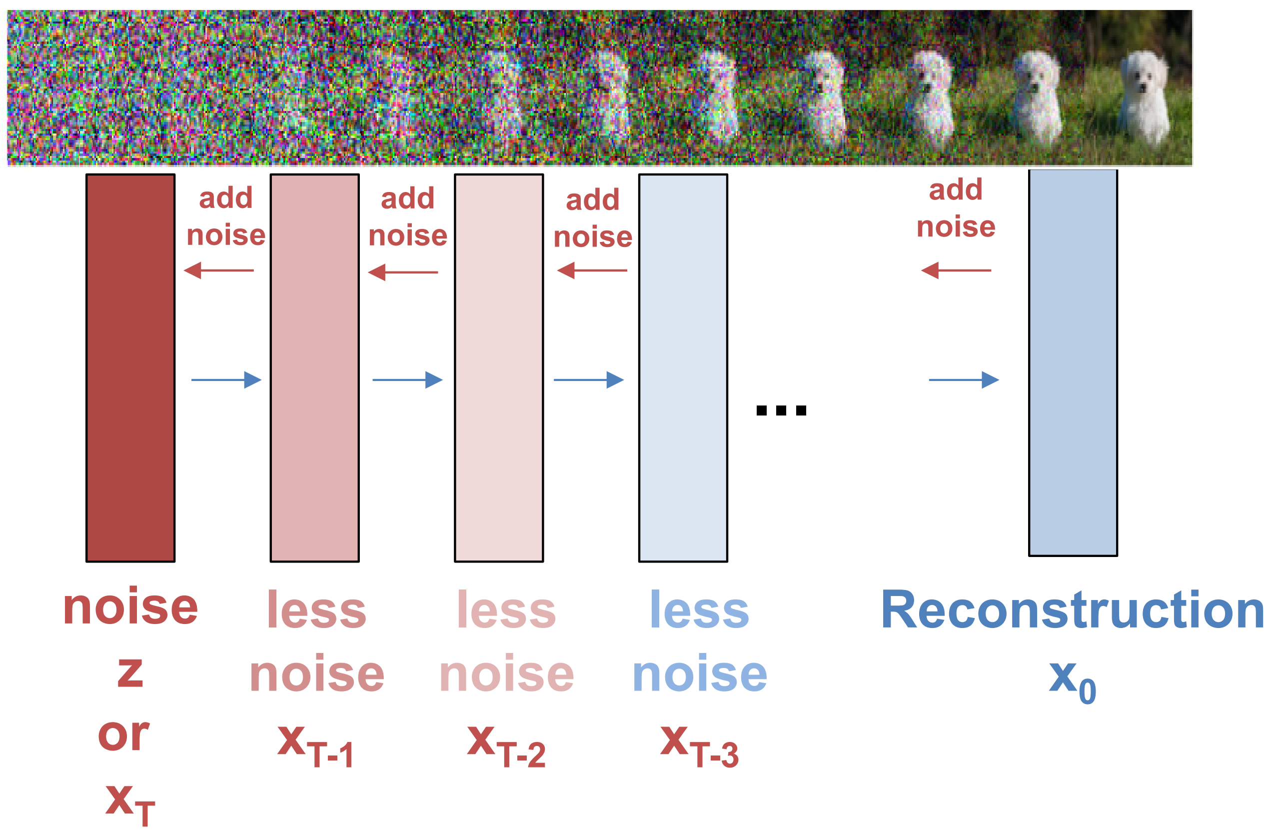
We destroyed the input making it unit Gaussian!



https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models - “Reverse” process

We now need to a way to map noise back to the data!

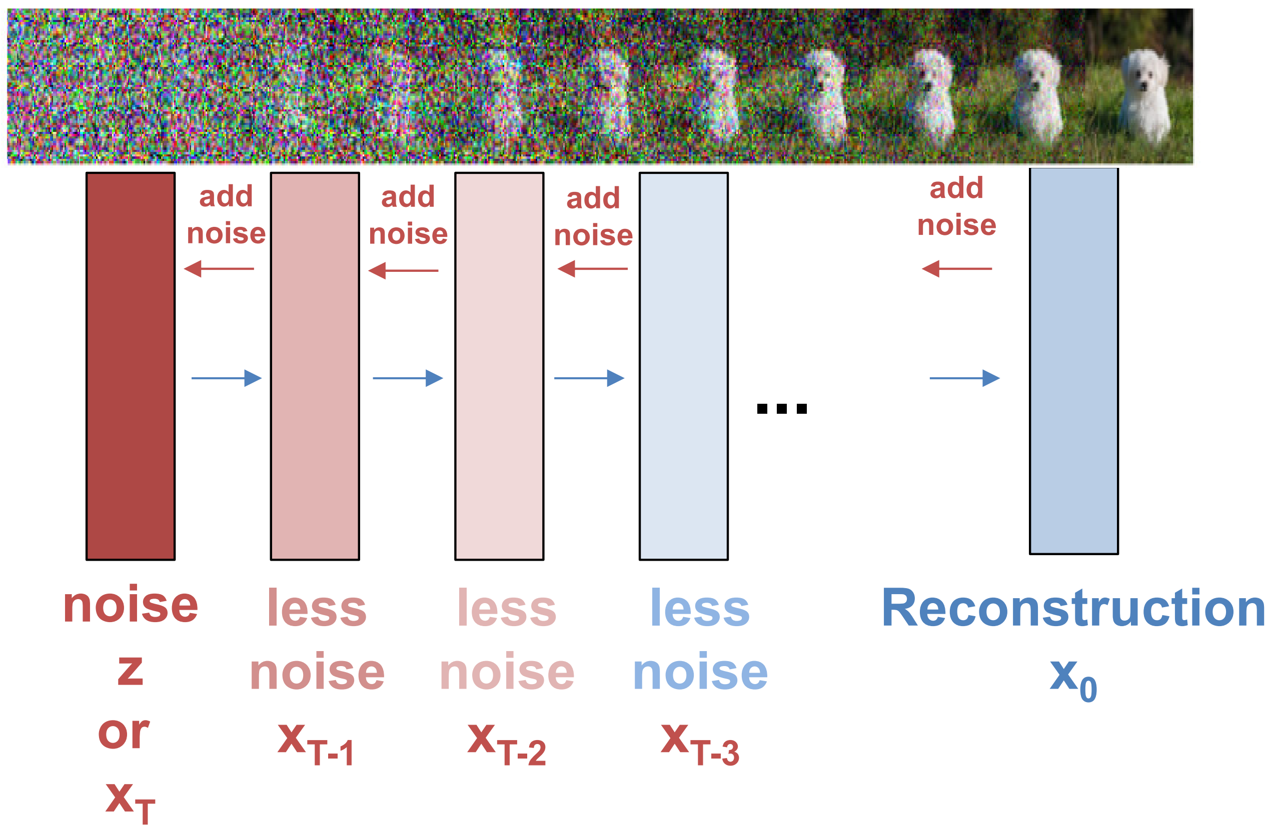


https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models - "Reverse" process

Remember that the forward process was:

$$q(\, x_t \mid x_{t-1}\,) = \textit{gaussian}(\textit{ previous image, some variance})$$

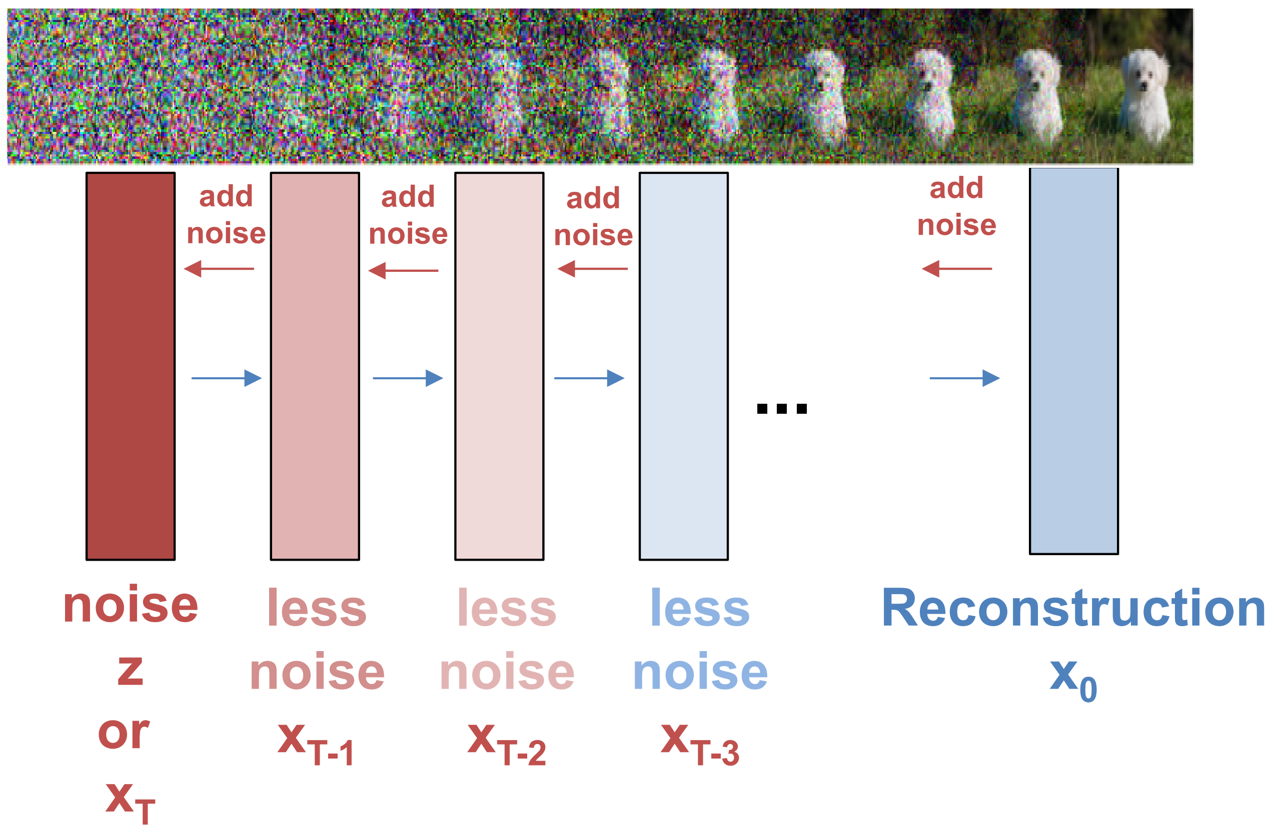


https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models - “Reverse” process

Reverse the process? Complex… depends on entire dataset!

$q(x_{t-1} \mid x_t)$ = ***not a gaussian!***

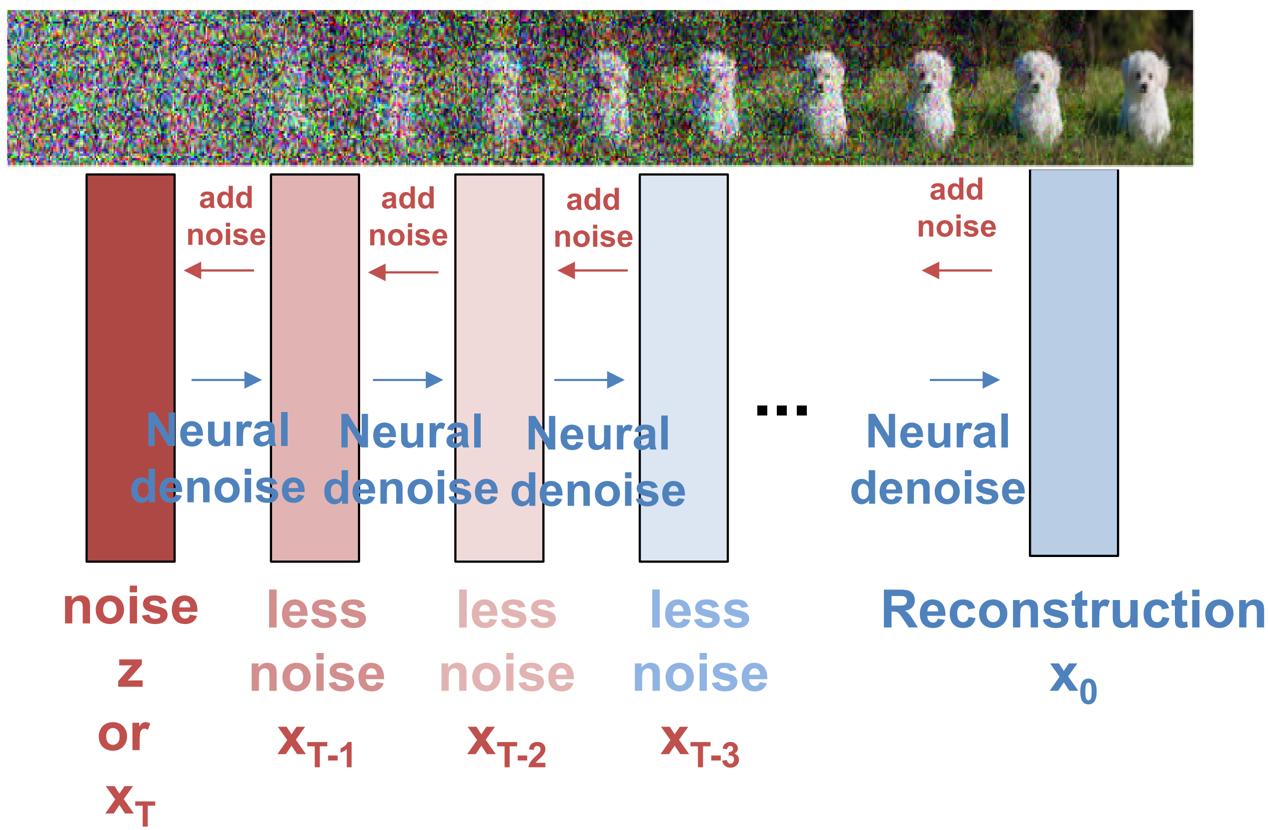


https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models - "Reverse" process

Use a neural network to approximate it in each small step

$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \approx \textit{gaussian(mean, variance)}$

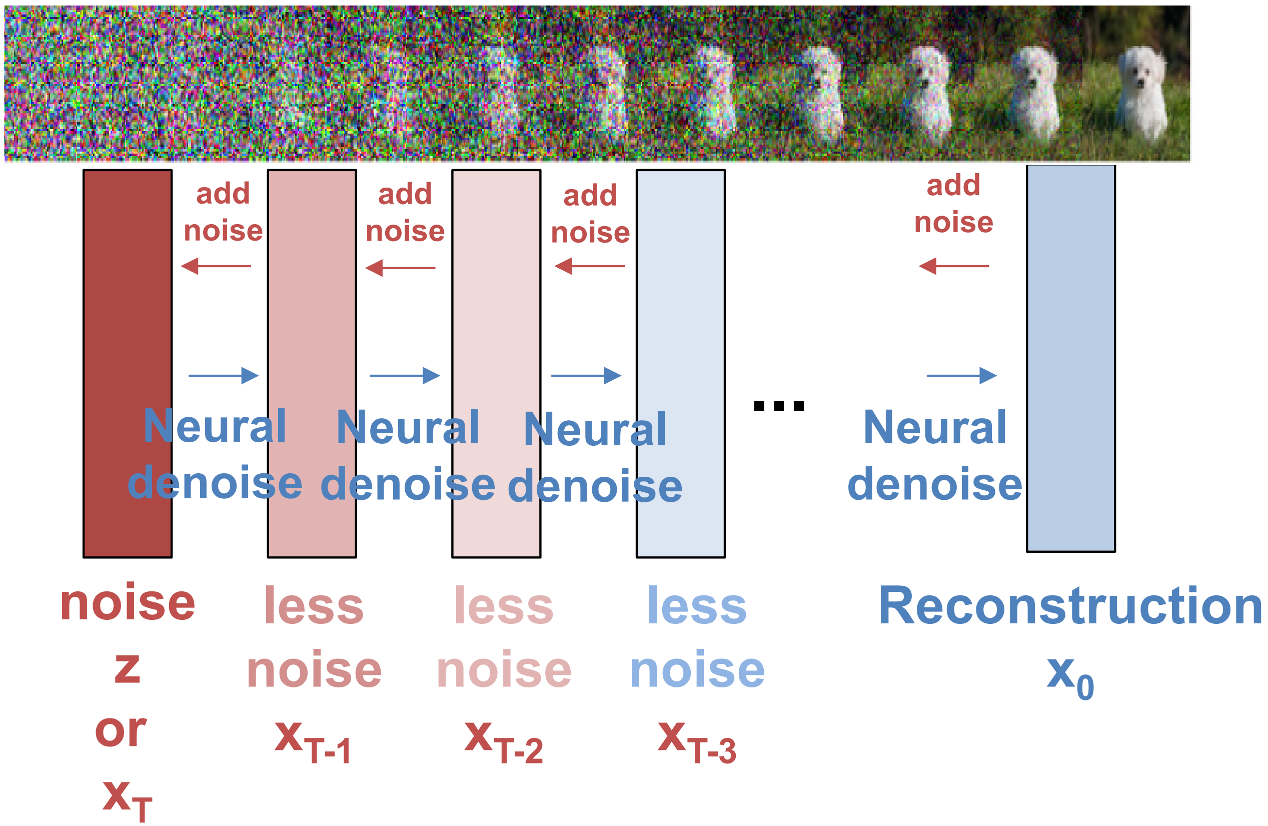


https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models - “Reverse” process

Given current noisy version $\mathbf{x}_t$ and time $\mathbf{t}$, the network predicts mean & covariance based on learned parameters $\boldsymbol{\theta}$:

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = N\left(\mu_\theta(\mathbf{x}_t, \mathbf{t}),\ \Sigma_\theta(\mathbf{x}_t, \mathbf{t})\right)$$

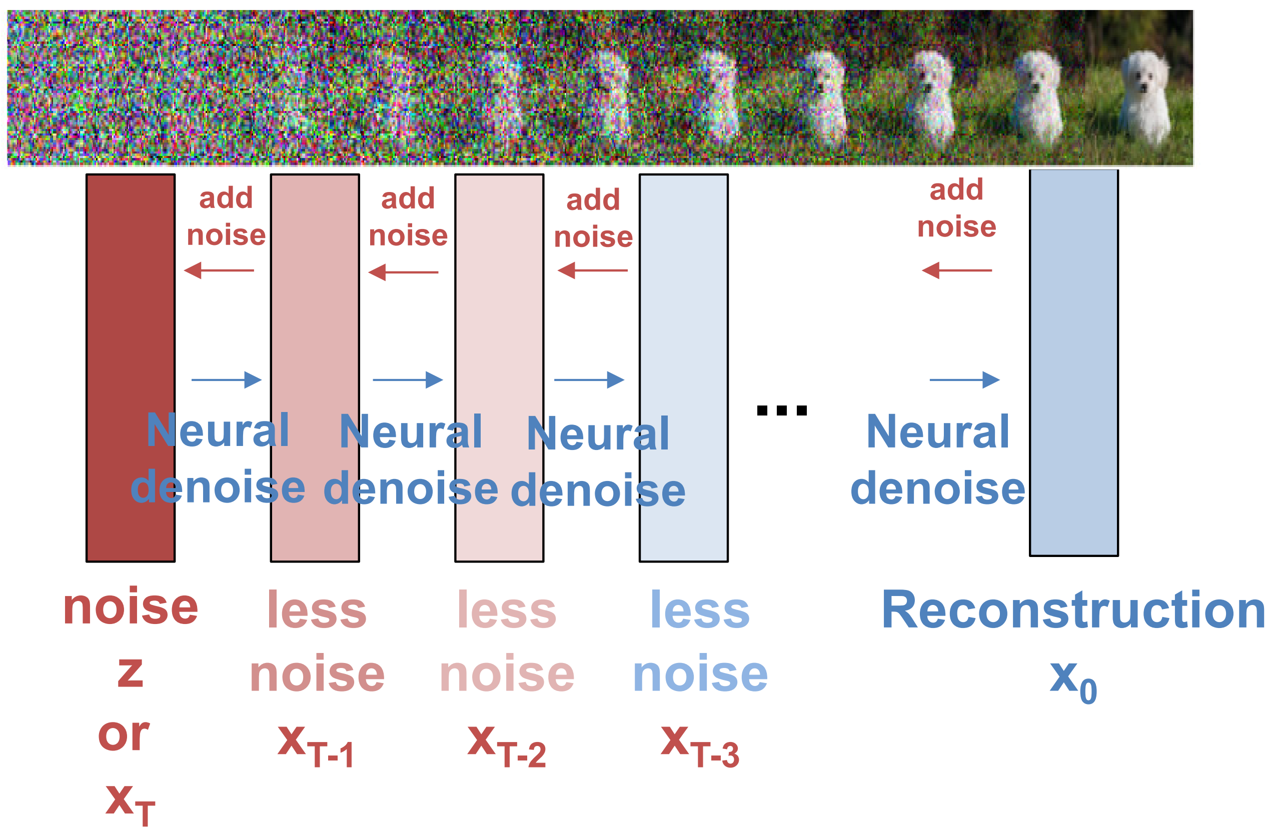


https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models - “Reverse” process

Need to learn these parameters **θ…**

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}\left(\mu_\theta(\mathbf{x}_t, \mathbf{t}),\ \Sigma_\theta(\mathbf{x}_t, \mathbf{t}) \right)$$

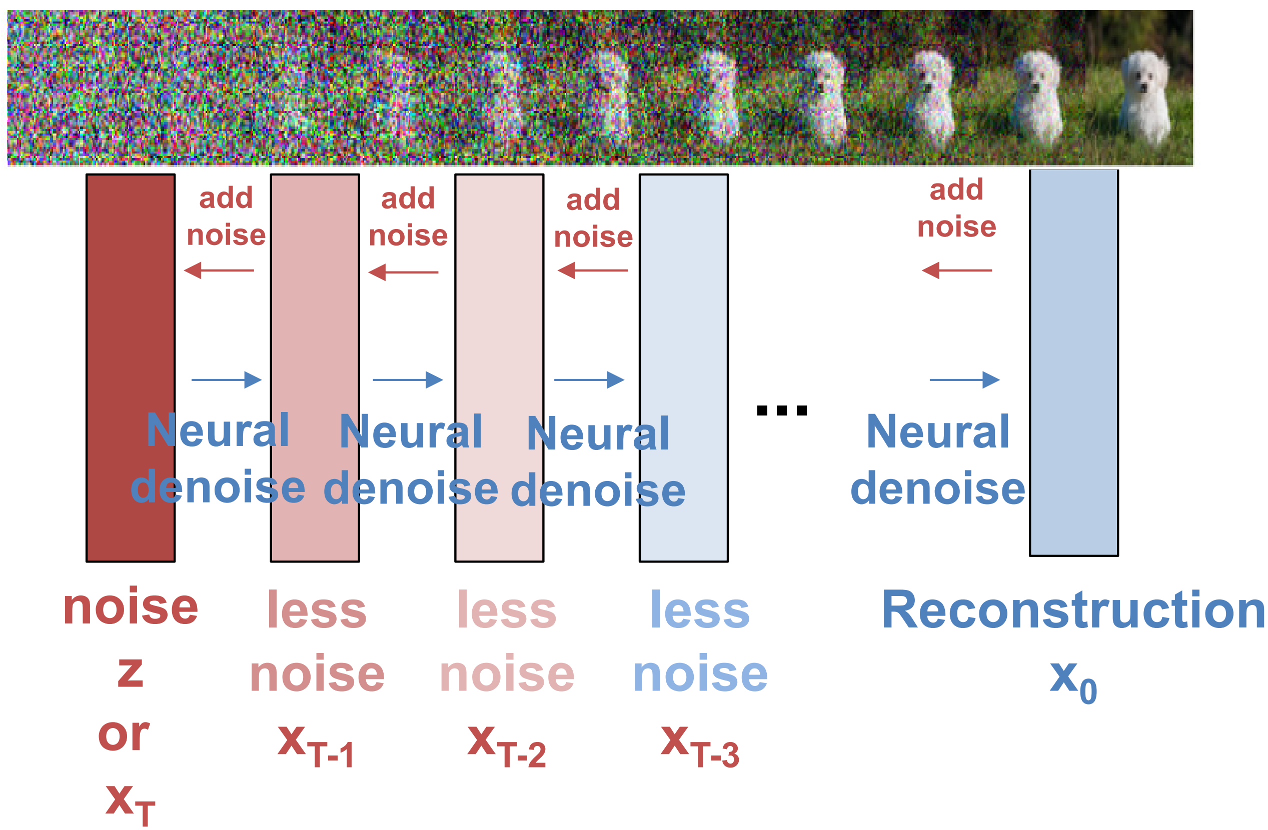


https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models - "Reverse" process

During training, we observe x_0 **(data to reconstruct)**

$q(x_{t-1} \mid x_t, x_0)$

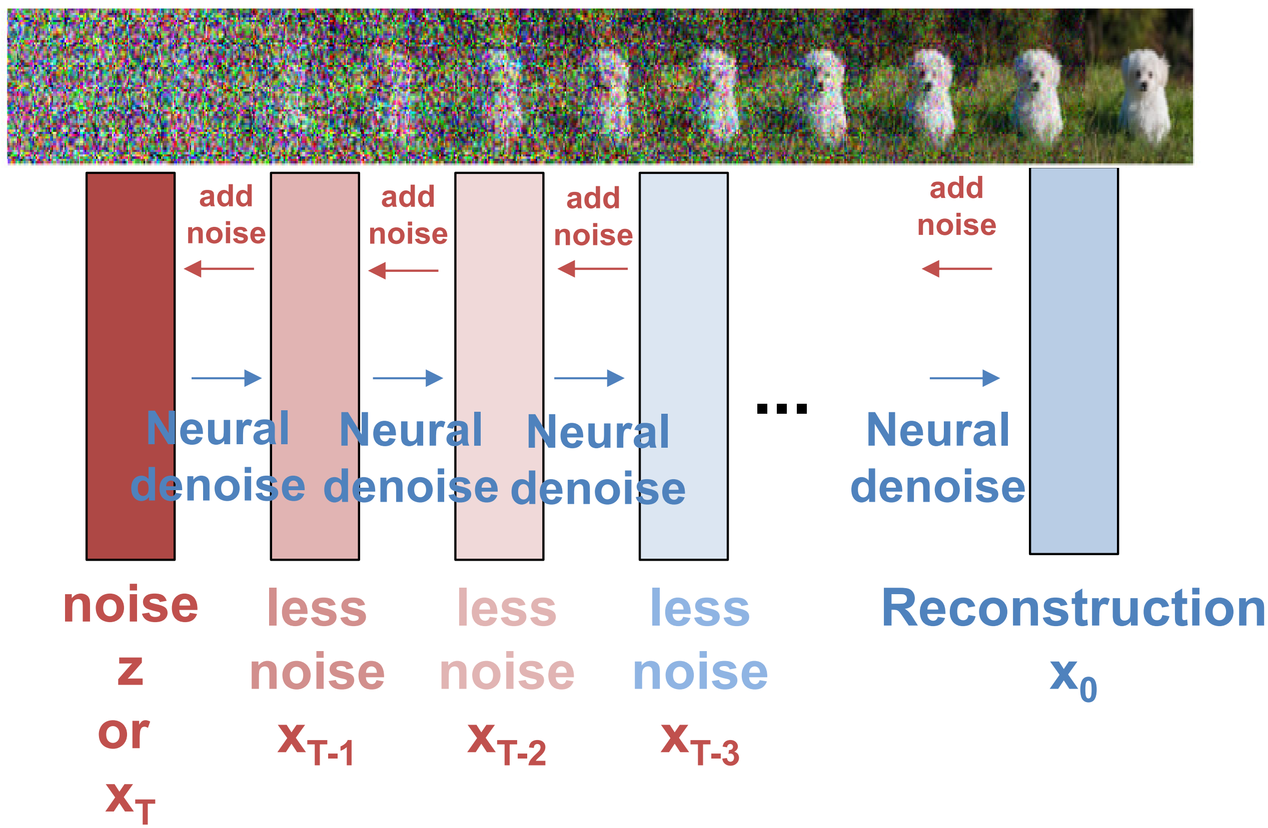


https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models - "Reverse" process

During training, we observe x_0 **(data to reconstruct)**

$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}(\tilde{\mu}_t, \tilde{\Sigma}_t)$ <= ***computable distribution***



https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models - "Reverse" process

During training, we observe $\mathbf{x}_0$ **(data to reconstruct)**

$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = N(\tilde{\mu}_t, \tilde{\Sigma}_t)$ *<= computable distribution*

Argh...

$$\tilde{\mu}_t = \left(\frac{\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_t}}{1-\bar{\alpha}_t}\mathbf{x}_0\right) / \left(\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_t}\right. \frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0$$

where $\bar{\alpha}_t = \prod_{i=1}^{T} \alpha_i$

https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models - "Reverse" process

During training, we observe $\mathbf{x}_0$ **(data to reconstruct)**

$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) = N(\tilde{\mu}_t, \tilde{\Sigma}_t)$ <= ***computable distribution***

Argh...

$$\tilde{\mu}_t = \left(\frac{\sqrt{\alpha_t}}{\beta_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_t}}{1-\bar{\alpha}_t}\mathbf{x}_0\right) / \left(\frac{\alpha_t}{\beta_t} + \frac{1}{1-\bar{\alpha}_t}\frac{\sqrt{\alpha_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}\mathbf{x}_t + \frac{\sqrt{\bar{\alpha}_{t-1}}\beta_t}{1-\bar{\alpha}_t}\mathbf{x}_0\right.$$

$\tilde{\Sigma}_t = \tilde{\beta}_t I$ **and** $\tilde{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t} \cdot \beta_t$

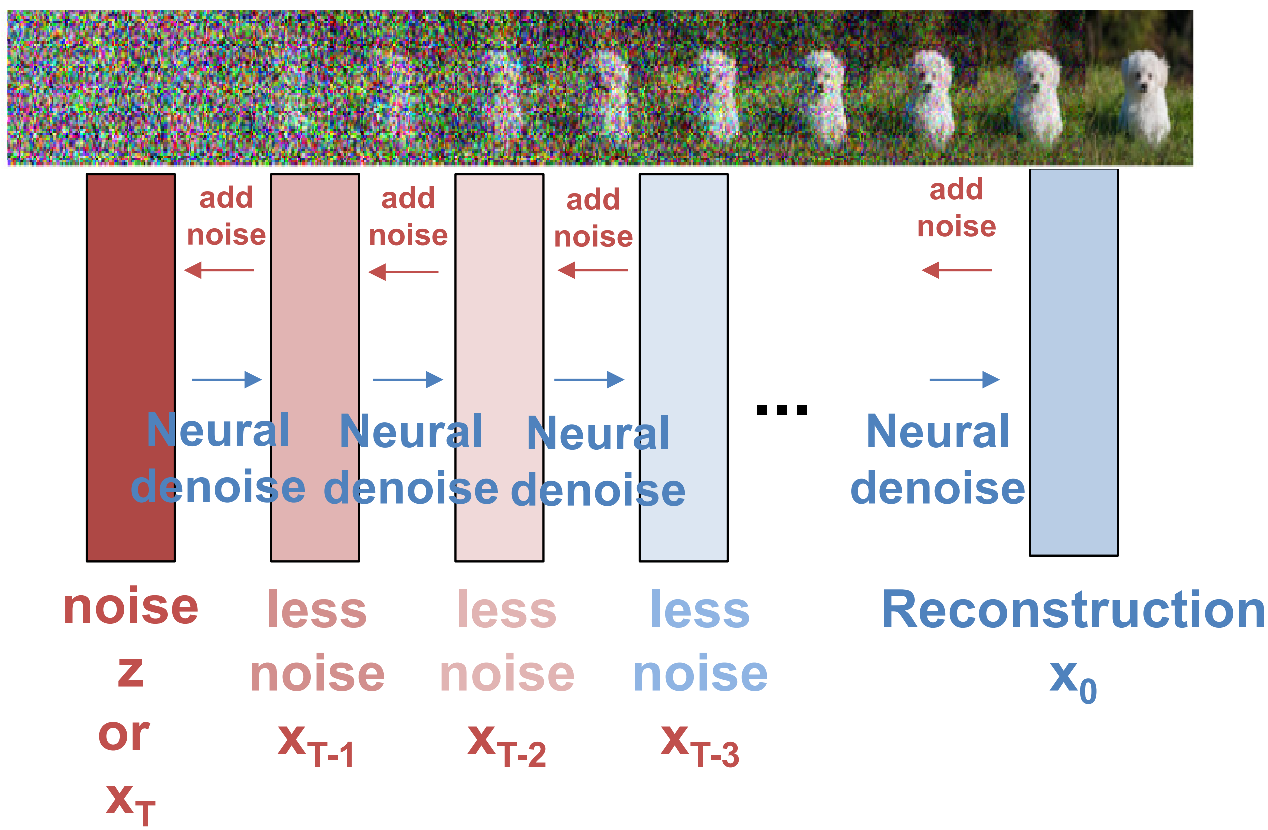
where $\bar{\alpha}_t = \prod_{i=1}^{T} \alpha_i$

https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models - “Reverse” process

Basic idea: make the network predict these previous means & covariances as closely as possible using KL divergence...

$$q(x_{t-1} \mid x_t) = \mathcal{N}\left(\mu_\theta(x_t, t),\ \Sigma_\theta(x_t, t) \right)$$

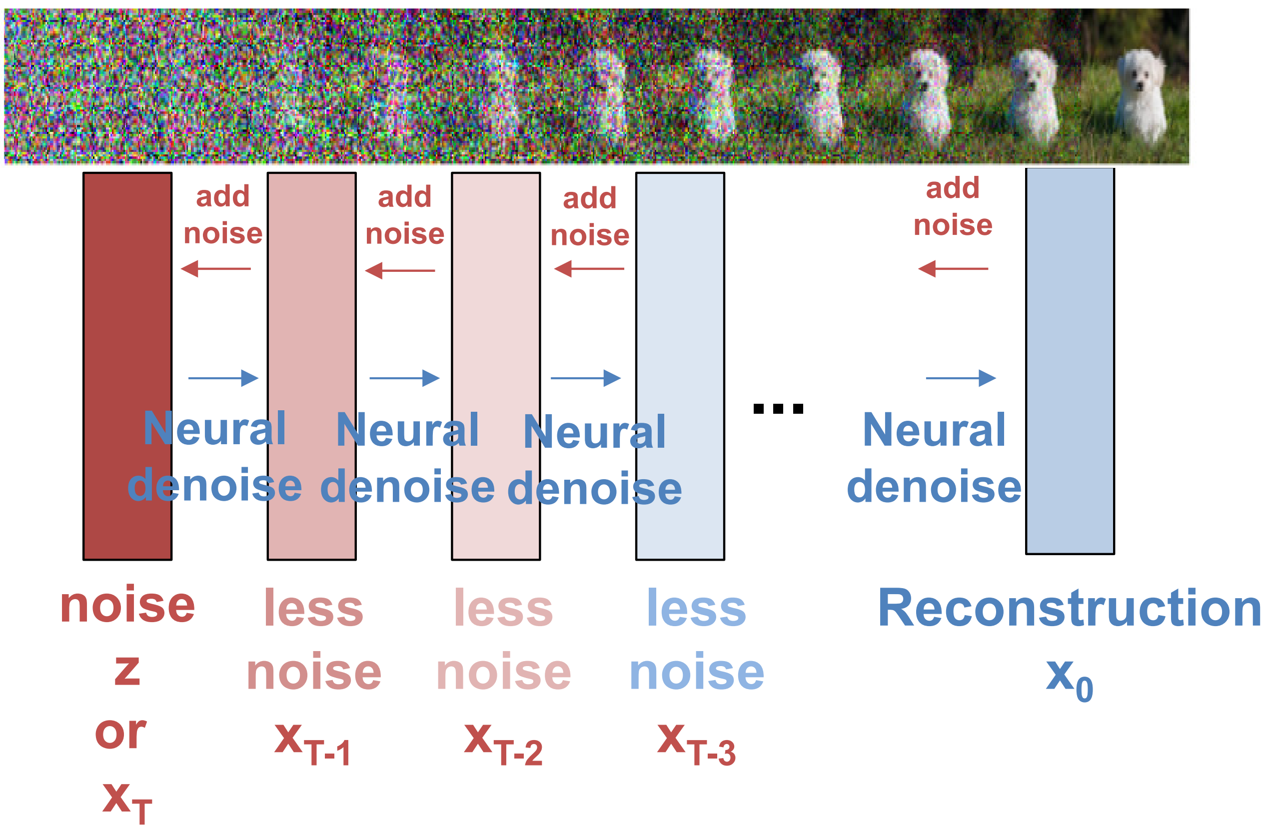


https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models - "Reverse" process

One more helpful trick. Instead of predicting the **mean...**

$$q(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}\left(\boldsymbol{\mu}_\theta(\mathbf{x}_t, \mathbf{t}),\ \Sigma_\theta(\mathbf{x}_t, \mathbf{t})\right)$$



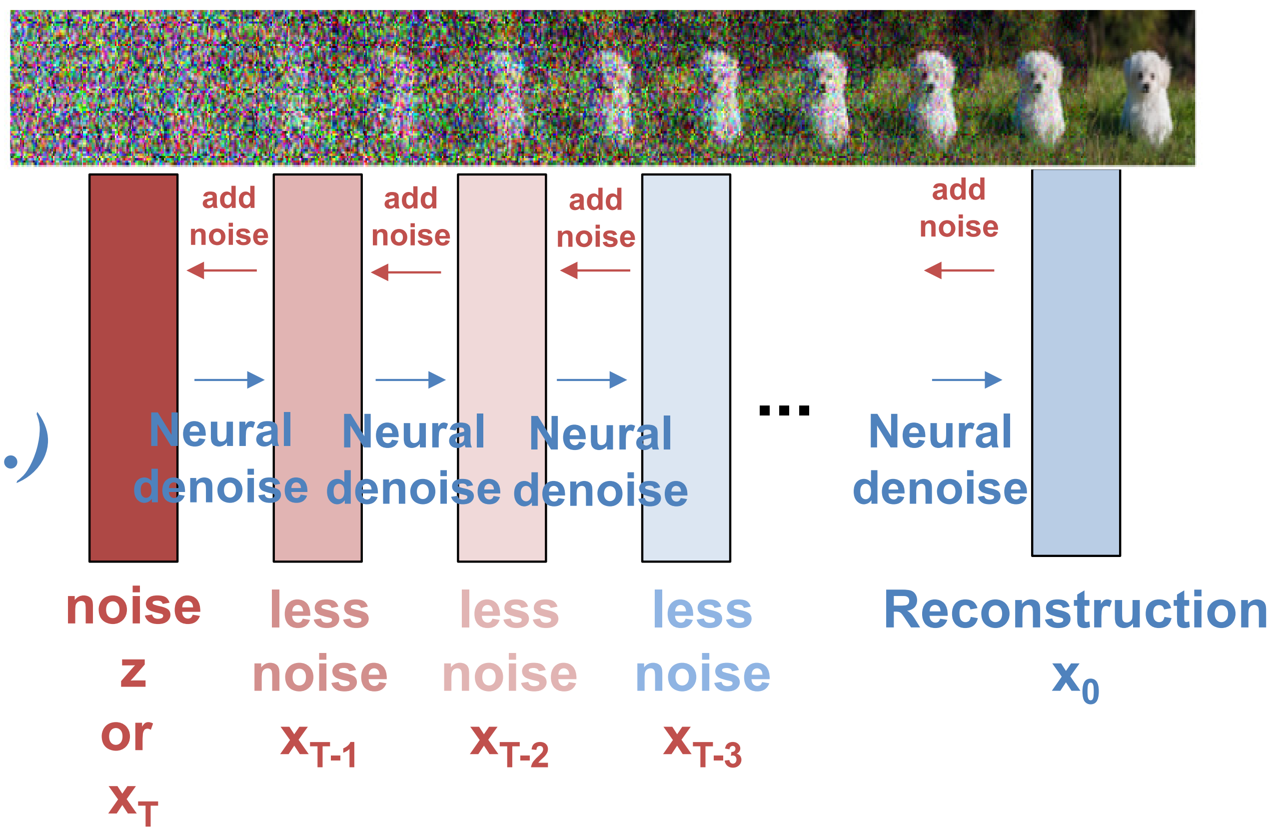
https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models - “Reverse” process

…predict the noise component (think of it as a residual)

$$q(x_{t-1} \mid x_t) = N\left(\alpha_t' x_t - \gamma_t' \varepsilon_\theta(x_t, t),\ \Sigma_\theta(x_t, t)\right)$$

(α_t', γ_t' are scaling factors derived analytically…)



https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Diffusion models - training summary

1. **Sampling step:** generate noisy versions of the input image for a random step

2. **Gradient descent step:** Make the network predict the noise components for that step

https://lilianweng.github.io/posts/2021-07-11-diffusion-models/

Conditional Diffusion models

At test time predict the noise component $\boldsymbol{\varepsilon}_{\theta}(\mathbf{x}_t, \mathbf{t}, \mathbf{c})$ conditioned on some input c e.g., class label, text embedding

or...

Conditional Diffusion models

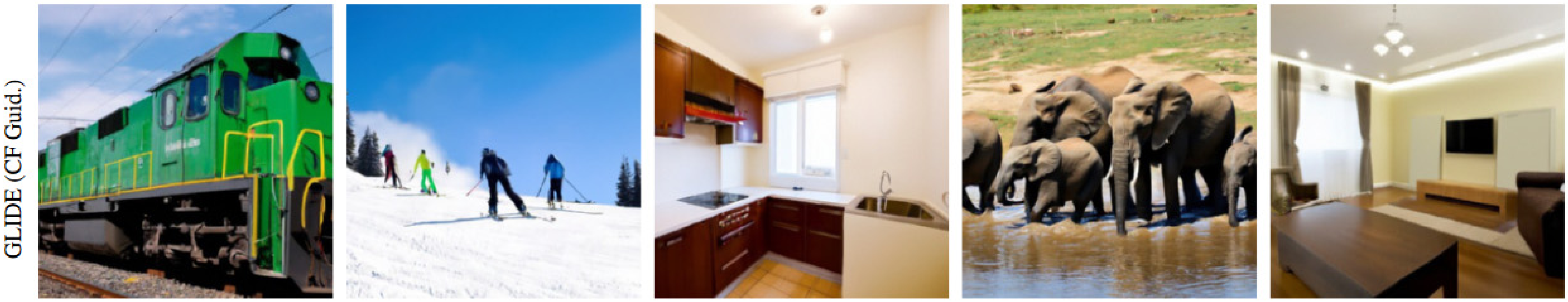
At test time predict the noise component $\boldsymbol{\varepsilon_\theta(x_t, t, c)}$ conditioned on some input c e.g., class label, text embedding

or...

Predict instead $\boldsymbol{\varepsilon_\theta(x_t, t, c) - \varepsilon_\theta(x_t, t)}$ i.e., push the diffusion towards the direction of the input c and away from the direction of input-agnostic noise

GLIDE: Towards Photorealistic Image Generation and Editing with Text-Guided Diffusion Models
https://www.youtube.com/watch?v=gwI6g1pBD84

GLIDE results

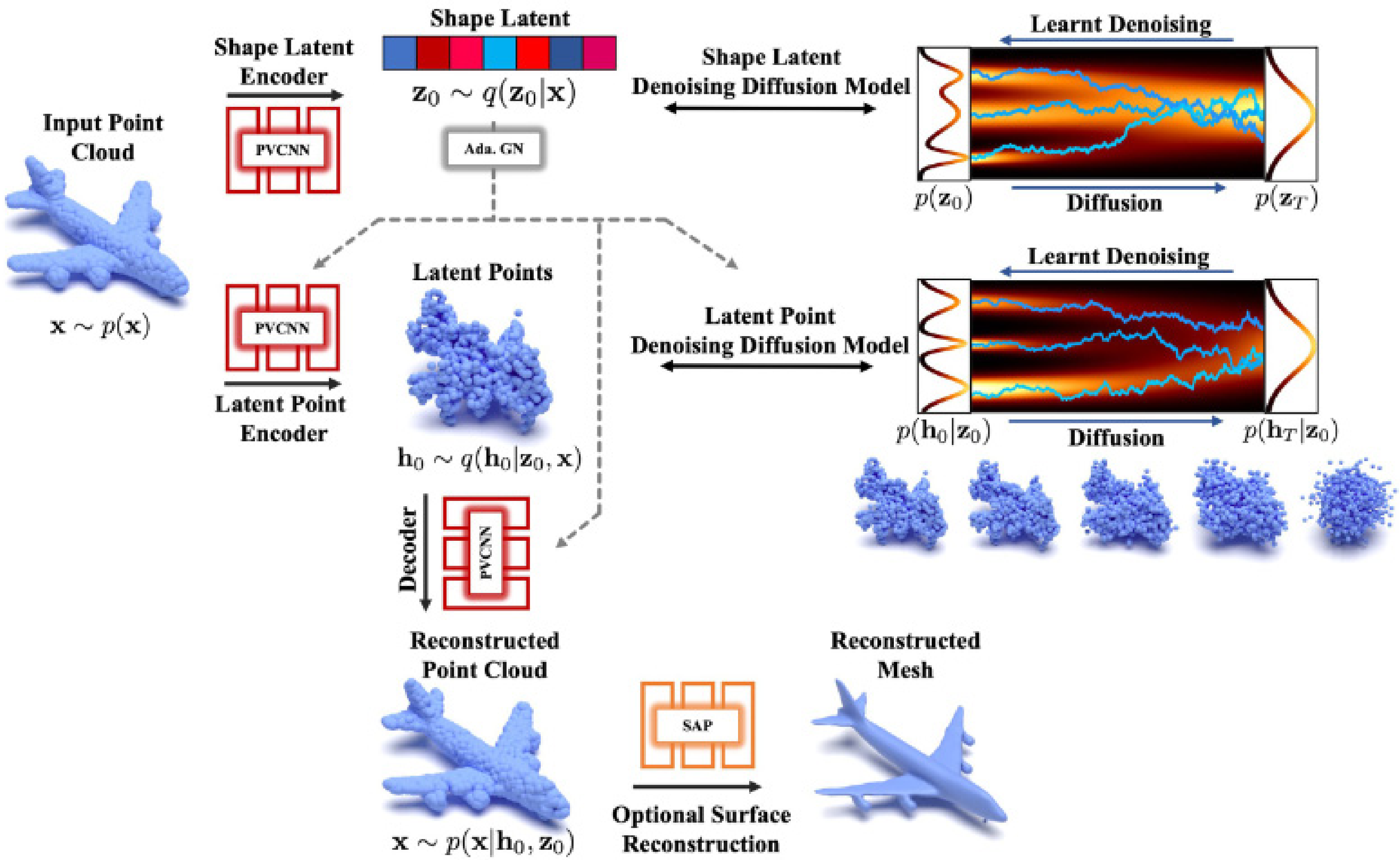


"a green train is coming down the tracks" | "a group of skiers are preparing to ski down a mountain." | "a small kitchen with a low ceiling" | "a group of elephants walking in muddy water." | "a living area with a television and a table"

"a hedgehog using a calculator" | "a corgi wearing a red bowtie and a purple party hat" | "robots meditating in a vipassana retreat" | "a fall landscape with a small cottage next to a lake"

See also **Dall-E 2**: https://cdn.openai.com/papers/dall-e-2.pdf

LION: Latent Point Diffusion Models for 3D Shape Generation



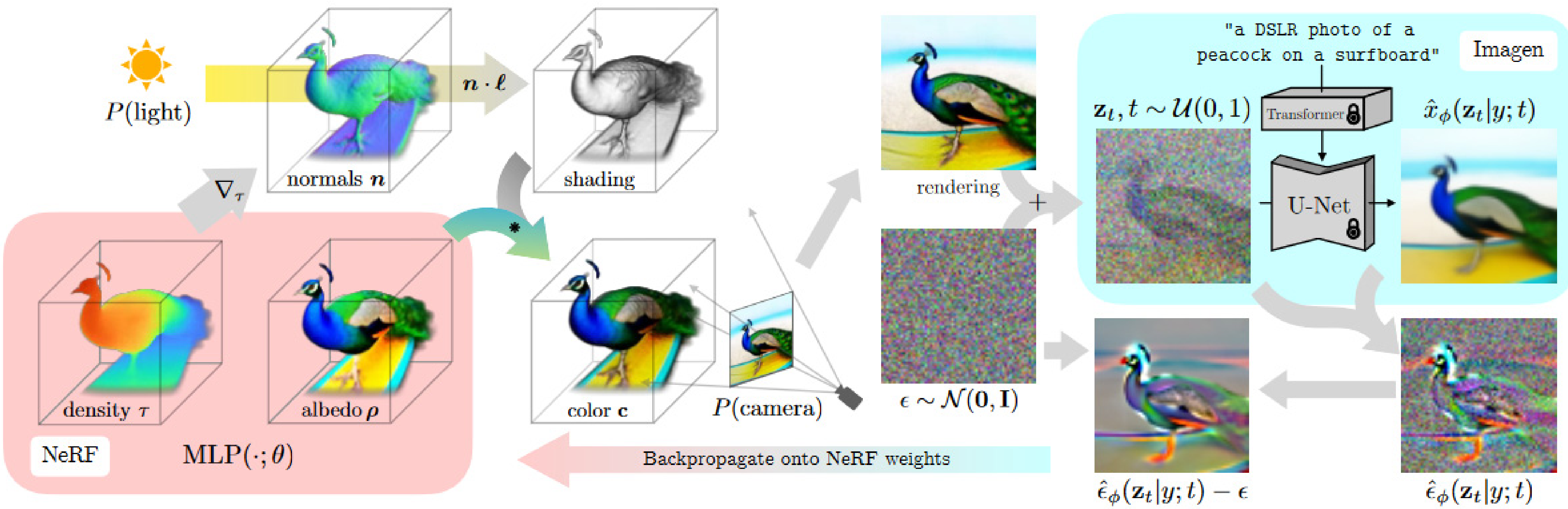
Training only on 3D data?

- 3D datasets are limited in size

- Image diffusion models e.g., Dall-E dataset is 250M images!

- **Can we train 3D deep models based on 2D supervision?**

DreamFusion!



Create 3D models that look like good images when rendered!

=> Last lectures: Differentiable Rendering