Generative Deep Learning

Naeemullah Khan
naeemullah.khan@kaust.edu.sa

جامعة الملك عبدالله
للعلوم والتقنية
King Abdullah University of
Science and Technology

KAUST Academy
King Abdullah University of Science and Technology

June 1, 2023

Introduction

- Can you imagine an image of dog?
- Now, can you imagine an image of that dog driving a car?
- How are we able to this?
- We excel at extracting knowledge from the data we observe and perform complex reasoning based on it

Introduction (cont.)

How can we generate such images/data through AI

What we would need? Let's think about it step by step.

- Some prior data
- A model which will "learn" the data
- A method through which the model will learn
- A method to generate new data from the trained model

This what generative deep learning is all about and what we will be covering in this course.

Introduction (cont.)

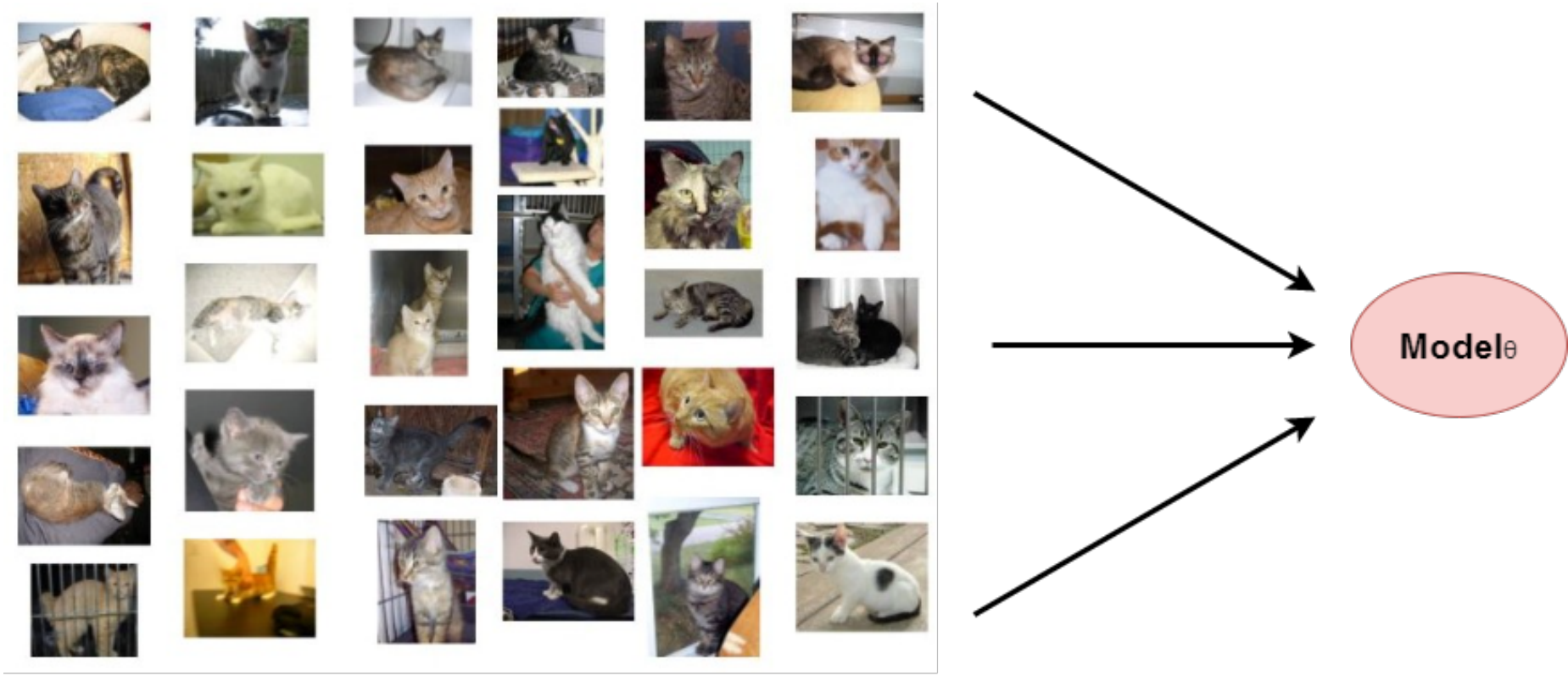


Figure 2: The model learns the cat images data

Introduction (cont.)

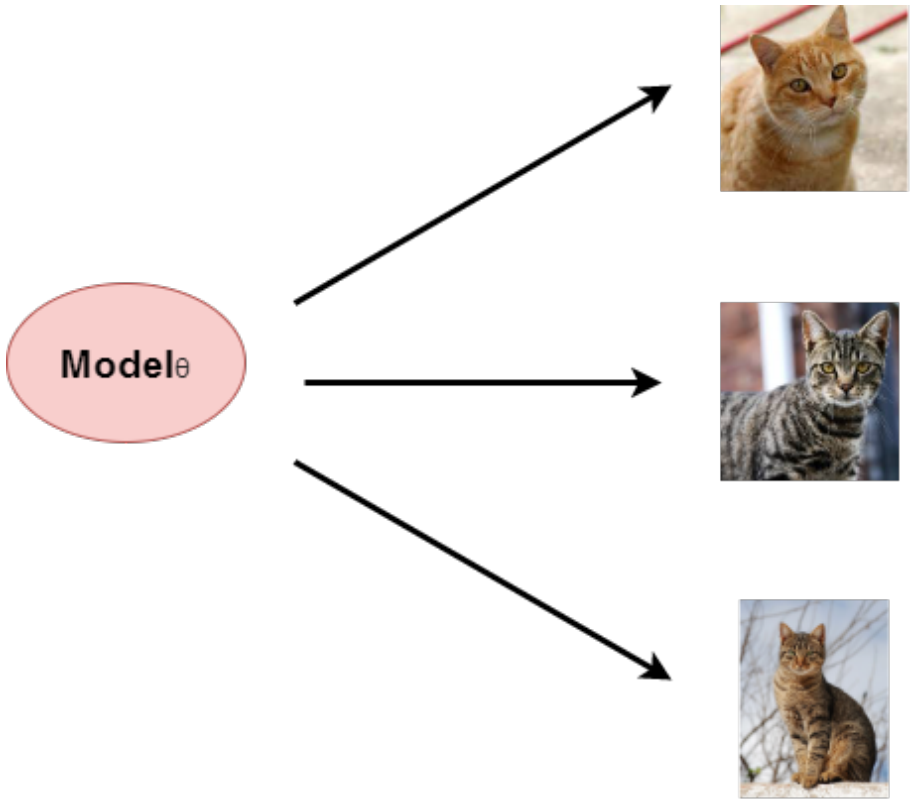


Figure 3: The trained model generates new cat images

Statistical Generative Models

Statistical generative models are generative models that are learned from data.

- Data $\mathcal{X} \sim \mathcal{P}(\mathcal{X})$
- Model $\theta \sim \mathcal{P}(\theta)$
- Optimize model s.t. $\mathcal{P}(\theta) \sim \mathcal{P}(\mathcal{X})$
- Generate $\mathcal{X}_{new} \sim \mathcal{P}(\theta)$

Discriminative vs. Generative

Discriminative Models

- E.g.: Classify between cat and dog images
- Doesn't really need to learn data distribution $\mathcal{P}(\mathcal{X})$
- Goal is to learn $\mathcal{P}(\mathcal{Y} = \text{dog}|\mathcal{X} = x)$

Generative Models

- E.g.: Generate new cat images
- Need to learn data distribution $\mathcal{P}(\mathcal{X})$
- Goal is to maximize $\mathcal{P}(\mathcal{X} = x_{new})$

Progress on Face Generation

Figure 4: Progress in face generation over the years[1]

[1] https://twitter.com/tamaybes/status/1450873331054383104

Completing Incomplete Content

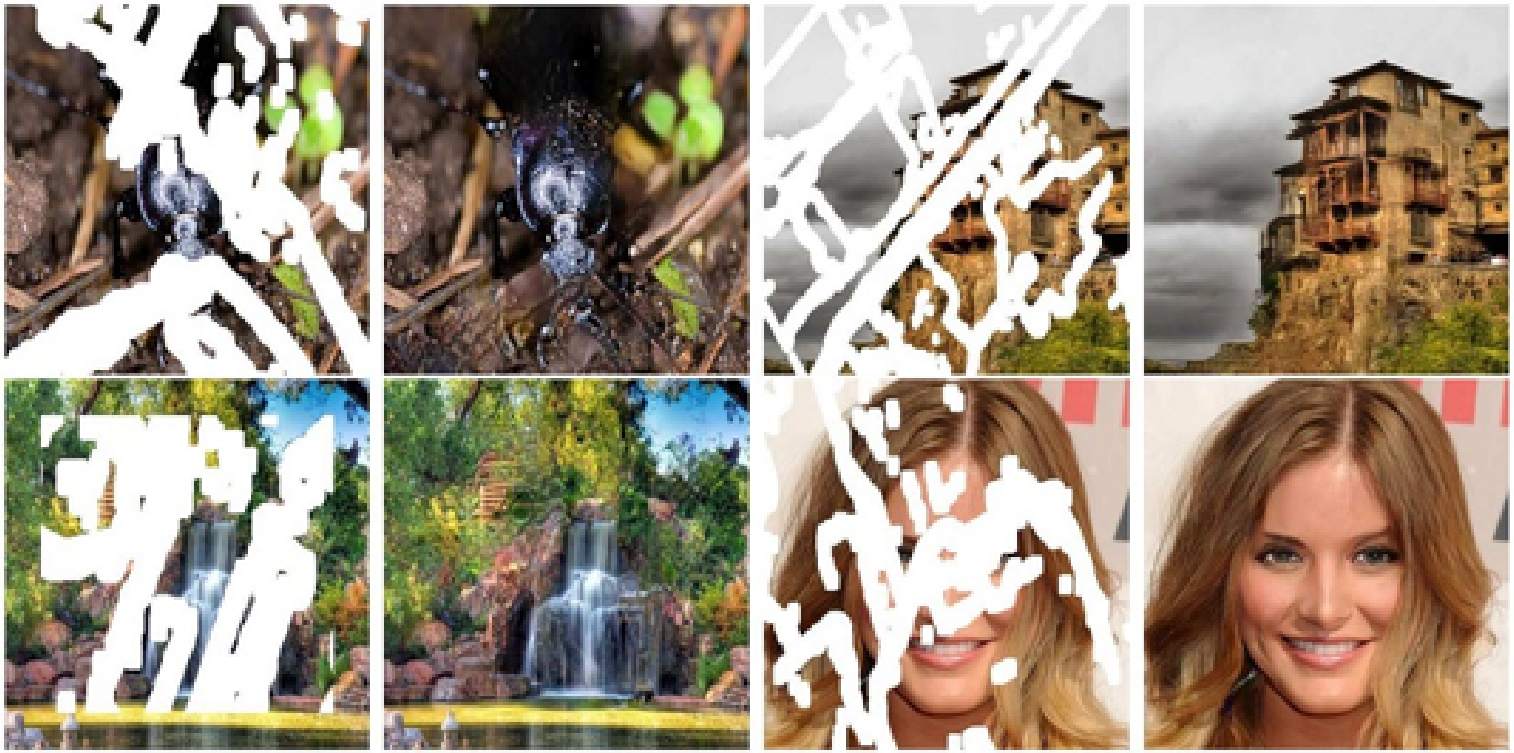


Figure 5: Complete missing patches in image

Completing Incomplete Content (cont.)

Figure 6: Improve image quality

Completing Incomplete Content (cont.)

Figure 7: Color b/w images

Completing Incomplete Content (cont.)

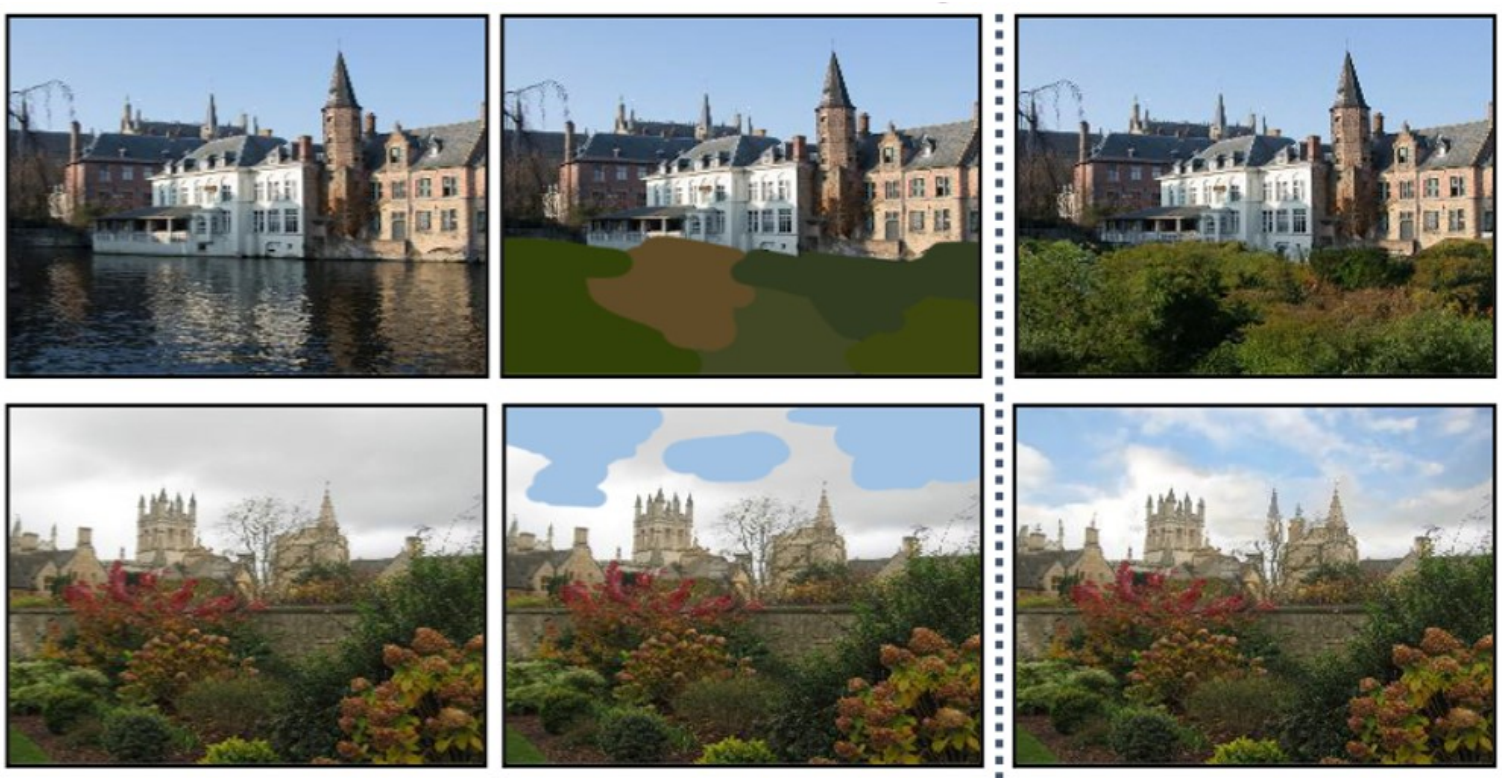


Figure 8: Editing based on strokes in image

Completing Incomplete Content (cont.)

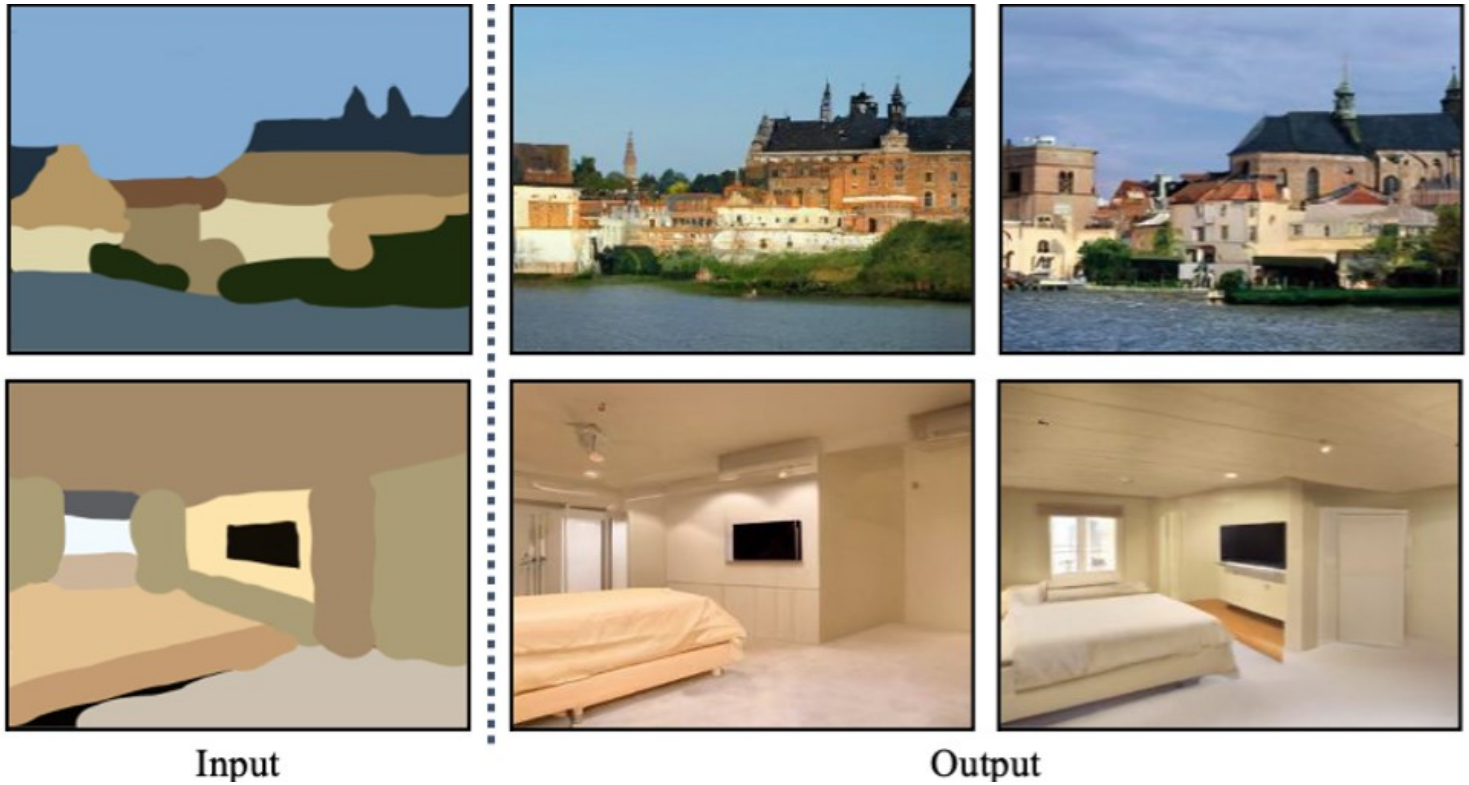


Figure 9: Converting strokes to proper images

Text Generation

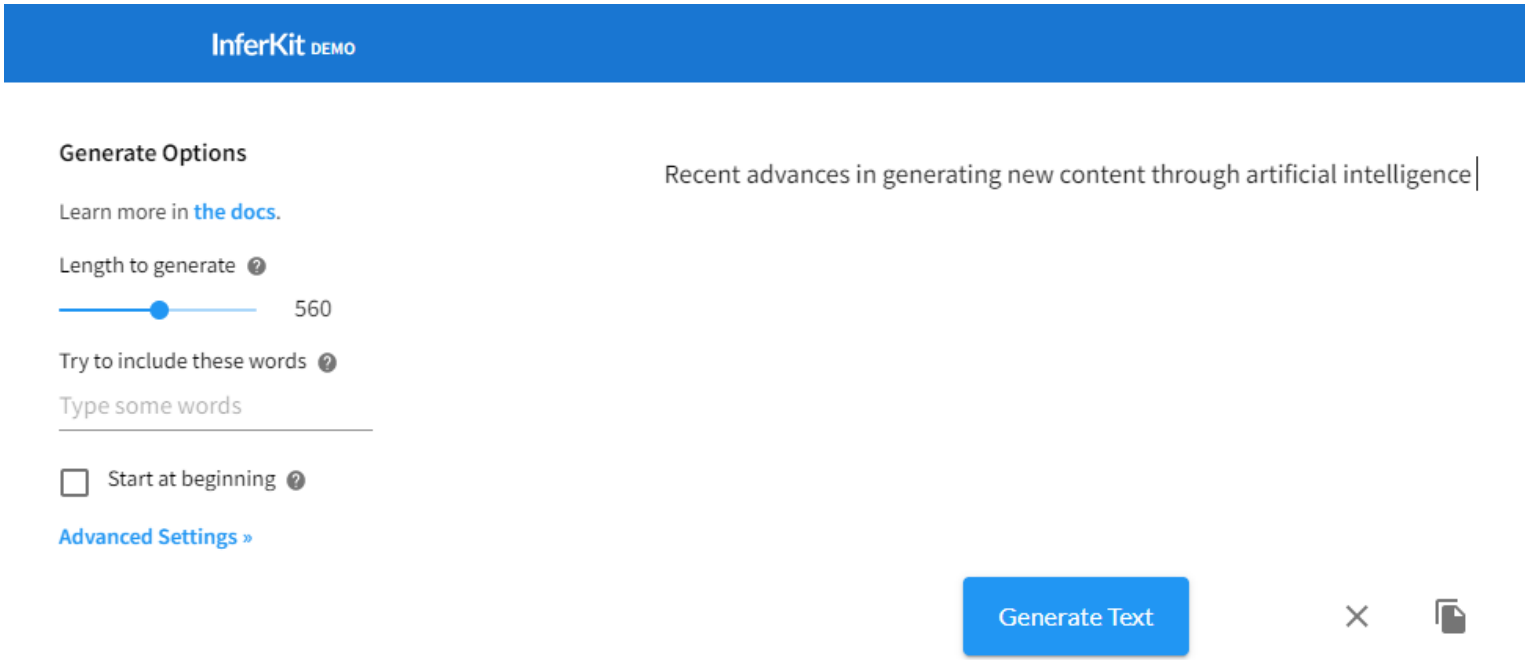


Figure 10: Completing Text. https://app.inferkit.com/demo

Text Generation (cont.)

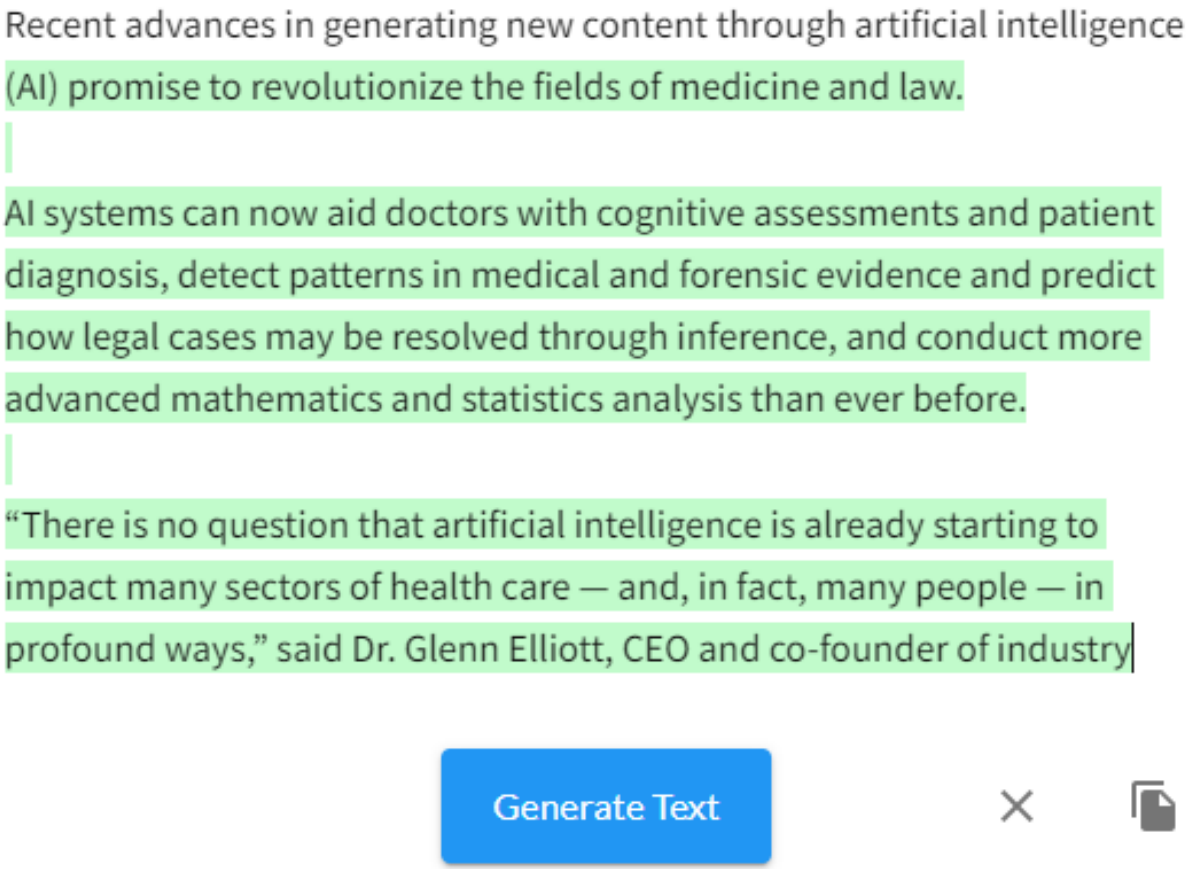


Figure 11: Completing Text. https://app.inferkit.com/demo

Machine Translation

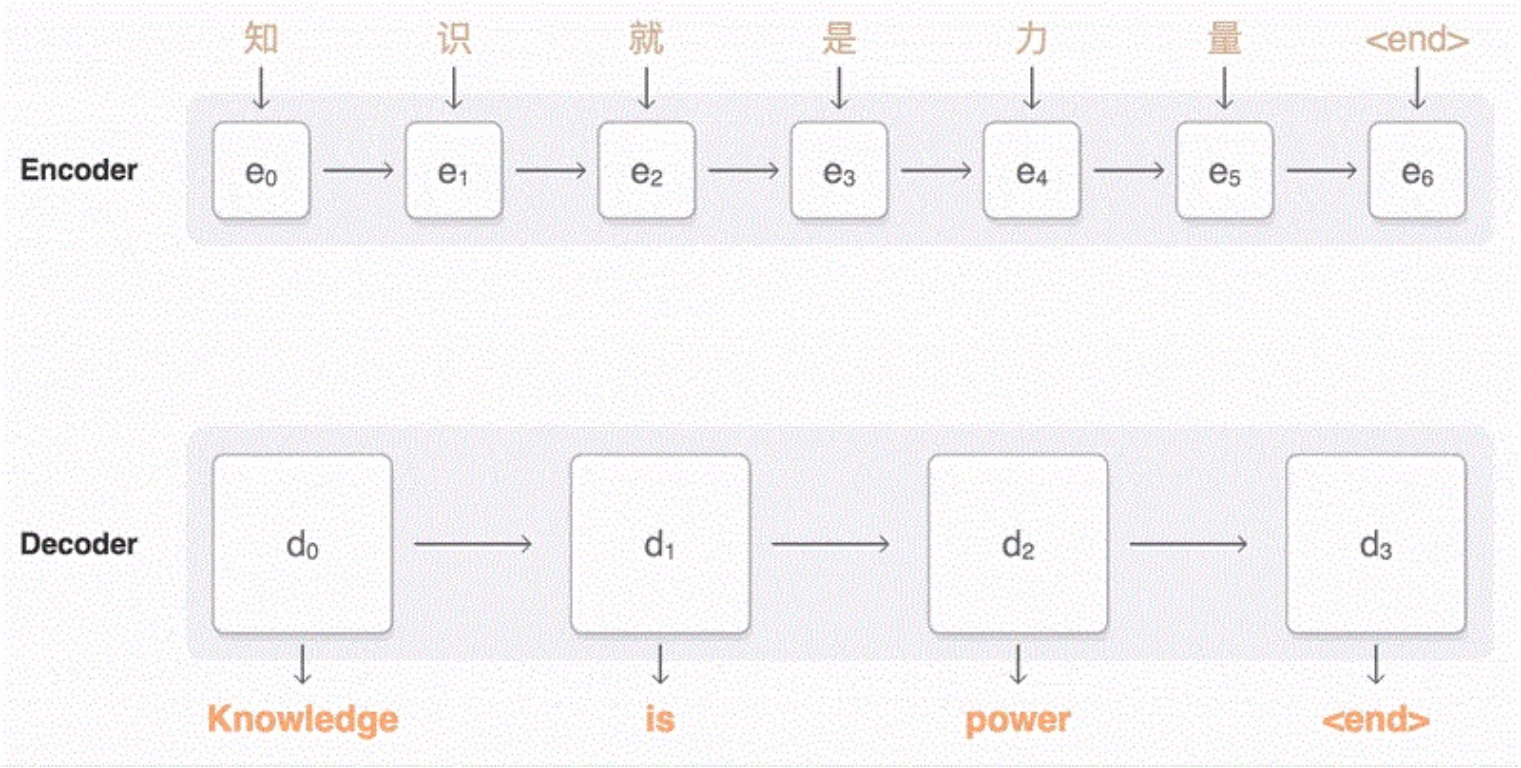


Figure 12: Language translation

Code Generation

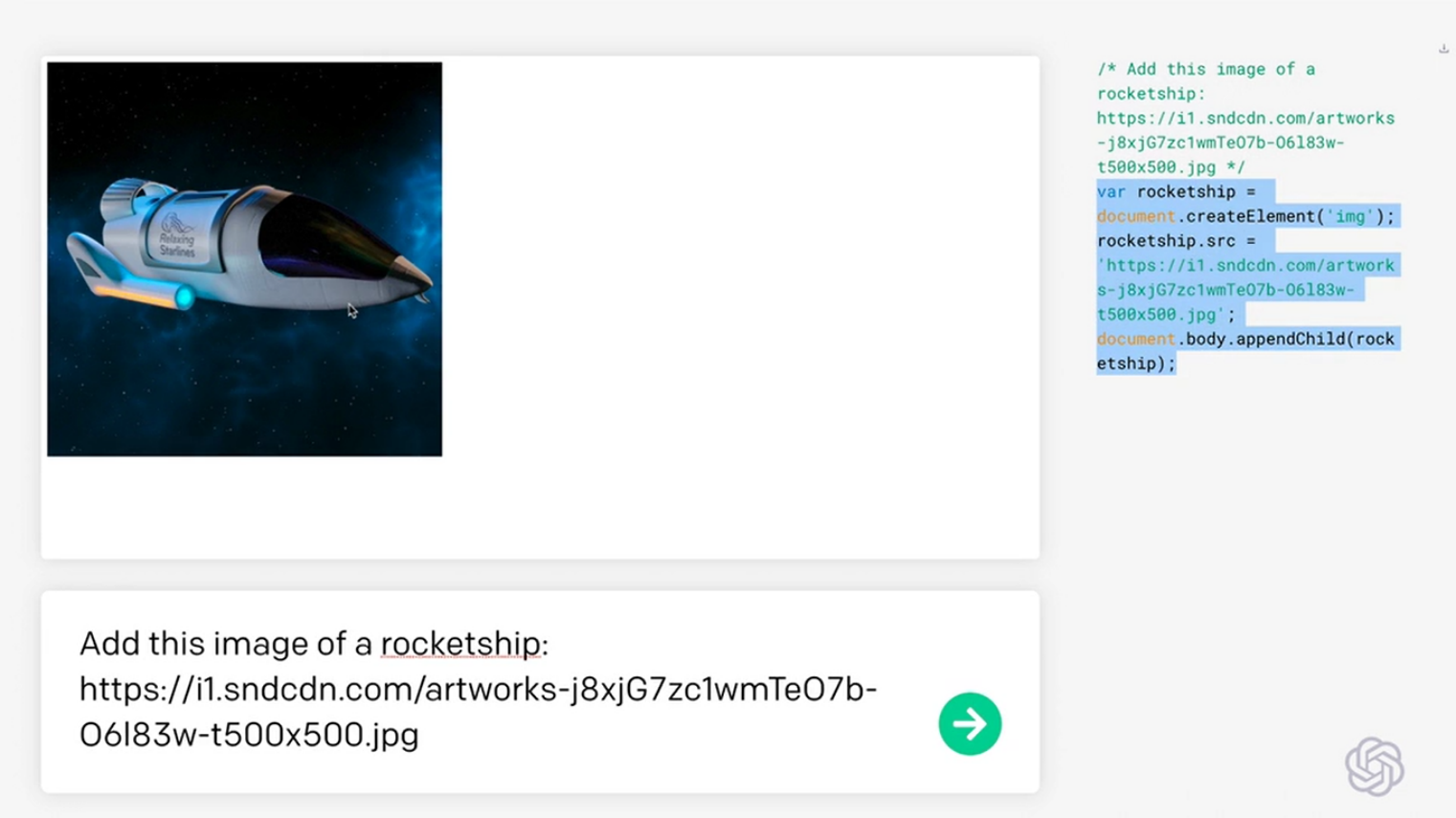


Figure 13: Code Generation. https://openai.com/blog/openai-codex/

Learning a Generative Model

For a given dataset of dog images, we want to learn a probability distribution $\mathcal{P}_{theta}(x)$ over images x such that

- **Generation:** If we sample $x_{new} \sim \mathcal{P}_{theta}(x)$, it should look like a dog image (sampling)
- **Density Estimation:** $\mathcal{P}_{theta}(x)$ should be high if x is cat image and low otherwise (anomaly detection)
- **Unupservised Representation Learning:** We should be able to learn what these images have in common e.g., ears, tail, etc. (features)

Let's Dive Deeper

- How do we represent the distribution of data $\mathcal{P}_{data}(x)$?
- Let's consider a simple case binarized MNIST digits dataset.
- We have an image x of 28 x 28. Flattening it gives us a vector of length 784. Each pixel can be either 0 or 1. $x \in \{0, 1\}^{784}$
- Now, we need to learn the joint distribution $\mathcal{P}(x) = \mathcal{P}(x_1, x_2, \cdots, x_{784})$

KL Divergence

- How do we evaluate "closeness" between model and data distribution?
- **Kullback-Leibler divergence** (KL-divergence) is one possibility:

$$D(\mathcal{P}_{data}||\mathcal{P}_\theta) = E_{x\sim\mathcal{P}_{data}}\left[log\left(\frac{\mathcal{P}_{data}(x)}{\mathcal{P}_\theta(x)}\right)\right] = \sum_x \mathcal{P}_{data}(x) log\frac{\mathcal{P}_{data}(x)}{\mathcal{P}_\theta(x)}$$

- $D(\mathcal{P}_{data}||\mathcal{P}_\theta)$ iff the two distributions are the same.
- It measures the "compression loss" (in bits) of using $\mathcal{P}_\theta$ instead of $\mathcal{P}_{data}$.

Expected Log-Likelihood

- We can simplify this somewhat:

$$\begin{aligned} D(\mathcal{P}_{data}||\mathcal{P}_\theta) &= E_{x\sim\mathcal{P}_{data}}\left[log\left(\frac{\mathcal{P}_{data}(x)}{\mathcal{P}_\theta(x)}\right)\right] \\ &= E_{x\sim\mathcal{P}_{data}}[log\mathcal{P}_{data}(x)] - E_{x\sim\mathcal{P}_{data}}[log\mathcal{P}_\theta(x)] \end{aligned} \tag{1}$$

- The first term does not depend on $\mathcal{P}_\theta$. Then, *minimizing* KL divergence is equivalent to *maximizing* the **expected log-likelihood**

$$\begin{aligned} \arg\min_{\mathcal{P}_\theta} D(\mathcal{P}_{data}||\mathcal{P}_\theta) &= \arg\min_{\mathcal{P}_\theta} -E_{x\sim\mathcal{P}_{data}}[log\mathcal{P}_\theta(x)] \\ &= \arg\max_{\mathcal{P}_\theta} E_{x\sim\mathcal{P}_{data}}[log\mathcal{P}_\theta(x)] \end{aligned} \tag{2}$$

  - Asks that $\mathcal{P}_\theta$ assign high probability to instances sampled from $\mathcal{P}_{data}$, so as to reflect the true distribution
  - Because of log, samples x where $\mathcal{P}_\theta(x) \approx 0$ weigh heavily in objective

- Although we can now compare models, since we are ignoring $H(\mathcal{P}_{data})$, we don't know how close we are to the optimum.
- Problem: In general we do not know $\mathcal{P}_{data}$

Maximum Likelihood

- Approximate the expected log-likelihood

$$E_{x \sim \mathcal{P}_{data}}[log \mathcal{P}_\theta(x)]$$

with the empirical log-likelihood:

$$E_{\mathcal{D}} = \frac{1}{\mathcal{D}} \sum_{x \in \mathcal{D}} log \mathcal{P}_\theta(x)$$

- **Maximum likelihood learning** is then:

$$\arg\max_{\mathcal{P}_\theta} \frac{1}{\mathcal{D}} \sum_{x \in \mathcal{D}} log \mathcal{P}_\theta(x)$$

Autoencoders

- Family of neural networks for which the input is the same as the output. They work by compressing the input into a latent-space representation, and then reconstructing the output from this representation.
- The idea is project the input into a latent space and then reconstruct the input from that latent space representation
- Consist of two parts: Encoder and decode.
  - Encoder projects the input to a latent space Z.
  - Decoder takes the encoded embedding vector and reconstructs the input from it.
  - We also use altered versions of input as output which can be even more interesting.

Autoencoders (cont.)

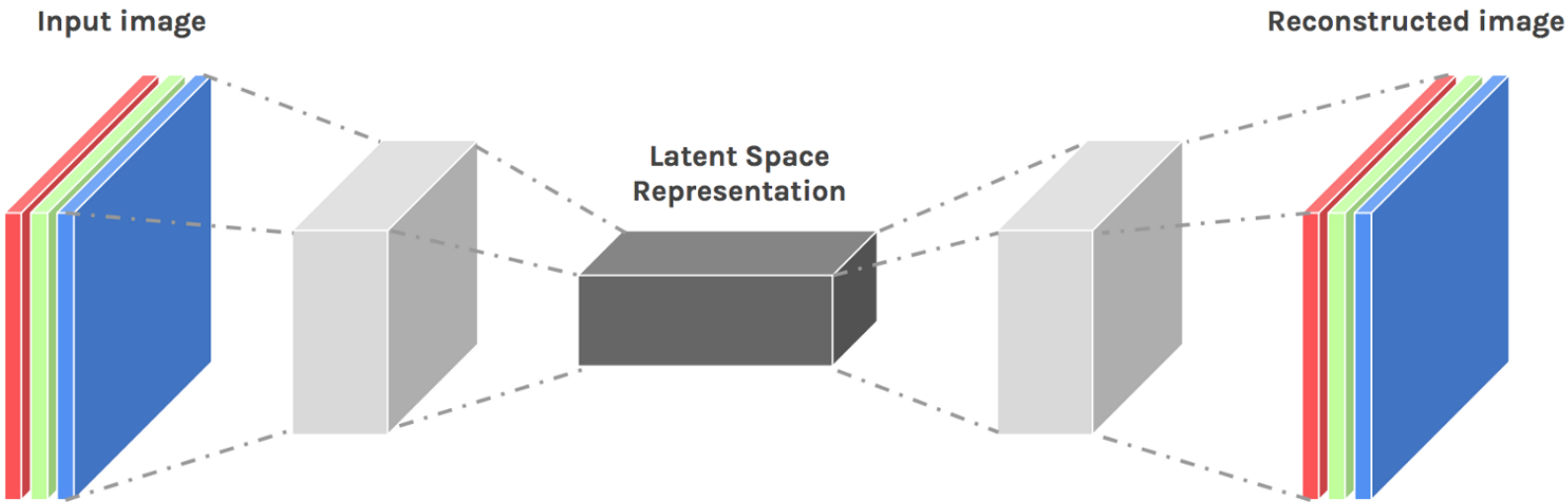


Figure 14: Autoencoder architecture

Autoencoders (cont.)

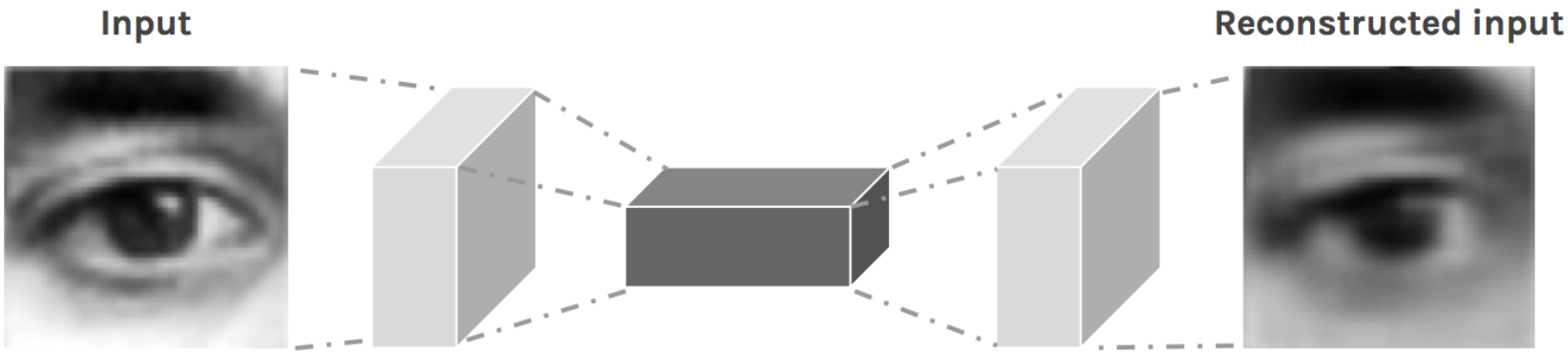


Figure 15: Sample Autoencoder

Autoencoders - Interactive Demo

https://douglasduhaime.com/posts/visualizing-latent-spaces.html

Autoencoders as generative models

- Autoencoders project data into a latent space Z.
- What if we sample a new embedding vector from Z and then have the decoder reconstruct the image from it?
- **Does not work**. Autoencoders just learn a function that maps input to output. The learned latent space is too discontinous to work as a generative model.

Autoencoders as generative models (cont.)

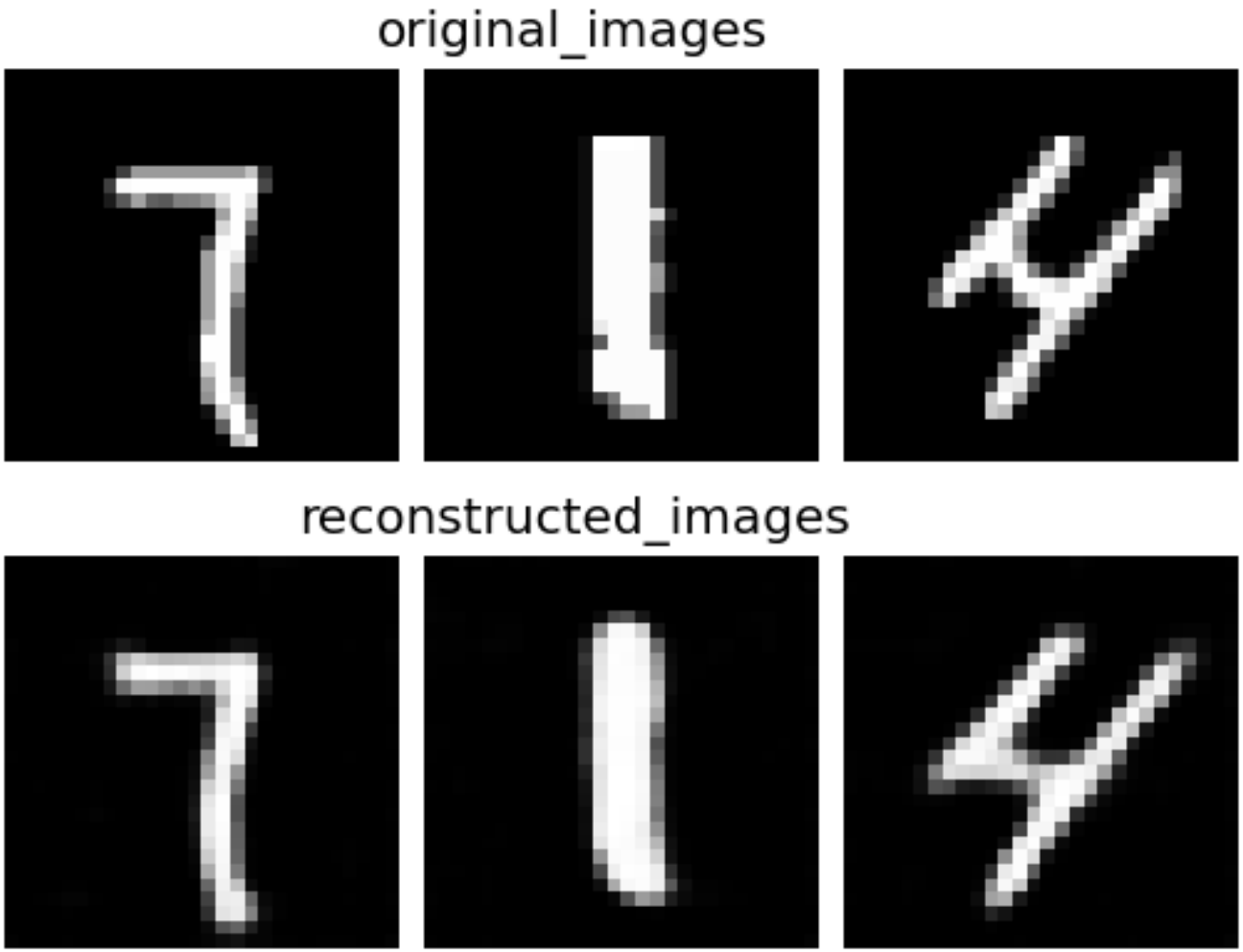


Figure 16: Image reconstruction with autoencoder trained on MNIST digits

Autoencoders as generative models (cont.)

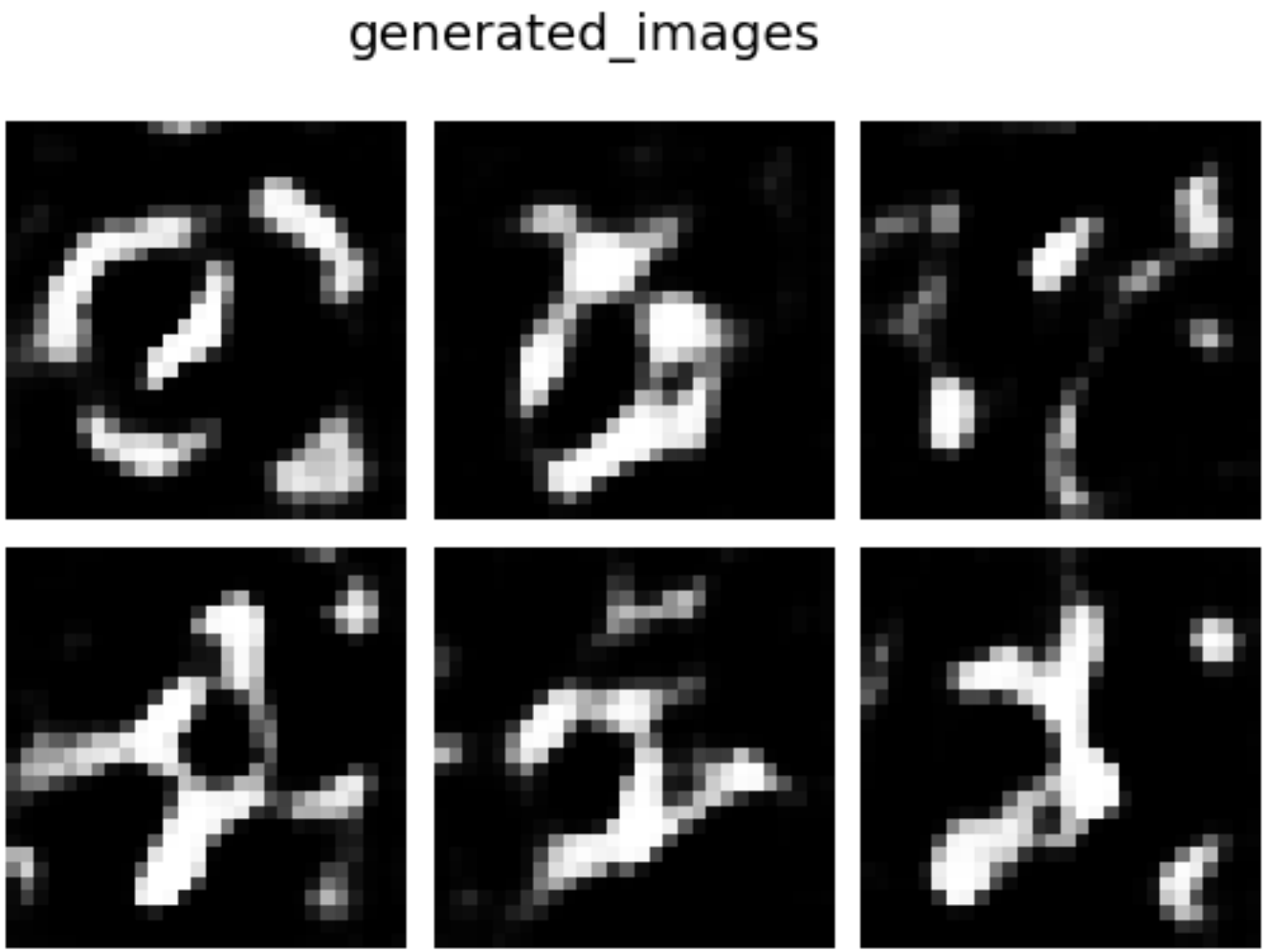


Figure 17: Image generation with autoencoder trained on MNIST digits. Encoding vector sampled from latent space Z and the passed to decoder.

Autoencoders - Applications

- While autoencoders themselves have very low generative power, we will soon talk about a type of autoencoders called **Variational Autoencoders** which are specifically designed for generative modeling.
- Other use cases of Autoencoders include:
  - Data encoding and dimensionality reduction
  - Image denoising and super-resolution
  - Image completion
  - Image colorization

Autoencoders - Applications (cont.)

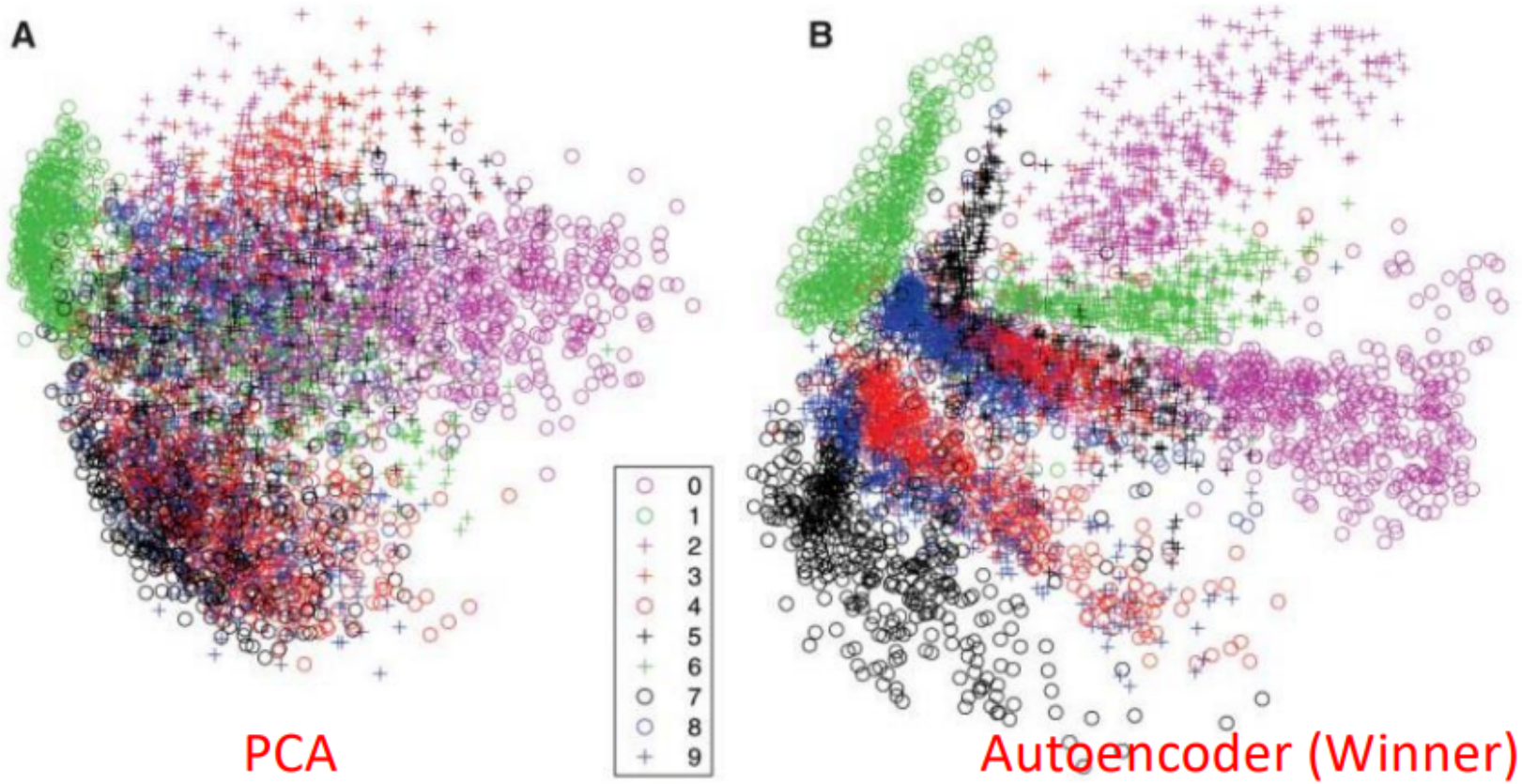


Figure 18: t-SNE visualization on MNIST digits dataset. PCA vs. Autoencoders. The image vector is projected into $\mathbb{R}^2$.

Autoencoders - Applications (cont.)

Figure 19: Image super-resolution using Autoencoders

Autoencoders - Applications (cont.)

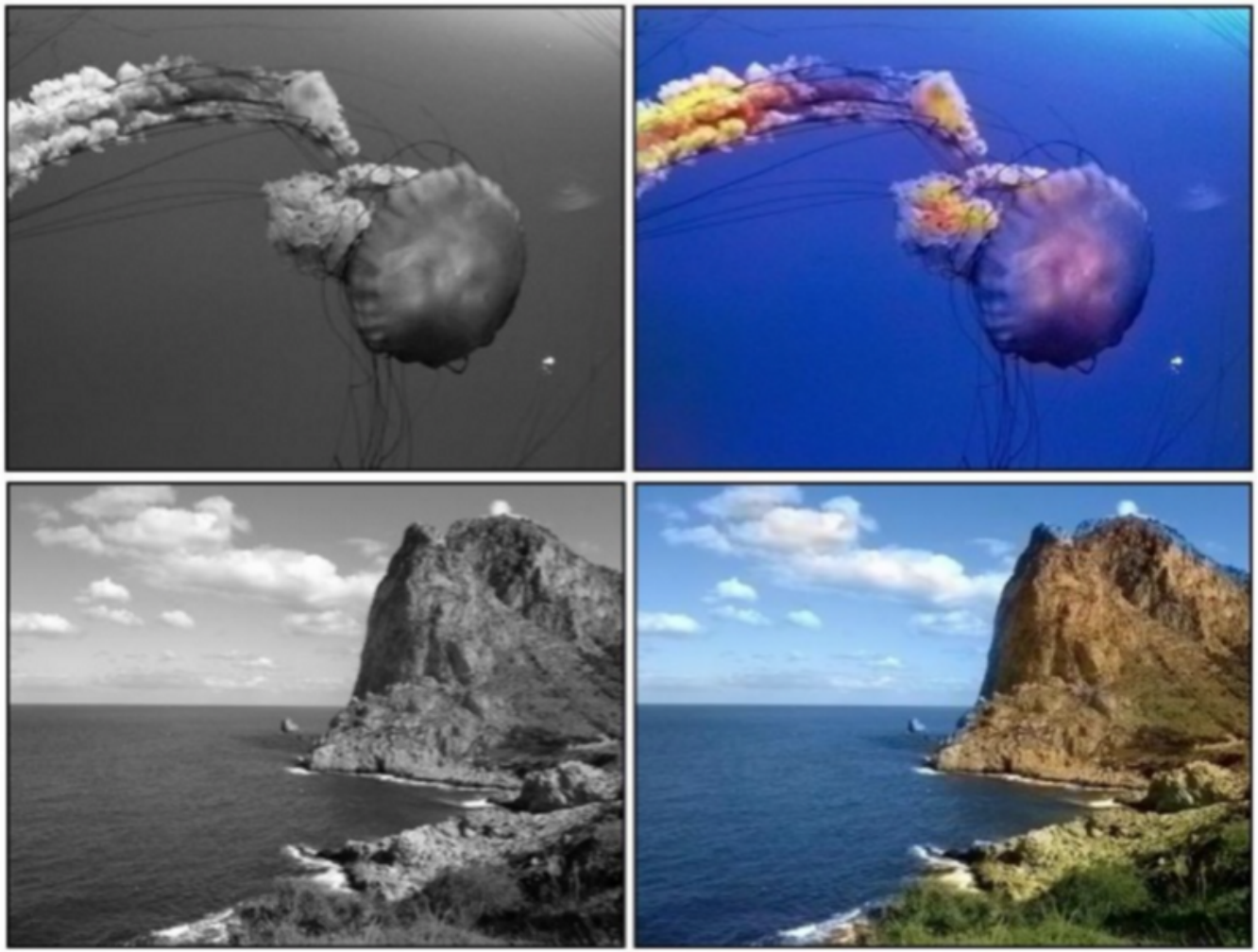


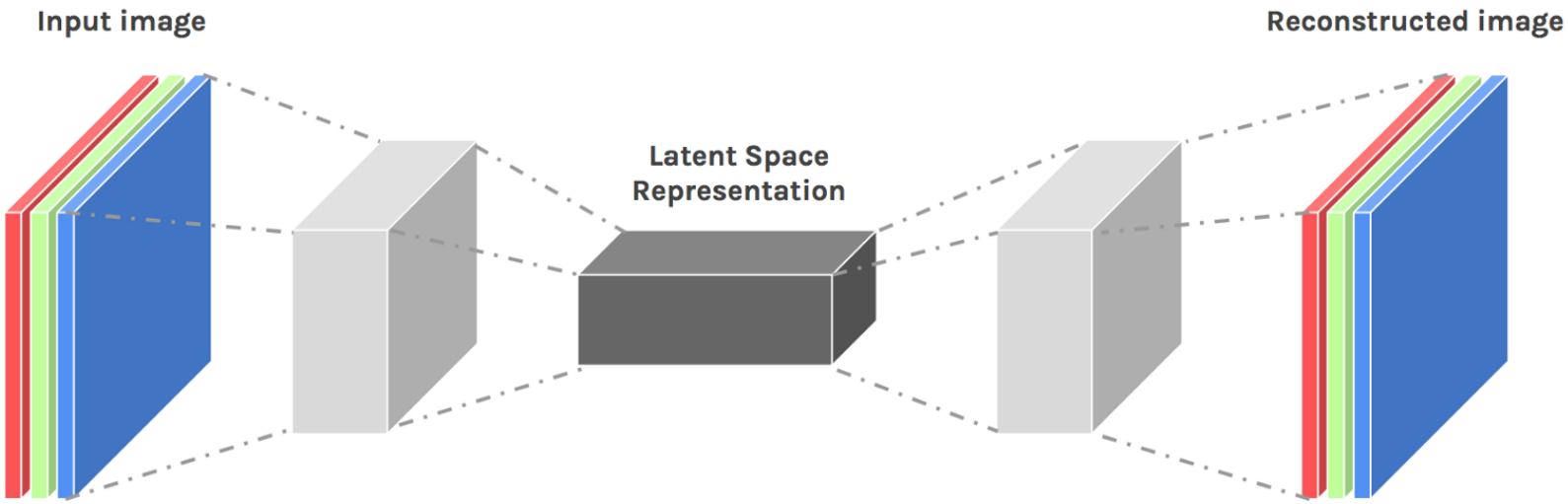
Figure 20: Image colorization using Autoencoders

References

Reference Slides

- Fei-Fei Li "Generative Deep Learning" CS231
- Hao Dong "Deep Generative Models"

Recap - Autoencoders

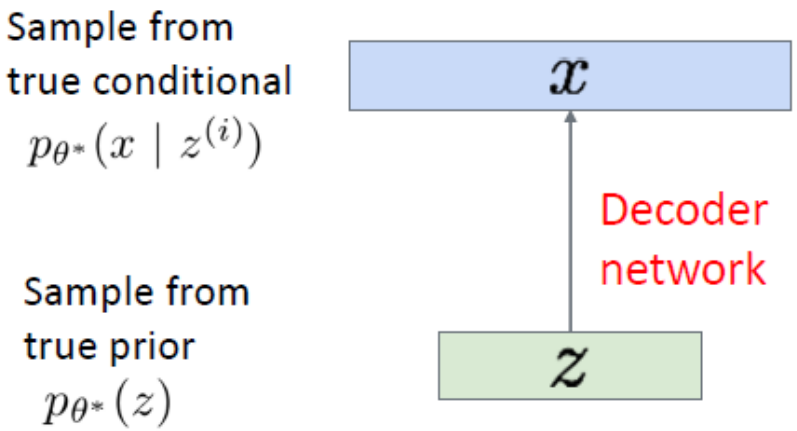


Variational Autoencoders

- Autoencoders don't work as generative models because the latent space Z is too discontinous.
- **Solution**: Let's rectify that.
- **Variational Autoencoders**: Probabilistic spin on autoencders - will let us sample from the model to generate data.

Variational Autoencoders (cont.)

- We want a generative model, which given a prior **z** outputs a new sample from data.
- To make the latent space Z continuous, let's choose it to be Gaussian
- So now, we want to estimate the true parameters θ^* of this generative model.



Variational Inference

- Now, how to train this model?
- How about following the same strategy as in FVSBNs? Learn model parameters to maximize the likelihood of training data.

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

- But there is a problem here. It is Intractible to compute $p(x|z)$ for every z!
- Intuitively, need to figure out which z corresponds to each x in the dataset, but such mapping is unknown.
- This also makes posterior density $p(z|x)$ intractable because it depends on $p_\theta(x)$

$$p(z|x) = \frac{p_\theta(x|z) p_\theta(z)}{p_\theta(x)}$$

Variational Inference (cont.)

- **Solution**
  - Let's approximate $p(z|x)$ with another distribution by another distribution $q(z)$
  - If $q(z)$ is a tractable distribution e.g. Gaussian distribution
  - We can adjust parameters of $q(z)$ and make it as close to $p(z|x)$
  - **Goal**: min $KL(q||p)$

Variational Inference (cont.)

$$
\begin{aligned}
KL(q(z)||p(z|x)) &= -\sum q(z) log \frac{p(z|x)}{q(z)} \\
&= -\sum q(z) log \frac{\frac{p(x,z)}{p(x)}}{q(z)} \\
&= -\sum q(z) log \left(\frac{p(x,z)}{q(z)} \frac{1}{p(x)} \right) \\
&= -\sum q(z) \left[log \frac{p(x,z)}{q(z)} + log \frac{1}{p(x)} \right] \\
&= -\sum q(z) \left[log \frac{p(x,z)}{q(z)} - log\, p(x) \right] \\
&= -\sum_z q(z) log \frac{p(x,z)}{q(z)} + log\, p(x) \sum_z q(z) \\
&= -\sum_z q(z) log \frac{p(x,z)}{q(z)} + log\, p(x) \quad \because \sum_z q(z) = 1
\end{aligned}
$$

Variational Inference (cont.)

- $$KL(q(z)||p(z|x) = -\sum_z q(z) log \frac{p(x,z)}{q(z)} + log\ p(x)$$
- We can also write above equation as:

$$log\ p(x) = KL(q(z)||p(z|x) + \sum_z q(z) log \frac{p(x,z)}{q(z)}$$

Variational Inference (cont.)

- Given x , $log\ p(x)$ is a constant
- $KL(q(z)||p(z|x)$ is the quantity we wanted to minimize
- Assume $L = \sum_z q(z) log \frac{p(x,z)}{q(z)}$, then

$$\text{constant} = KL + L$$

$$L \leq log\ p(x) \quad \because kl \geq 0$$

- **Instead of minimizing KL we can maximise L**

Variational Lower Bound L

Looking at Lower bound L

$$
\begin{aligned}
L &= \sum_z q(z) log \frac{p(x,z)}{q(z)} \\
&= \sum_z q(z) log \frac{p(x|z)p(z)}{q(z)} \\
&= \sum_z q(z) \left[log\, p(x|z) + log \frac{p(z)}{q(z)} \right] \\
&= \underbrace{\sum_z q(z) log\, p(x|z)}_{\text{Expectation } E_{q(z)}(log\, p(x|z))} + \underbrace{\sum_z q(z) log \frac{p(z)}{q(z)}}_{-KL(q(z)||p(z)}
\end{aligned}
$$

So,

$$L = E_{q(z)}(log\, p(x|z)) - KL(q(z)||p(z))$$

Variational Lower Bound L (cont.)

- $E_{q(z)}(log\ p(x|z))$ is conceptually reconstruction
- We can assume z to be Standard Normal Distribution

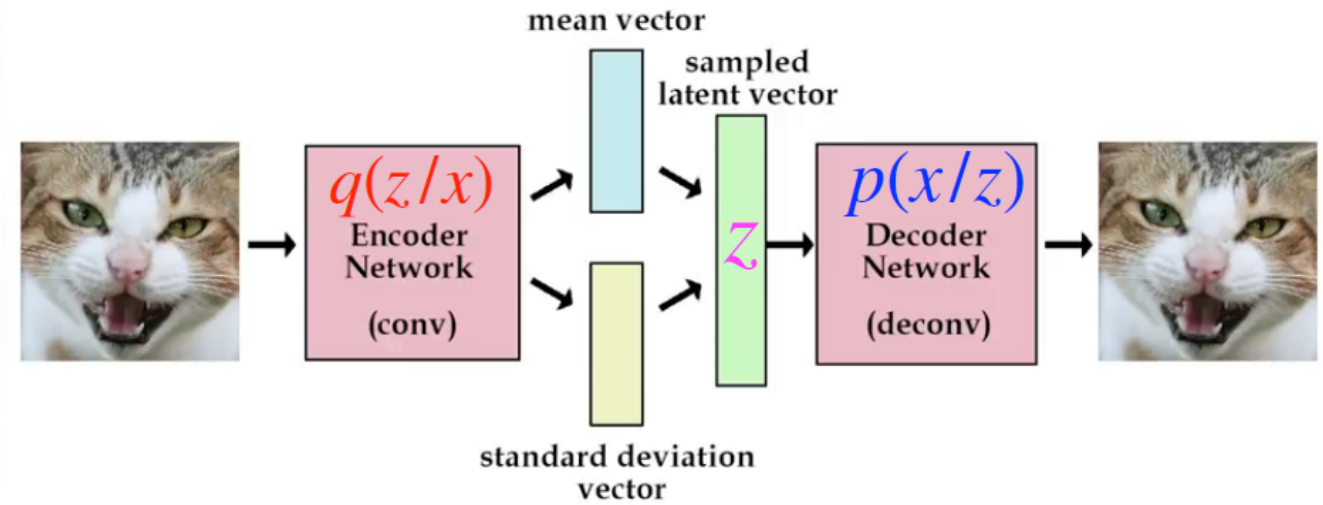
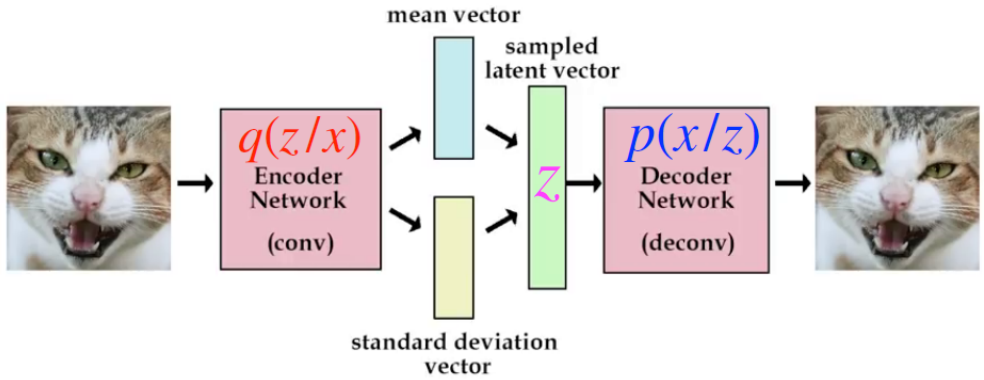


Figure 21: Variational Autoencoder Architecture

Variational Lower Bound L (cont.)



$$p(x|\hat{x}) = e^{-|x-\hat{x}|^2}$$

$$log\ e^{-|x-\hat{x}|^2} = -|x-\hat{x}|^2$$

$$L = E_{q(z)}(-|x-\hat{x}|^2) - KL(q(z)||p(z))$$

$$\min |x-\hat{x}| + KL(q(z|x)||\mathcal{N}(0,1))$$

$$\min |x-\hat{x}| - 0.5 * (1 + log\ \sigma^2 - \sigma^2 - \mu^2)$$

For full derivation of KL Loss, read here

Reparameterization Trick

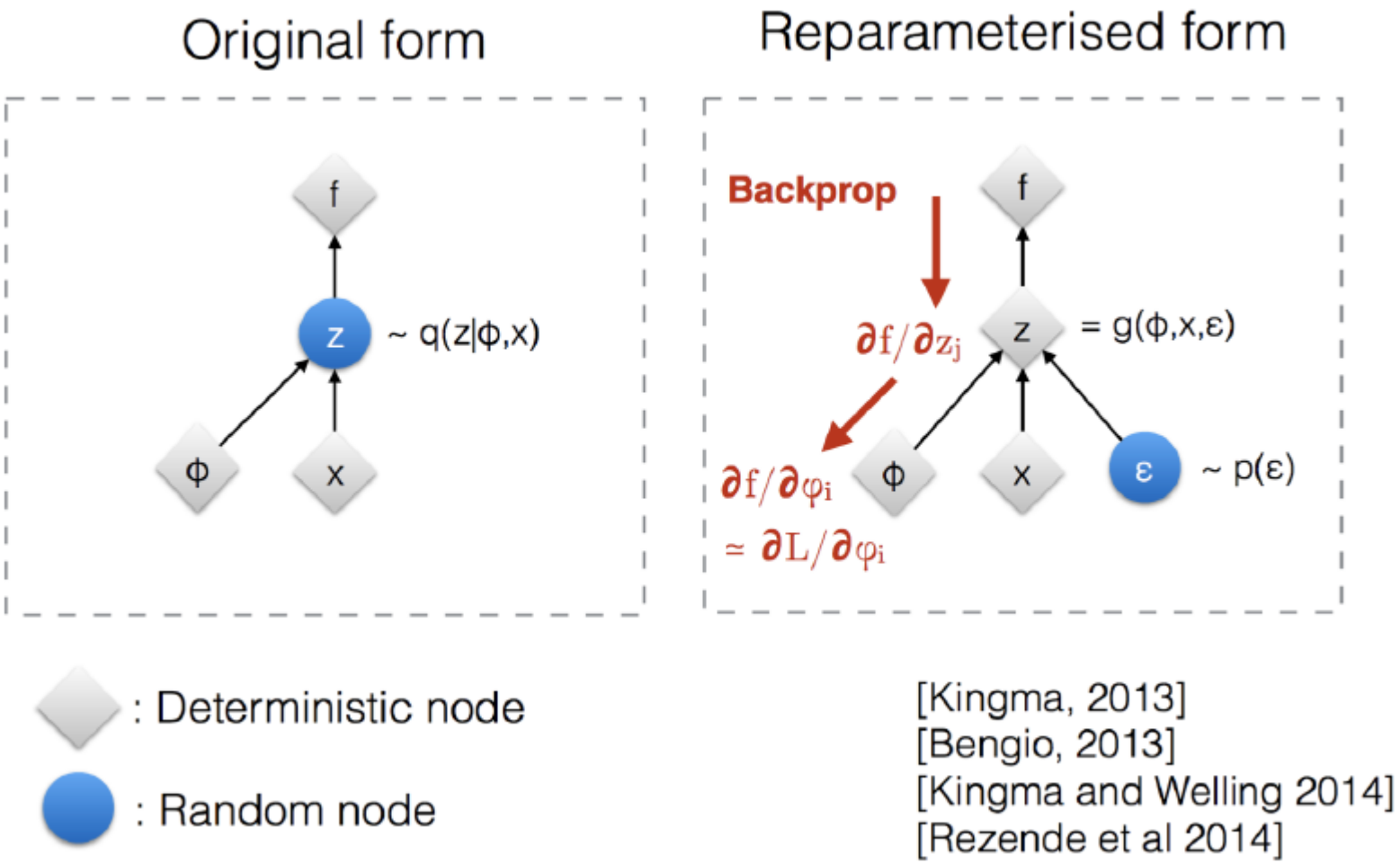


Figure 22: Reparameterization trick to make back propagation possible

Reparameterization Trick (cont.)

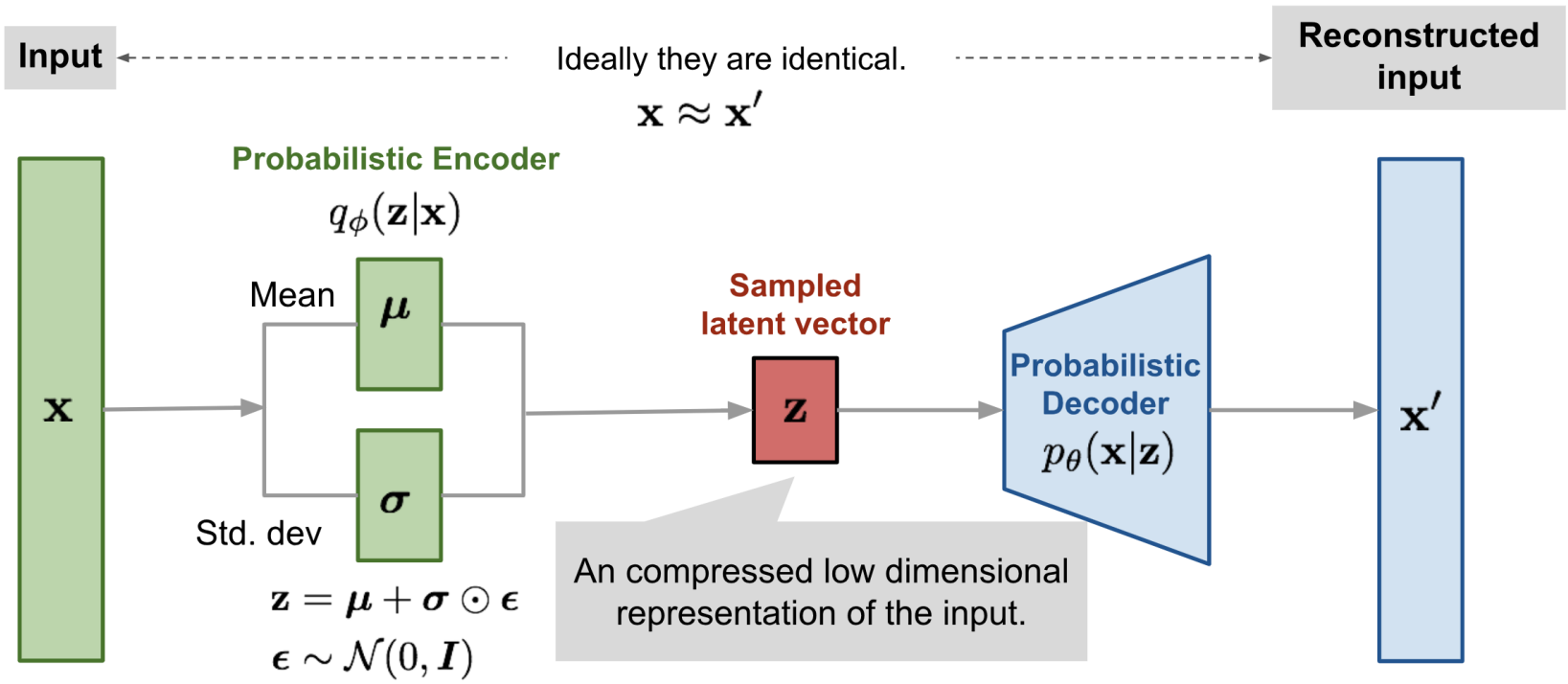


Figure 23: Variational Autoencoder with reparameterization trick

VAE - Results

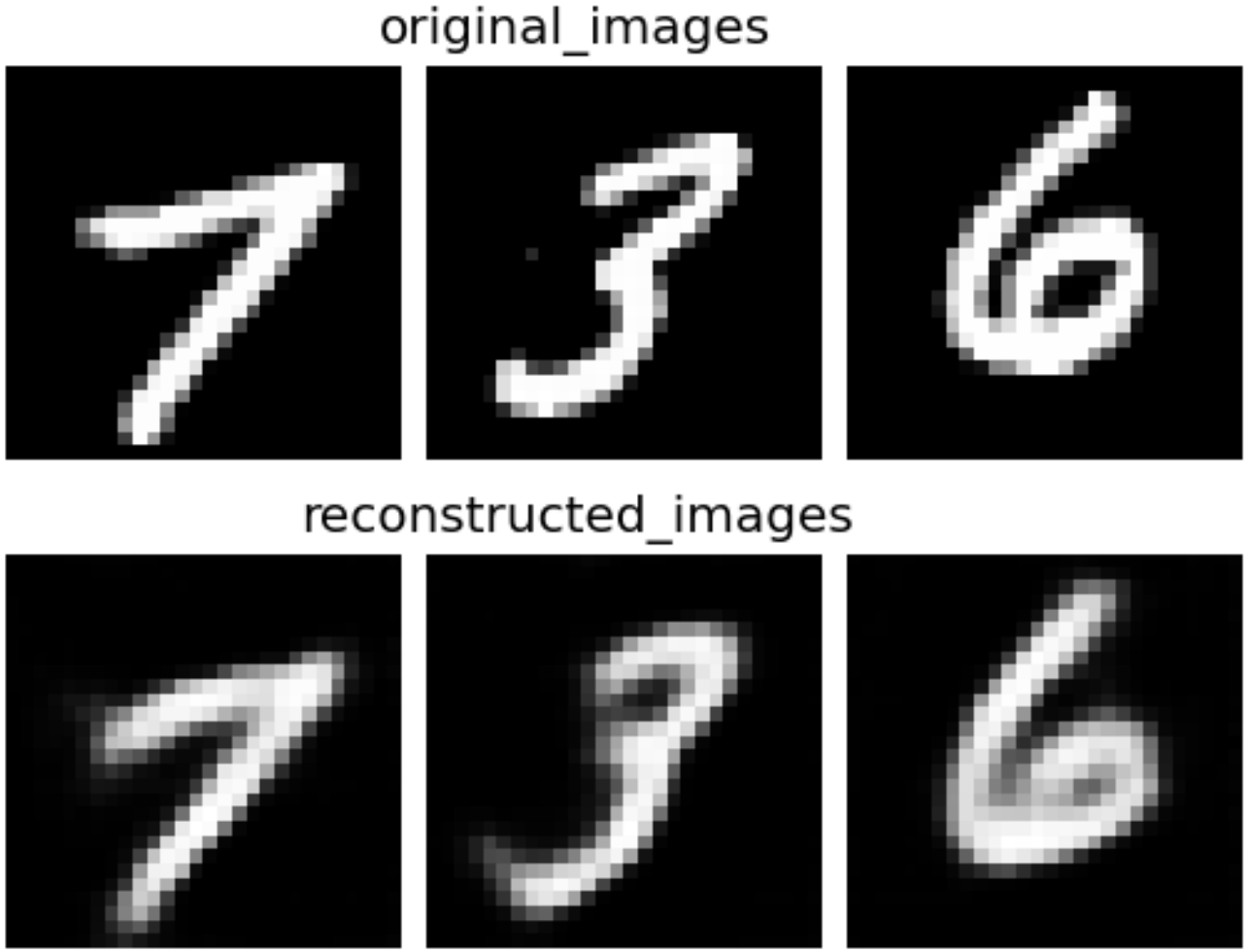


Figure 24: Image reconstruction with variational autoencoders on MNIST digits dataset

VAE - Results (cont.)

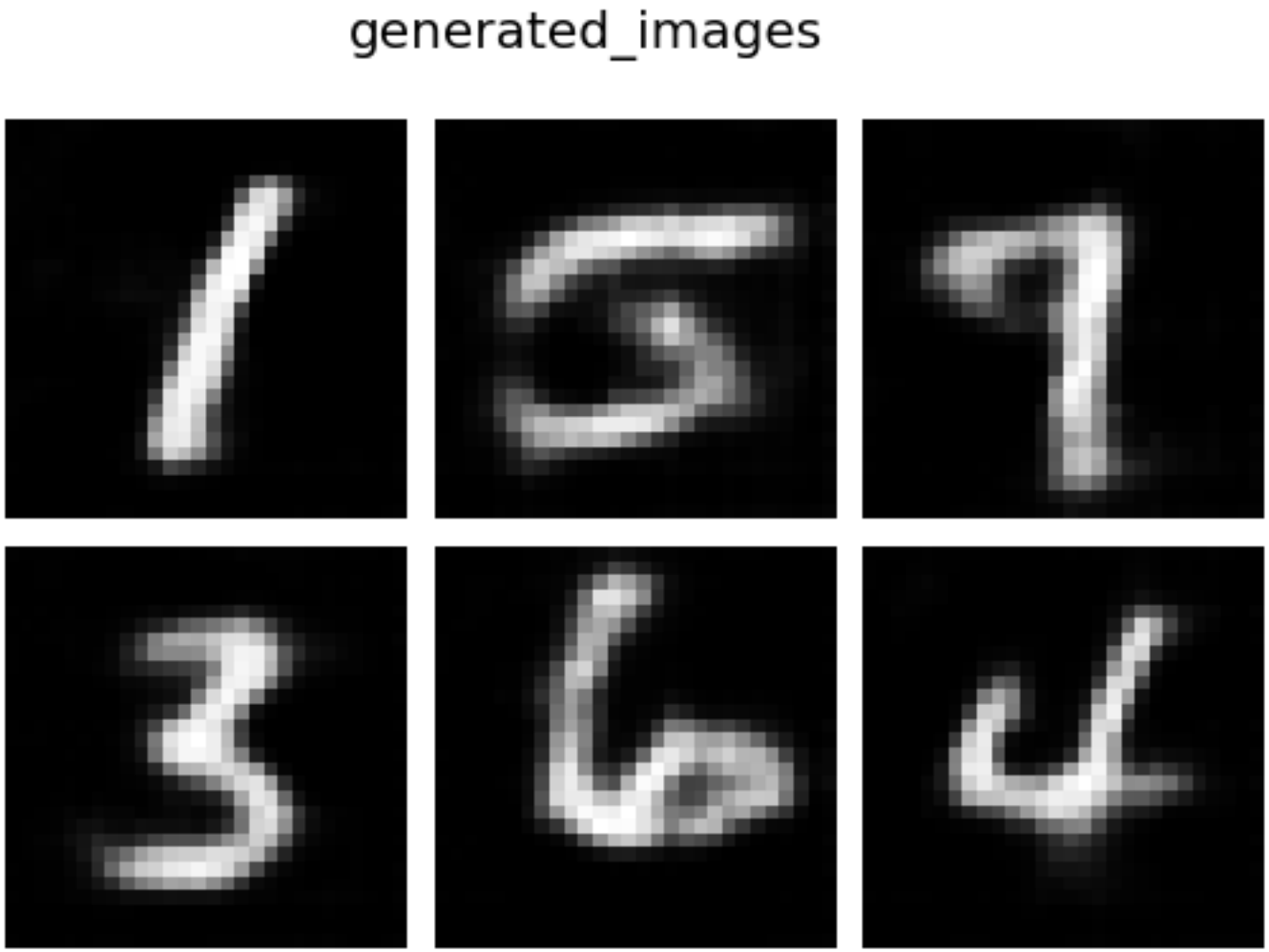


Figure 25: Image generation with variational autoencoders on MNIST digits dataset. Sample an encoding vector from $\mathcal{N}(0, 1)$ and passed it through decoder

VAE - Results (cont.)

Figure 26: Image generation with variational autoencoders on CIFAR-10 32x32 dataset

Disentangled Variational Autoencoders (β-VAEs)

- **Basic Idea**: Different neurons in latent space should be uncorrelated i.e. they all try to learn something different about input data.
- **Implementation**:

$$\mathcal{L}(\theta, \phi; x, z, \beta) = E_{q_\phi(z|x)}(log\ p_\theta(x|z)) - \beta\ KL(q_\phi(z|x)||p(z))$$

- Increasing the β is forcing variational autoencoder to encode the information in only few latent variables

Disentangled Variational Autoencoders (β-VAEs) (cont.)

Figure 27: Azimuthal rotation in β-VAEs and simple VAEs. β-VAEs produce more disentangled rotation whereas some other features also change in simple VAEs.

Variational Autoencoders Summary

- Add a probabilistic spin to Autoencoders to make them generative models
- Assume Z to be from Gaussian Distribution.
- But $p(z|x)$ is intractable.
- **Solution**: Approximate $p(z|x)$ with a Gaussian distribution $q(z)$.
- To minimize the KL divergence between them maximize the Evidence lower bound

$$L = \sum_z q(z) log \frac{p(x,z)}{q(z))}$$

- But image produced by variational autoencoders are blurry.

Normalizing Flow Models

- We continue on our quest for likelihood based generative model.
- So far, we have studied two different type of generative model.
- **Autoregressive Models**: $p_\theta(x) = \prod_{i=1}^{N} p_\theta(x_i|x_{<i})$
  - Provide tractable likelihoods
  - But have no direct mechanism for learning features
  - Slow generation process
- **Variational Autoencoders**: $p_\theta(x) = \int_z p_\theta(x|z)p_\theta(z)$
  - Has intractable marginal likelihood
  - Can learn feature representation
  - We optimize the lower bound instead of maximizing the likelihood ... we don't know the gap between them

Normalizing Flow Models (cont.)

- **Question**: Can we design a latent variable model with tractable likelihoods?
- **Answer**: Normalizing Flow Models
- They combine the best of both worlds, allowing both feature learning and tractable marginal likelihood estimation.
- **Key Idea**: we wish to map simple distributions (easy to sample and evaluate densities) to complex ones (learned via data) using **change of variables**.

The change of variables formula describe how to evaluate densities of a random variable that is a deterministic transformation from another variable.

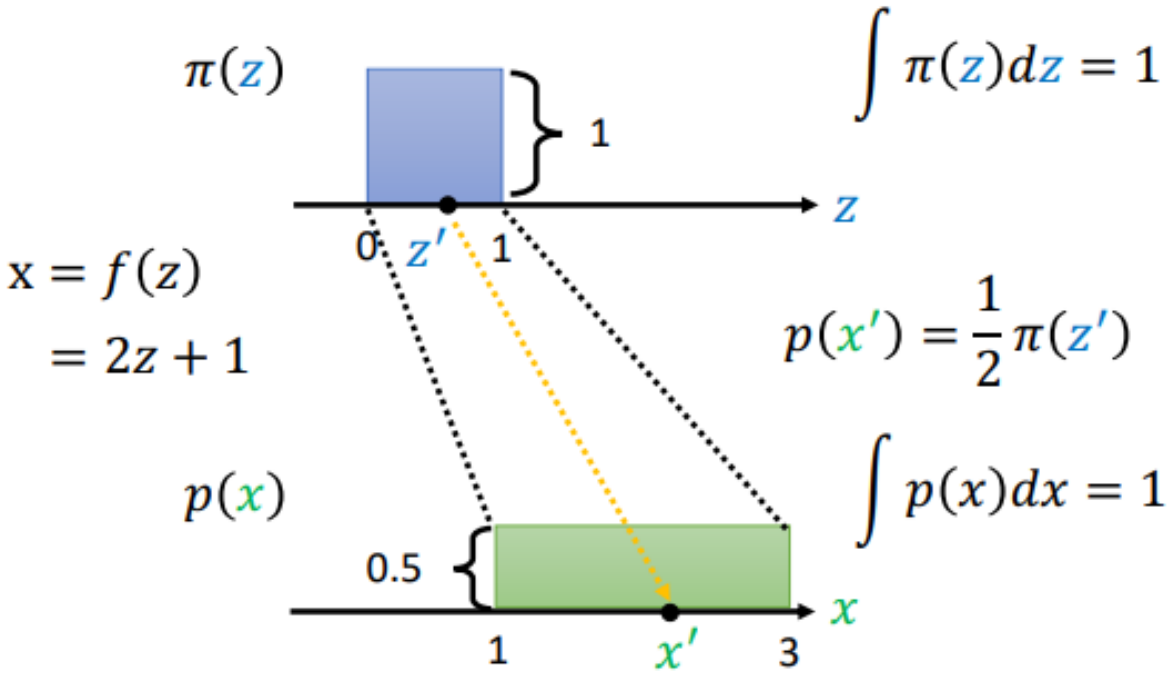


Figure 28: Transformation of a 1-D random variable to another random variable variable.

Change of Variables Theorem (cont.)

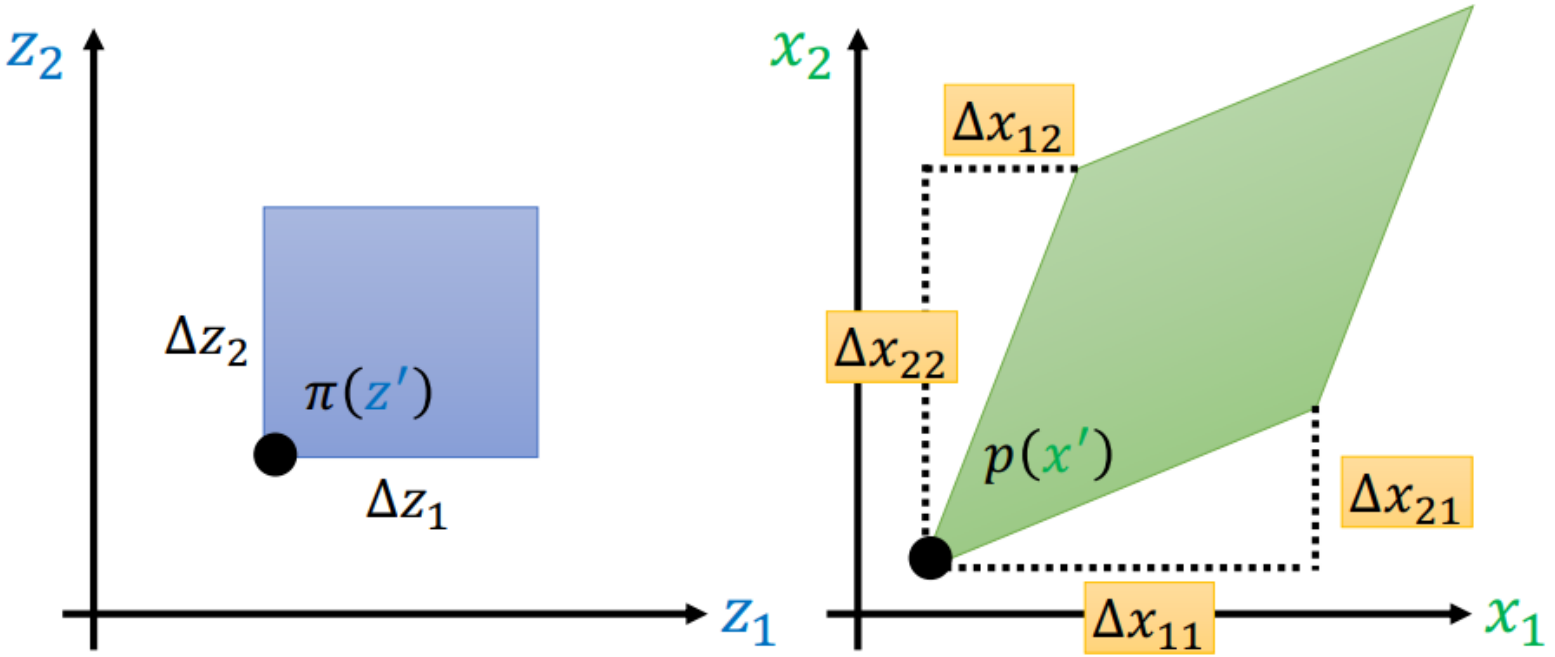


Figure 29: Transformation of a 2-D random variable to another random variable variable.

Change of Variables Theorem (cont.)

Theorem: Let Z and X be random variables which are related by a mapping $f : \mathbb{R}^n \to \mathbb{R}^n$ such that $X = f(Z)$ and $Z = f^{-1}(X)$.
Then

$$p_X(x) = p_Z(f^{-1}(x)) \left| det \left(\frac{\partial f^{-1}(x)}{\partial x} \right) \right|$$

There are several things to note here.

- x and z need to be continuous and have the same dimension.
- $\frac{\partial f^{-1}(x)}{\partial x}$ is a matrix of dimension $n \times n$, where each entry at location (i, j) is defined as $\frac{\partial f^{-1}(x_i)}{\partial x_j}$. This matrix is also known as the Jacobian matrix.
- For any invertible matrix A, $det(A^{-1}) = det(A)^{-1}$, so for $z = f^{-1}(x)$ we have

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

Jacobian (2-D Case)

1) $x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$

$x = f(z) \quad z = f^{-1}(x)$

2) $\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} z_1 + z_2 \\ 2z_1 \end{bmatrix} = f\left(\begin{bmatrix} z_1 \\ z_2 \end{bmatrix}\right)$

$\begin{bmatrix} x_2/2 \\ x_1 - x_2/2 \end{bmatrix} = f^{-1}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right)$

3) input

$J_f = \begin{bmatrix} \partial x_1/\partial z_1 & \partial x_1/\partial z_2 \\ \partial x_2/\partial z_1 & \partial x_2/\partial z_2 \end{bmatrix}$ output

$J_{f^{-1}} = \begin{bmatrix} \partial z_1/\partial x_1 & \partial z_1/\partial x_2 \\ \partial z_2/\partial x_1 & \partial z_2/\partial x_2 \end{bmatrix}$

4) $J_f = \begin{bmatrix} 1 & 1 \\ 2 & 0 \end{bmatrix}$

$J_{f^{-1}} = \begin{bmatrix} 0 & 1/2 \\ 1 & -1/2 \end{bmatrix}$

$J_f J_{f^{-1}} = I$

19

Figure 30: Example of a 2-D jacobain matrix.

Matrix Determinant

The determinant of a **square matrix** is a **scalar** that provides information about the matrix.

- 2 X 2

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$det(A) = ad - bc$$

- 3 x 3

$$A = \begin{bmatrix} a_1 & a_2 & a_3 \\ a_4 & a_5 & a_6 \\ a_7 & a_8 & a_9 \end{bmatrix}$$

$$det(A) = a_1a_5a_9 + a_2a_6a_7 + a_3a_4a_8 - a_3a_5a_7 - a_2a_4a_9 - a_1a_6a_8$$

$$det(\mathrm{A}) = 1/det(A^{-1})$$

$$det(J_f) = 1/det(J_{f^{-1}})$$

Figure 31: Example of matrix determinant.

- A generator G is a network which maps a simple distribution (for example, normal distribution) $\pi(z)$ to a complex data distribution $p_G(x)$ which aims to be as close to real data distribution $p_{data}(x)$ as possible.

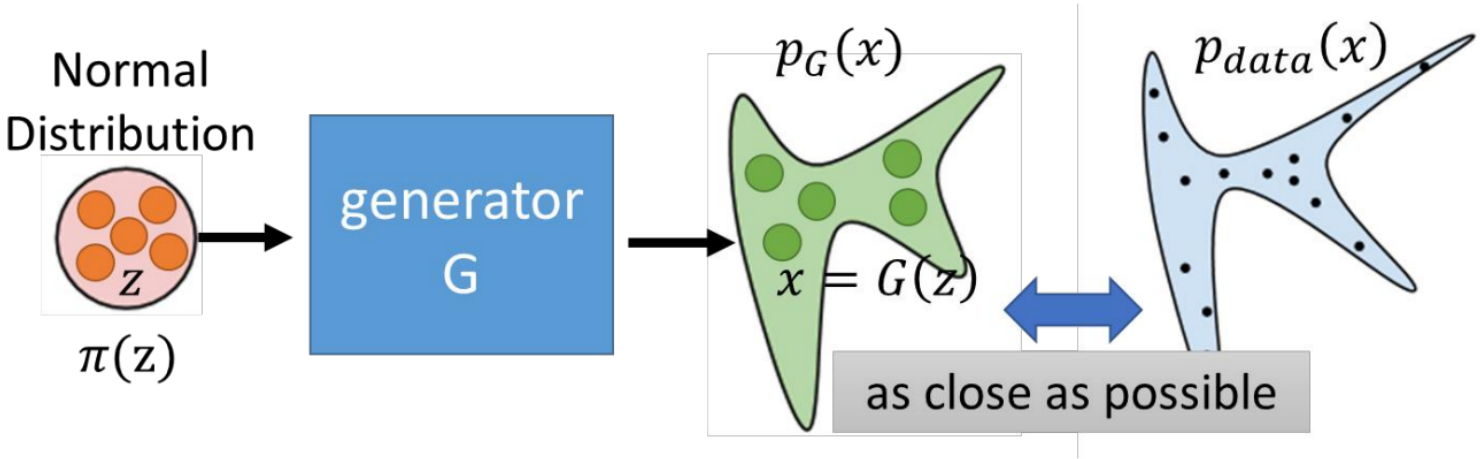
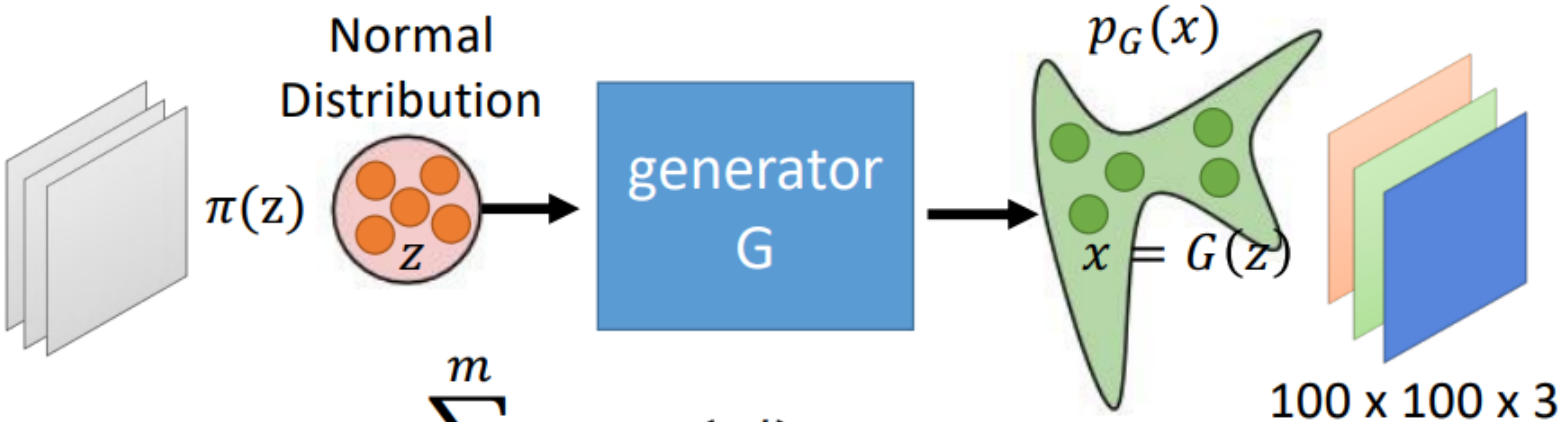


Figure 32: The goal of a generator network in a generative model

- Let us consider a directed, latent-variable model over observed variables X and latent variables Z. In a normalizing flow model, the mapping between Z and X, given by $G : \mathbb{R}^n \to \mathbb{R}^n$, is deterministic and invertible such that $X = G(Z)$ and $Z = G^{-1}(X)$.
- Using change of variables, the marginal likelihood $p(x)$ is given by

$$p_X(x;\theta) = p_Z(G_\theta^{-1}(x)) \left| det \left(\frac{\partial G_\theta^{-1}(x)}{\partial x} \right) \right|$$

Normalizing Flow Models (cont.)



$$G^* = arg \max_G \sum_{i=1}^{m} log p_G(x^i)$$

G has limitation

$$p_G(x^i) = \pi(z^i)|det(J_{G^{-1}})|$$

You can compute $det(J_G)$

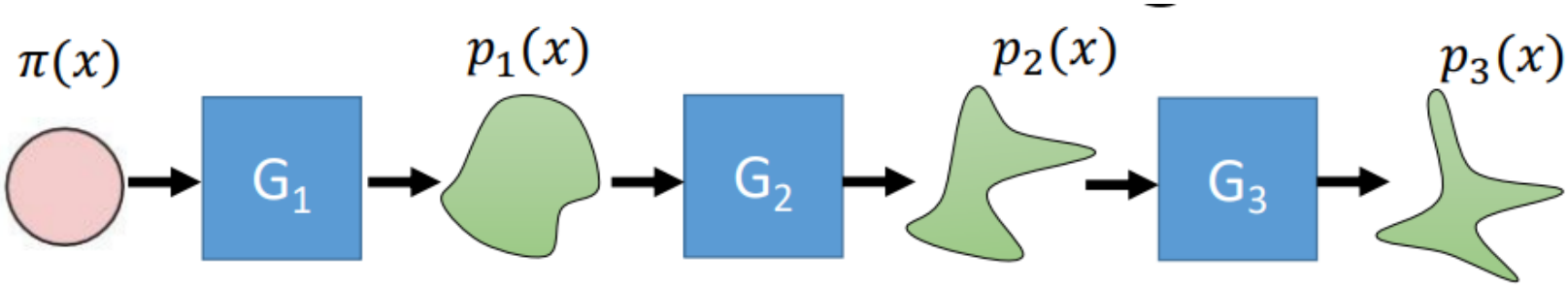
$$z^i = G^{-1}(x^i)$$

You know G^{-1}

$$log p_G(x^i) = log\pi\left(G^{-1}(x^i)\right) + log|det(J_{G^{-1}})|$$

Normalizing Flow Models (cont.)

G is limited. We need more generators



$$p_1(x^i) = \pi(z^i)\left(\left|det\left(J_{G_1^{-1}}\right)\right|\right) \qquad z^i = G_1^{-1}\left(\cdots G_K^{-1}(x^i)\right)$$

$$p_2(x^i) = \pi(z^i)\left(\left|det\left(J_{G_1^{-1}}\right)\right|\right)\left(\left|det\left(J_{G_2^{-1}}\right)\right|\right)$$

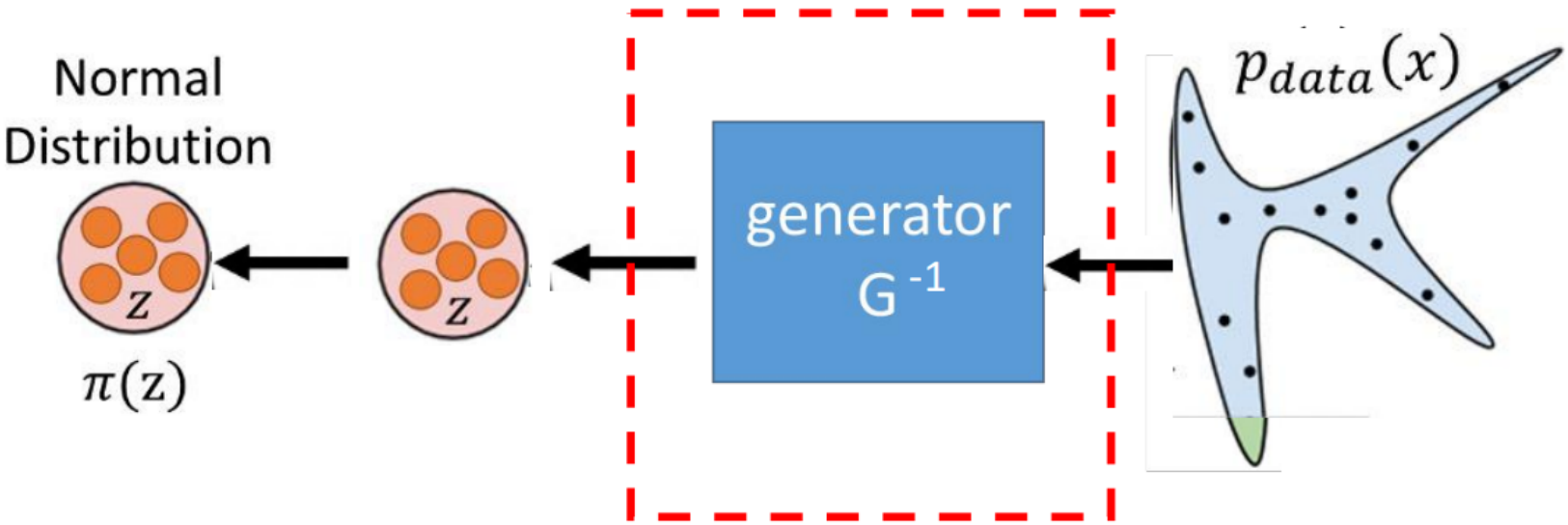
$$\vdots$$

$$p_K(x^i) = \pi(z^i)\left(\left|det\left(J_{G_1^{-1}}\right)\right|\right)\cdots\left(\left|det\left(J_{G_K^{-1}}\right)\right|\right)$$

$$logp_K(x^i) = log\pi(z^i) + \sum_{h=1}^{K} log\left|det\left(J_{G_K^{-1}}\right)\right| \quad \text{Maximise}$$

Normalizing Flow Models (cont.)

Actually, we train G^{-1} , but we use G for generation.



Normalizing Flow Models (cont.)

- **Normalising** means that the change of variables gives a normalised density after applying an invertible transformation.
- **Flow** means that the invertible transformations can be composed with each other to create more complex invertible transformations.

Learning and Inference

- Learning via maximum likelihood over the dataset D

$$\max_{\theta} log\ p(D;\theta) = \sum_{x \in D} + log\ \pi(G_{\theta}^{-1}(x)) log \left| det \left(\frac{\partial G_{\theta}^{-1}(x)}{\partial x} \right) \right|$$

- Exact likelihood evaluation via inverse transformation and change of variables formula
- Sampling via forward transformation $G_{\theta} : Z \rightarrow X$

$$z \sim \pi(z), x = G_{\theta}(z)$$

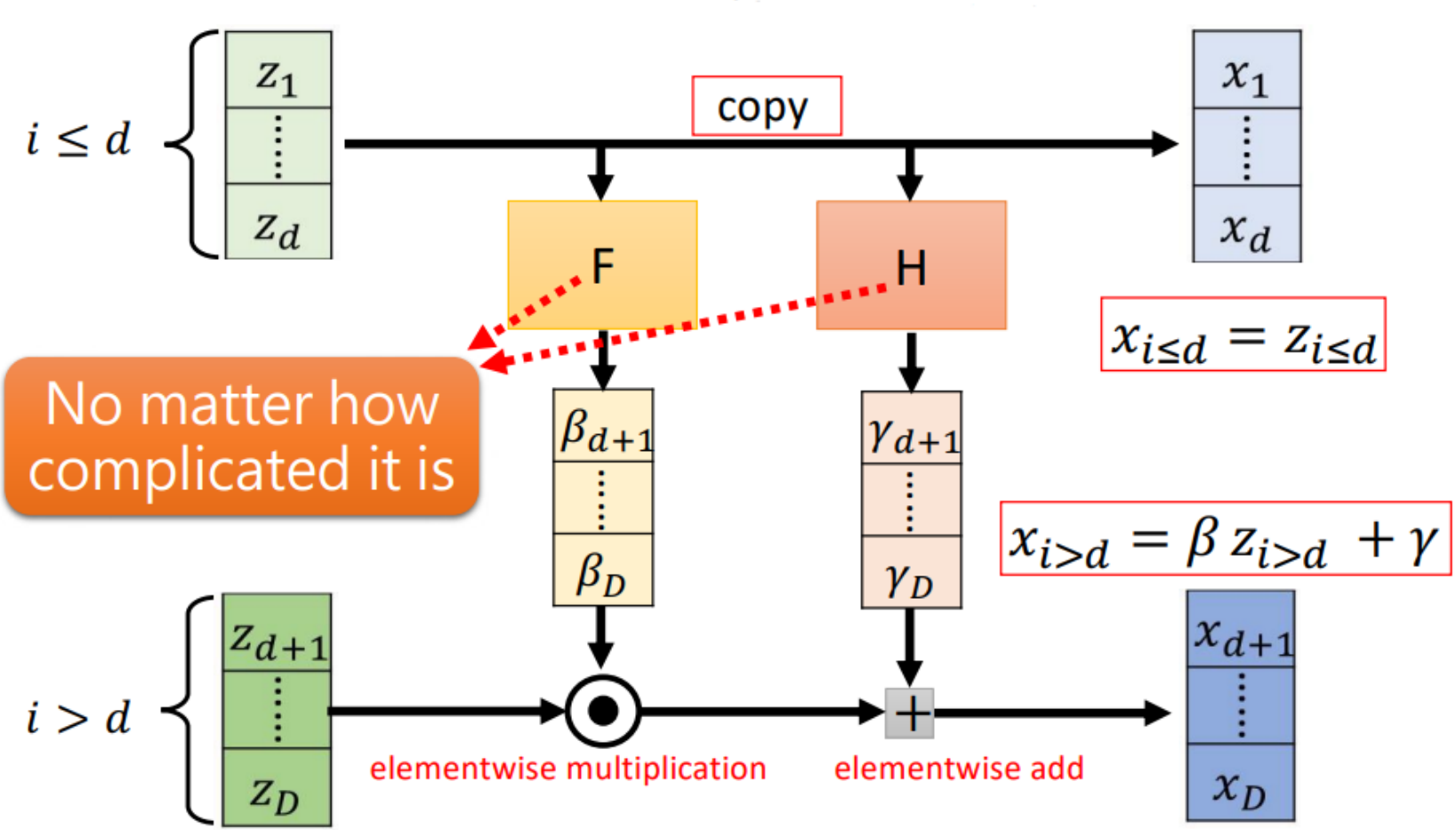
- Latent representations inferred via inverse transformation (no inference network required!)

$$z = G_{\theta}^{-1}(x)$$

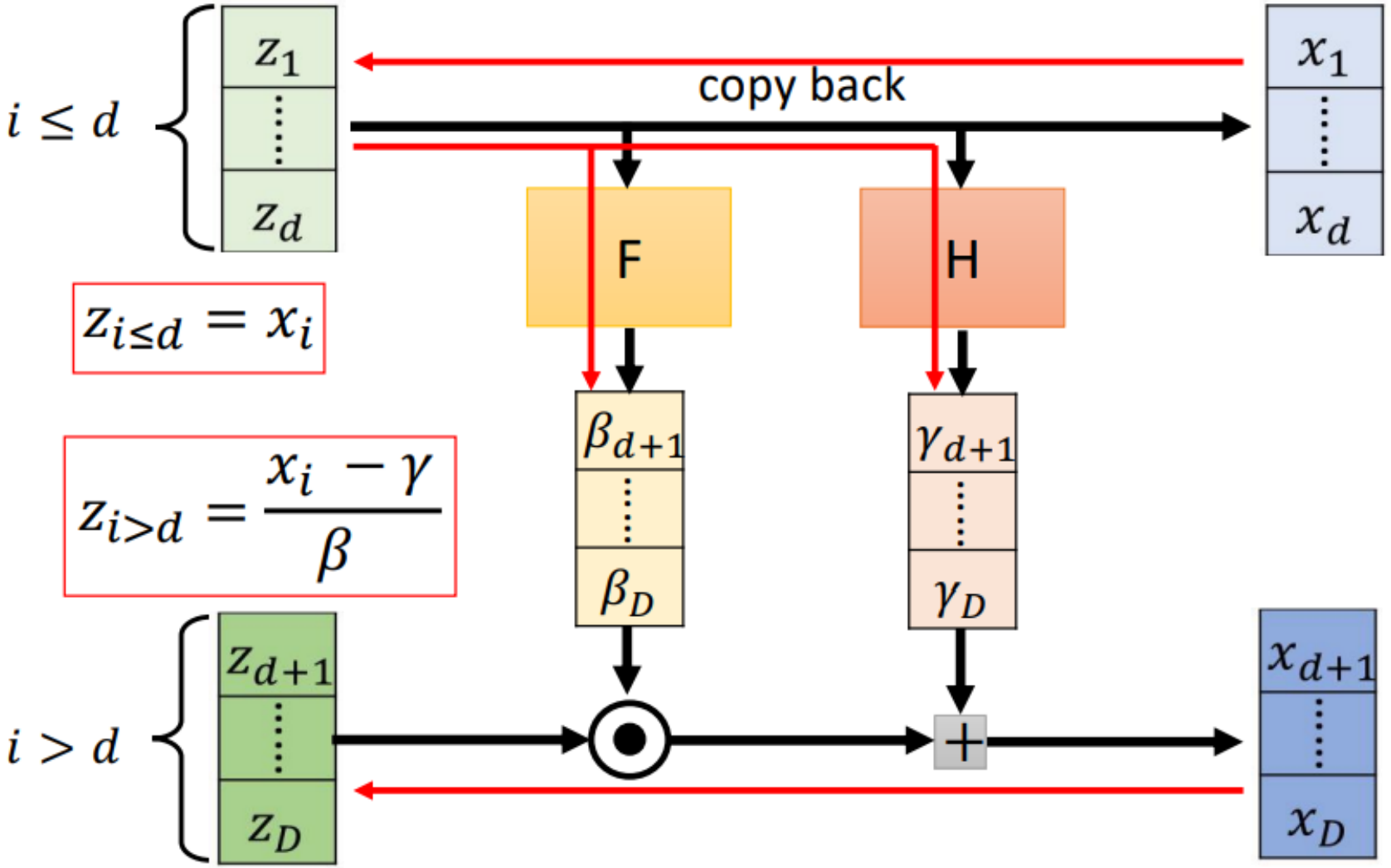
Requirements for flow models

- Simple prior $\pi(z)$ that allows for efficient sampling and tractable likelihood evaluation. E.g., Gaussian
- Invertible transformations
- Computing likelihoods also requires the evaluation of determinants of $n \times n$ Jacobian matrices, where n is the data dimensionality
  - Computing the determinant for an $n \times n$ matrix is $O(n^3)$ which is prohibitively expensive within a learning loop!
  - Key idea: Choose transformations so that the resulting Jacobian matrix has special structure. For example, the determinant of a triangular matrix is the product of the diagonal entries, i.e., an $O(n)$ operation
- We a fast and simple prior (Gaussian)
- Now, the big question. How do we get invertible transformations and make determinants of jacobian efficient to compute
- **One solution**: We can use coupling layer to design invertible function and calculate the determinant of Jacobian efficiently!

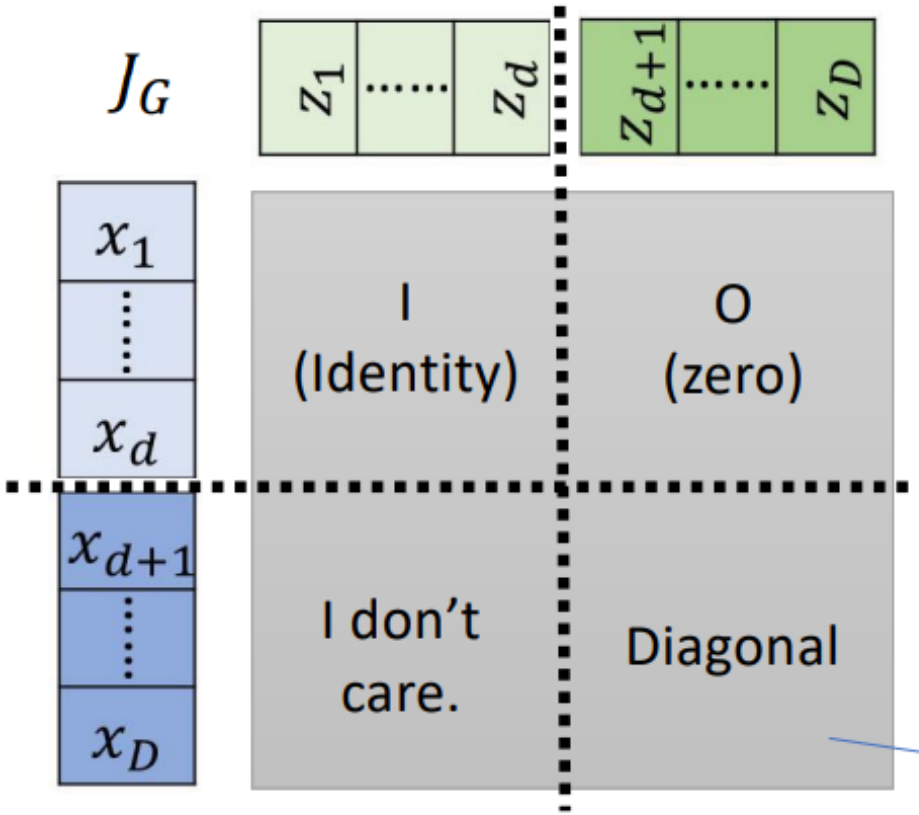
Coupling layers



Coupling layers (cont.)



Coupling layers (cont.)



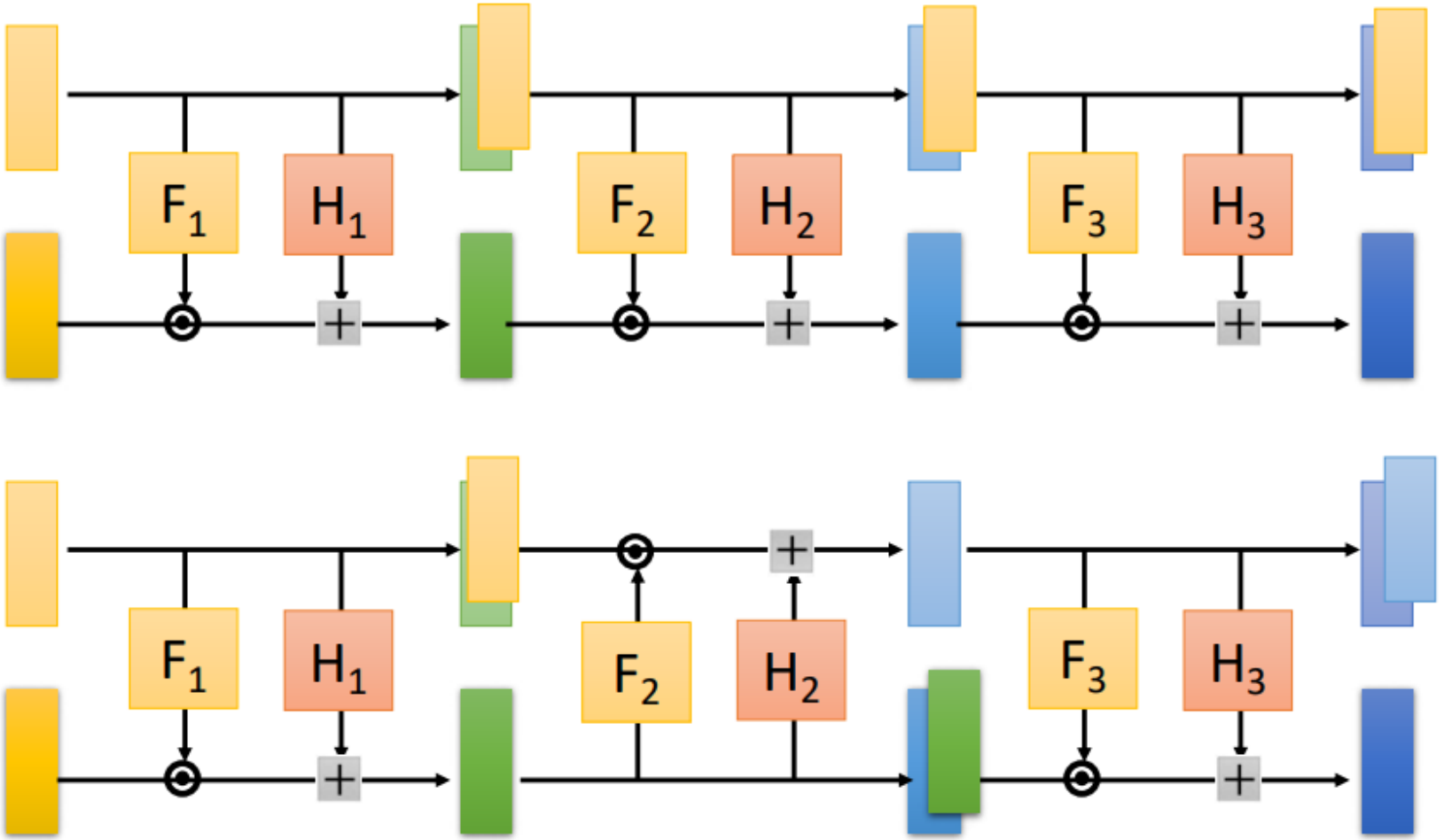
$$det(J_G)$$

$$= \frac{\partial x_{d+1}}{\partial z_{d+1}} \frac{\partial x_{d+2}}{\partial z_{d+2}} \cdots \frac{\partial x_D}{\partial z_D}$$

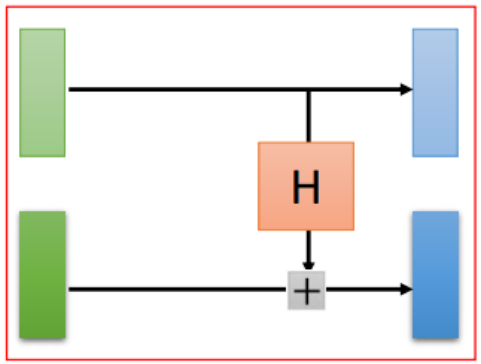
$$= \beta_{d+1}\beta_{d+2} \cdots \beta_D$$

$$x_{i>d} = \beta z_{i>d} + \gamma$$

Coupling layers (cont.)



- Additive coupling layers
  - Partition the variables z into two disjoint subsets
  - $x_{1:d} = z_{1:d}$
  - $x_{d+1:n} = z_{d+1:n} + H(z_{1:d})$



- Additive coupling layers are composed together (with arbitrary partitions of variables in each layer)
- Final layer of NICE applies a rescaling transformation

NICE - Results

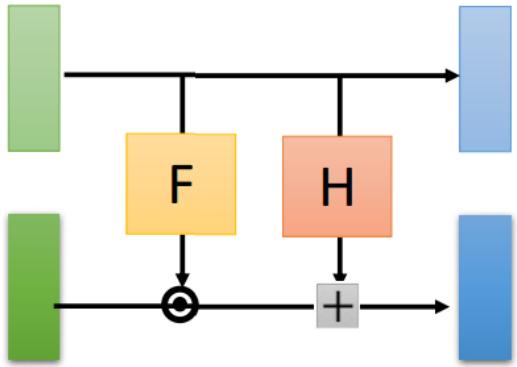
Figure 33: NICE generated samples when trained on MNIST digits dataset

NICE - Results (cont.)

Figure 34: NICE generated samples when trained on CIFAR-10 dataset

Real NVP

- Coupling layers
  - Partition the variables z into two disjoint subsets
  - $x_{1:d} = z_{1:d}$
  - $x_{d+1:n} = z_{d+1:n} \odot F(z_{1:d}) + H(z_{1:d})$

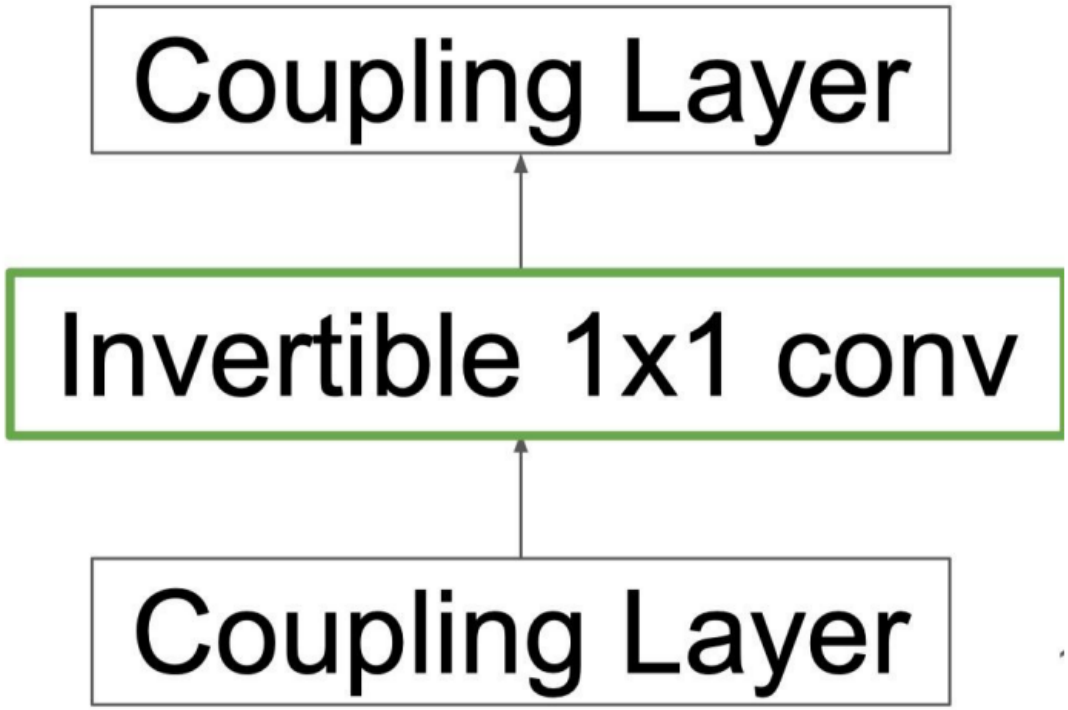


- Coupling layers are composed together (with arbitrary partitions of variables in each layer)

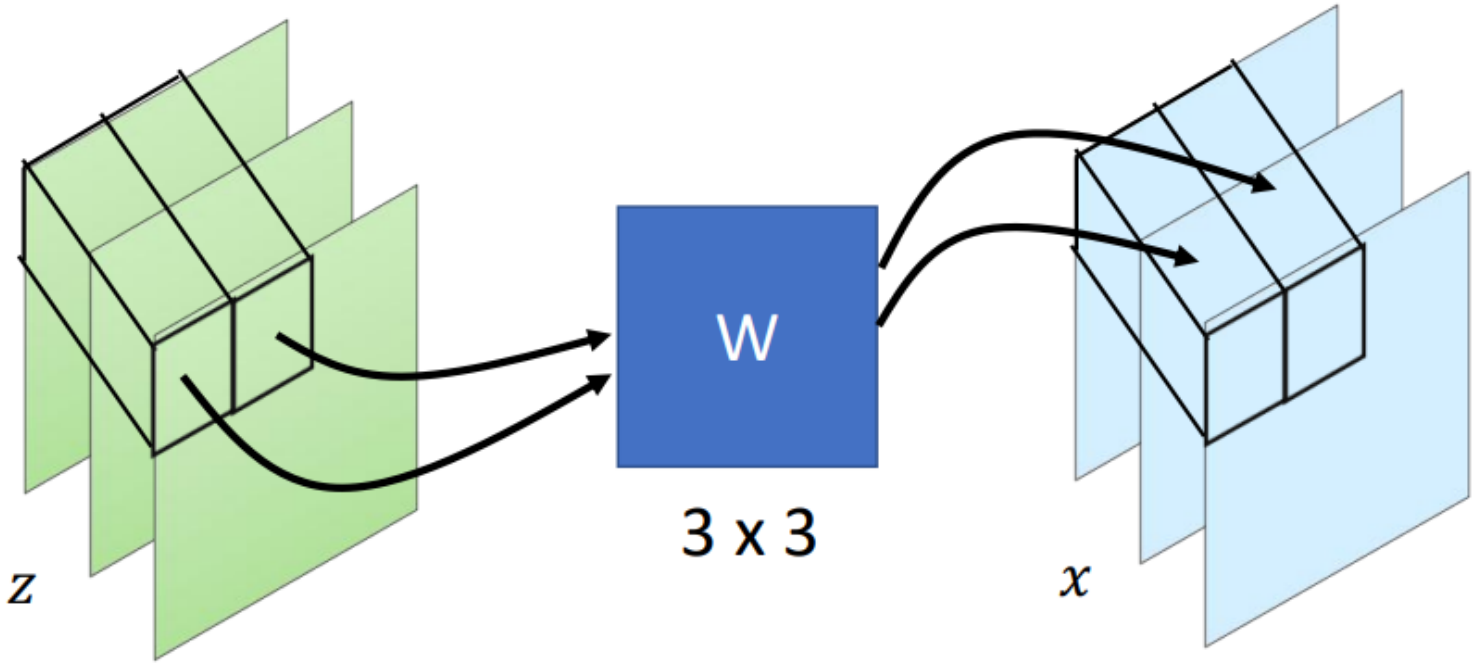
Real NVP - Results

Figure 35: Real NVP generated samples

Add an invertible 1x1 convolution between coupling layers



GLOW: Generative Flow with Invertible 11 Convolutions (cont.)



W can shuffle the channels.

If W is invertible, it is easy to compute W^{-1}.

$$\begin{bmatrix} 3 \\ 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

Glow - Results

Figure 36: GLOW generated samples

Glow - Results (cont.)

Figure 37: Linear interpolation in latent space between real images with GLOW

Normalizing Flow Models - Summary

- Transform simple distributions into more complex distributions via change of variables
- Set up invertible transformations between latent space and data
- Jacobian of transformations should have tractable determinant for efficient learning and density estimation

References

Reference Slides

- Fei-Fei Li "Generative Deep Learning" CS231
- Hao Dong "Deep Generative Models"
- Hung-Yi Lee "Machine Learning"
- Murtaza Taj "Deep Learning" CS437