# (Deep) Generative Models

**Phillip Rust, May 16, 2022**

# Previous Lecture — Transformers

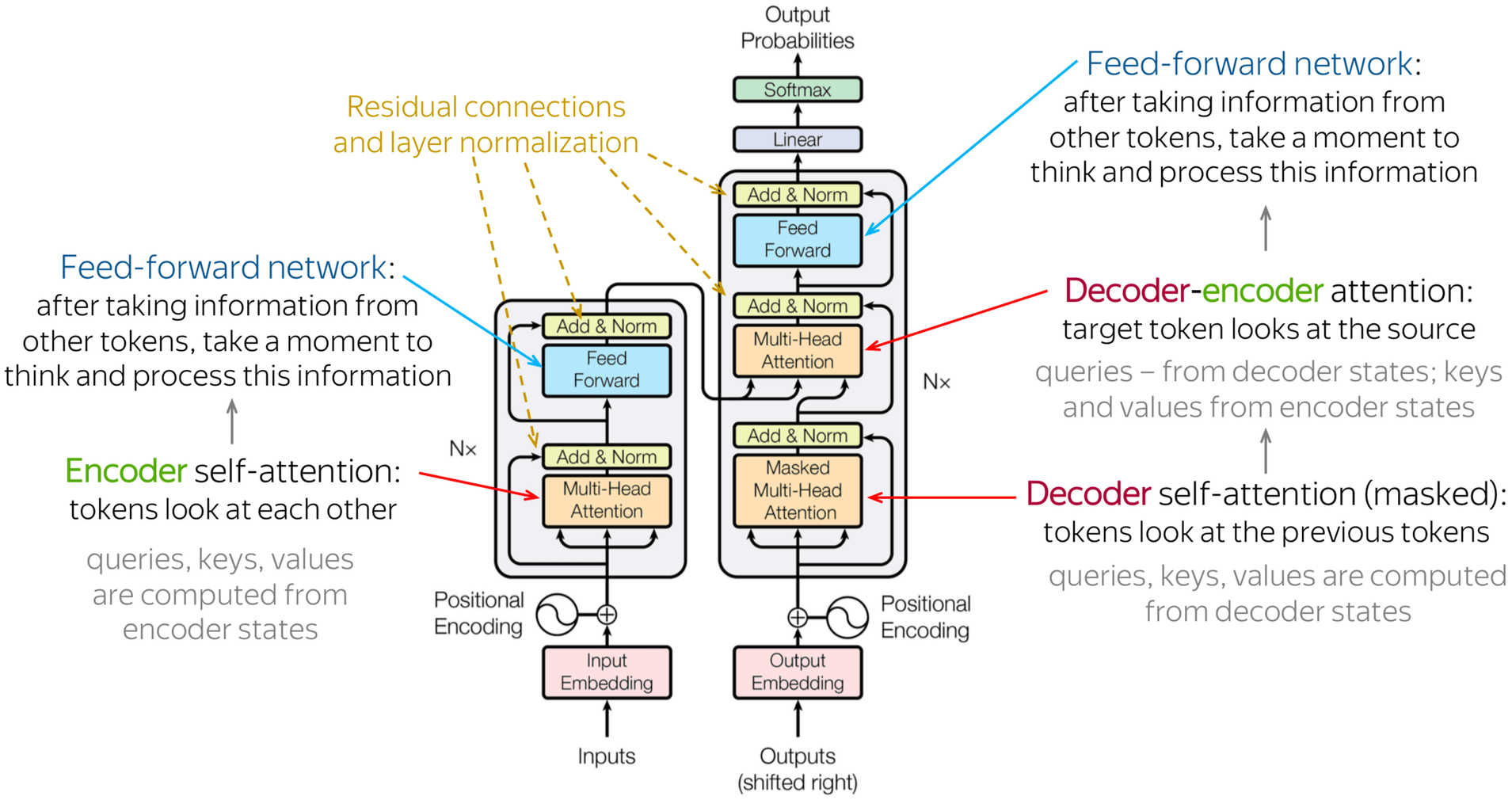



Figure taken from https://lena-voita.github.io/nlp_course/seq2seq_and_attention.html

# Today's Lecture

1. Supervised vs Unsupervised Learning
2. Generative Modeling
3. Applications of generative models
4. Fully-visible belief networks (FVBNs) & Autoregressive Decoders
5. (Variational) Autoencoders
6. Generative Adversarial Networks (GANs)

# Today's Lecture

1. **Supervised vs Unsupervised Learning**
2. Generative Modeling
3. Applications of generative models
4. Fully-visible belief networks (FVBNs) & Autoregressive Decoders
5. (Variational) Autoencoders
6. Generative Adversarial Networks (GANs)

# Supervised vs Unsupervised Learning

## Supervised Learning

- Dataset: (x, y)

  x = input (features), y = label

- **Goal**: Learn a *mapping function* f: x → y

→ **Cat**

- **Examples:** Classification, Regression, Object Detection, Image Captioning, …

***"Horrible service! I ordered a cat but they delivered raw chicken instead. 0/10"***

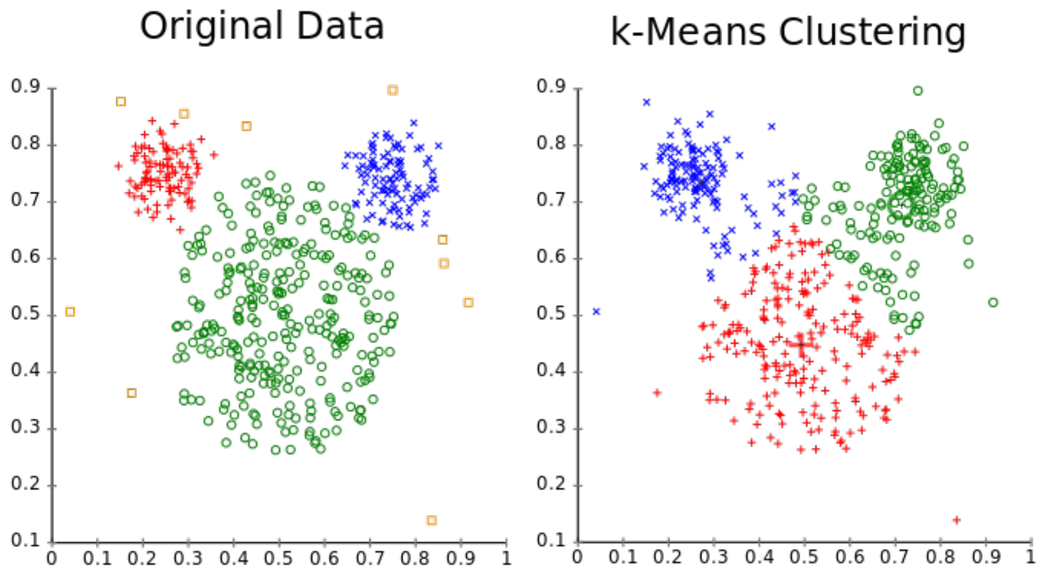
→ **Negative**

# Supervised vs Unsupervised Learning

## Unsupervised Learning

- Dataset: x (only inputs, no labels)
- **Goal**: Learn some underlying hidden structure of the data
- **Examples**: Clustering, dimensionality reduction, density estimation, …

Figure from https://upload.wikimedia.org/wikipedia/commons/thumb/0/09/ClusterAnalysis_Mouse.svg/1024px-ClusterAnalysis_Mouse.svg.png



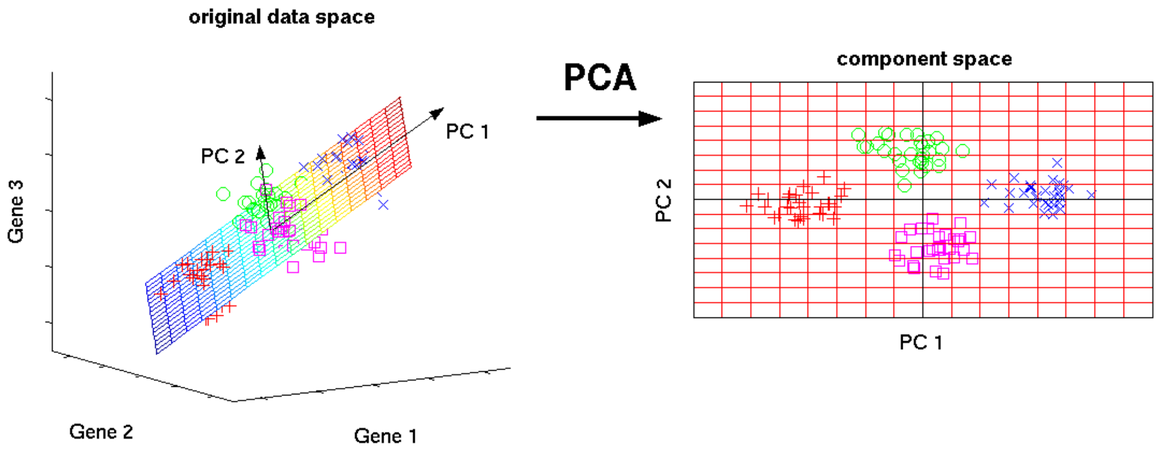

**Clustering**

# Supervised vs Unsupervised Learning

## Unsupervised Learning

- Dataset: x (only inputs, no labels)
- **Goal**: Learn some underlying hidden structure of the data
- **Examples**: Clustering, dimensionality reduction, density estimation, …

Figure from http://phdthesis-bioinformatics-maxplanckinstitute-molecularplantphys.matthias-scholz.de/fig_pca_illu3d.png




**Dimensionality Reduction (3D → 2D)**

# Supervised vs Unsupervised Learning

## Unsupervised Learning

- Dataset: x (only inputs, no labels)
- **Goal**: Learn some underlying hidden structure of the data
- **Examples**: Clustering, dimensionality reduction, density estimation, …

1D

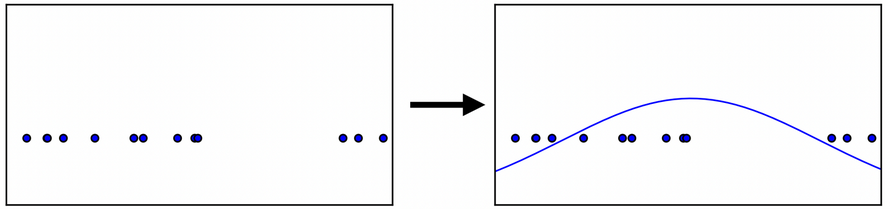



Figure from https://www.iangoodfellow.com/slides/2018-06-22-gan_tutorial.pdf

2D

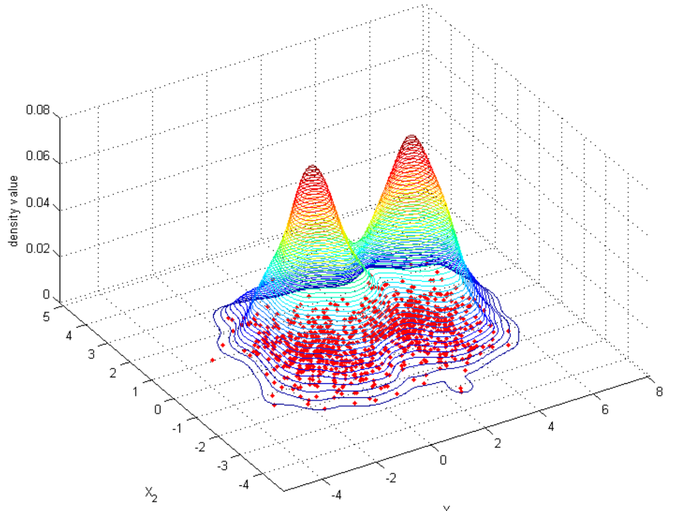



Figure from https://upload.wikimedia.org/wikipedia/commons/1/1b/Bivariate_example.png

**Density Estimation**

# Supervised vs Unsupervised Learning

## Supervised Learning

- Dataset: (x, y)

  x = input (features), y = label

- **Goal**: Learn a *mapping function* f: x →

- **Examples:** Classification, Regression, Object Detection, Image Captioning, …

## Unsupervised Learning

- Dataset: x (only inputs, no labels)

- **Goal**: Learn some underlying hidden structure of the data

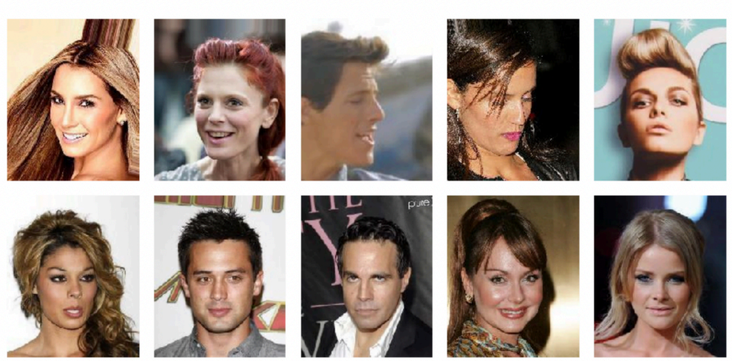
- **Examples**: Clustering, dimensionality reduction, density estimation, …

# Today's Lecture

1. Supervised vs Unsupervised Learning
2. **Generative Modeling**
3. Applications of generative models
4. Fully-visible belief networks (FVBNs) & Autoregressive Decoders
5. (Variational) Autoencoders
6. Generative Adversarial Networks (GANs)

# Generative Modeling

## Learning the data-generating distribution through density estimation



Training Data
(CelebA)

Sample Generator
(Karras et al, 2017)

Figure taken from https://www.iangoodfellow.com/slides/2018-06-22-gan_tutorial.pdf

**Premise:** There is a distribution p(x) that underlies our training data x

**Plan:** We want to learn the distribution p(x) as closely as possible and sample from it to generate new data

**Method: Density estimation**

- **Explicit :** Explicitly define $\mathbf{p_{model}(x)}$ and solve for it
- **Implicit :** Learn to sample from $\mathbf{p_{model}(x)}$ without having to define it

# Today's Lecture

1. Supervised vs Unsupervised Learning
2. Generative Modeling
3. **Applications of generative models**
4. Fully-visible belief networks (FVBNs) & Autoregressive Decoders
5. (Variational) Autoencoders
6. Generative Adversarial Networks (GANs)

# Applications of generative models

## Creative purposes

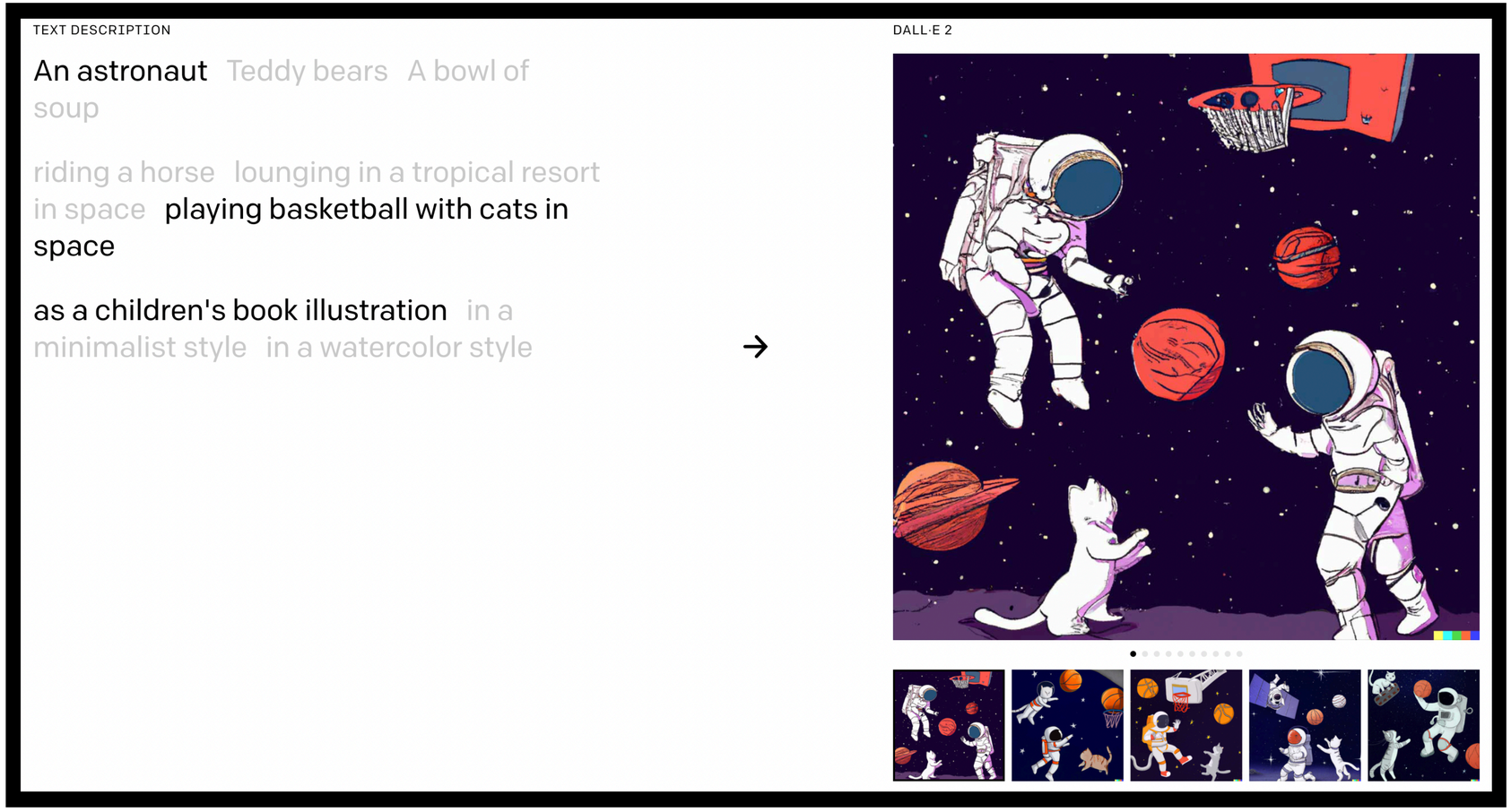



Figure from https://openai.com/dall-e-2/

Copyright Sofia Crespo
Figure from https://blogs.nvidia.com/blog/2022/05/12/ai-art-times-square-endangered-species/

# Applications of generative models

## Creative purposes





Figure from https://blogs.nvidia.com/blog/2022/05/12/ai-art-times-square-endangered-species/

# Applications of generative models

## Compression & Super-resolution

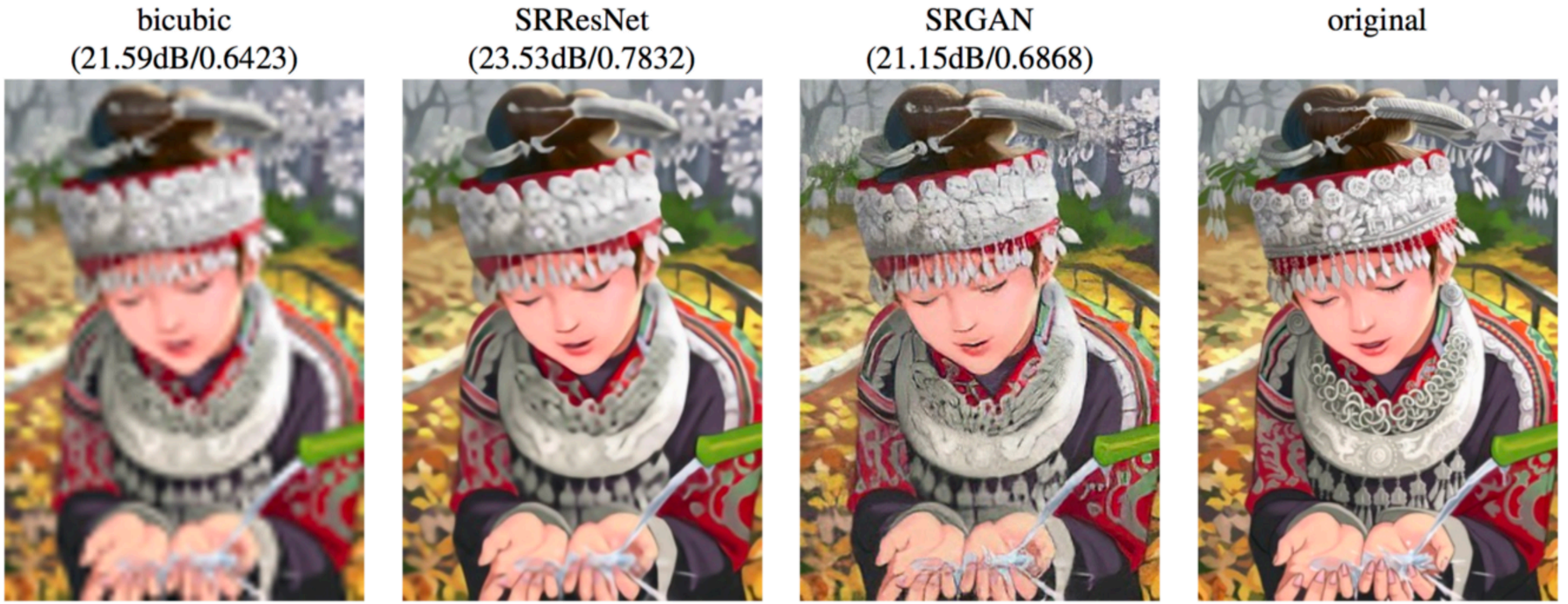



Figure 2: From left to right: bicubic interpolation, deep residual network optimized for MSE, deep residual generative adversarial network optimized for a loss more sensitive to human perception, original HR image. Corresponding PSNR and SSIM are shown in brackets. [4× upscaling]

Ledig et al. (2017), Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network

# Applications of generative models

## Drug Design & Other medical applications

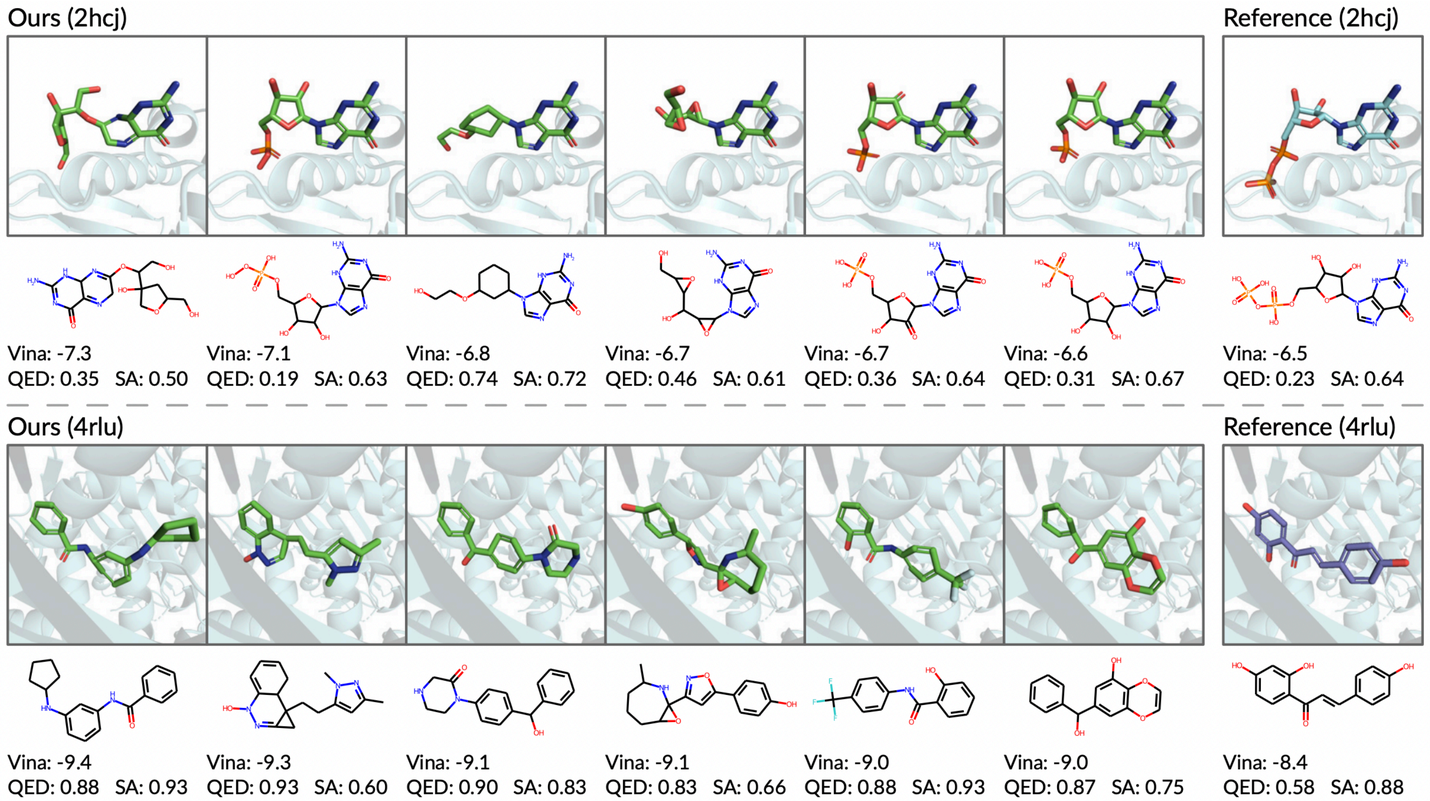



Figure 4: Generated molecules with top binding affinity and the reference molecule for two representative binding sites. Lower Vina score indicates higher binding affinity.

Luo et al. (2021), A 3D Generative Model for Structure-Based Drug Design

# Applications of generative models

- Learning general purpose features for downstream tasks (e.g. classification)
- Gaining insights from high-dimensional data (physics, medical imaging, etc.)
- Modeling the physical world for simulation and planning (robotics, reinforcement learning)
- …

# Generative models — Taxonomy

**Maximum likelihood estimation**: Find the set of parameters **θ*** that maximizes the (log) likelihood of our data **x**

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{x \sim p_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{\theta})$$

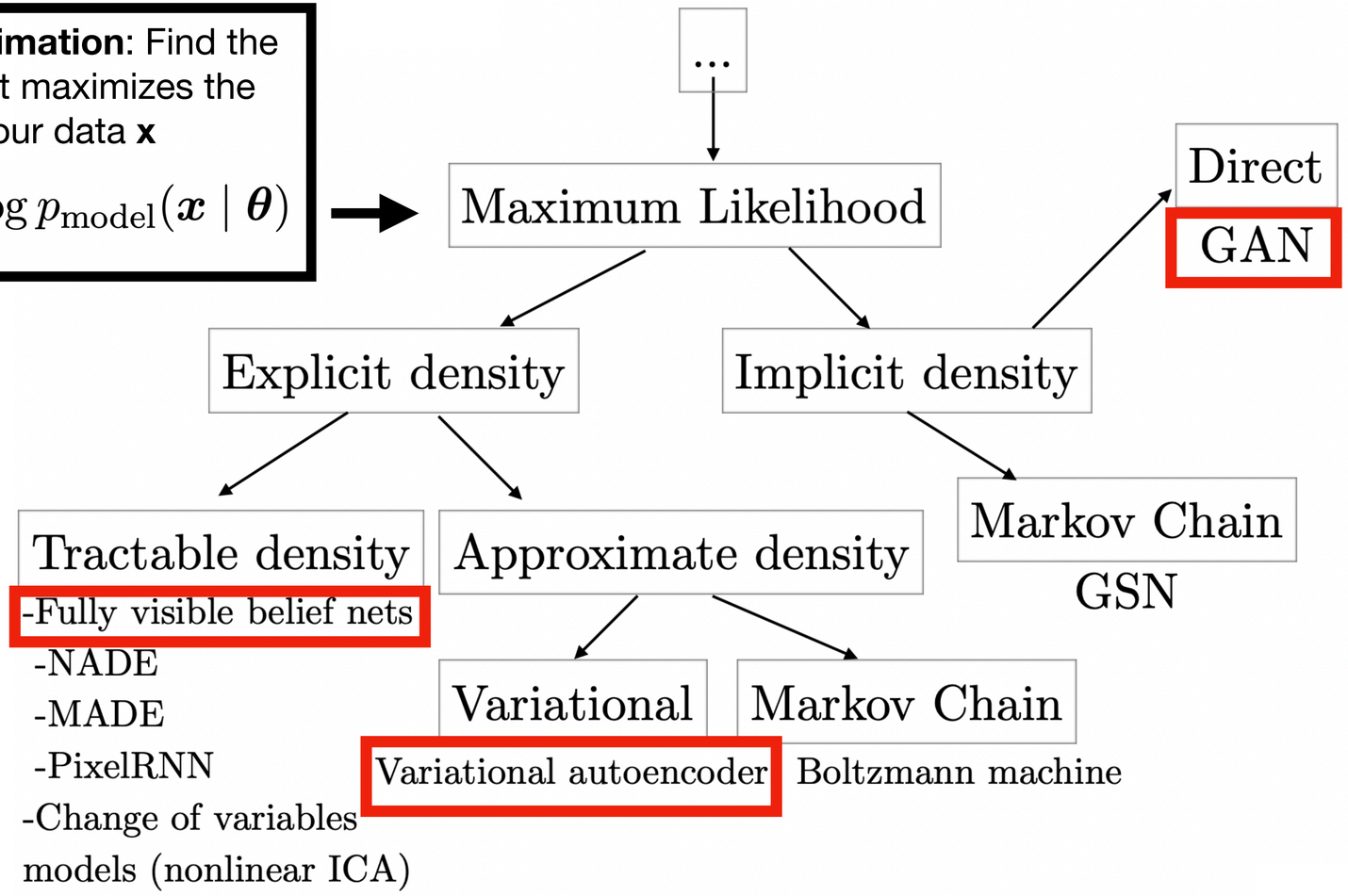



Figure taken from https://www.iangoodfellow.com/slides/2016-12-04-NIPS.pdf

# Today's Lecture

1. Supervised vs Unsupervised Learning
2. Generative Modeling
3. Applications of generative models
4. **Fully-visible belief networks (FVBNs) & Autoregressive Decoders**
5. (Variational) Autoencoders
6. Generative Adversarial Networks (GANs)

# Fully-visible Belief Networks (FVBNs)

## Autoregressive Decoders

- Explicit density model
- Decompose joint likelihood into a product of tractable conditional likelihoods using the chain rule

$$p(x) = p(x_1, x_2, \ldots, x_n)$$

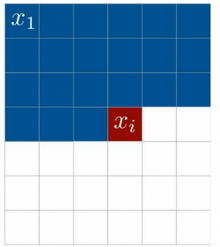
Likelihood of image x

Joint likelihood of each pixel in the image

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

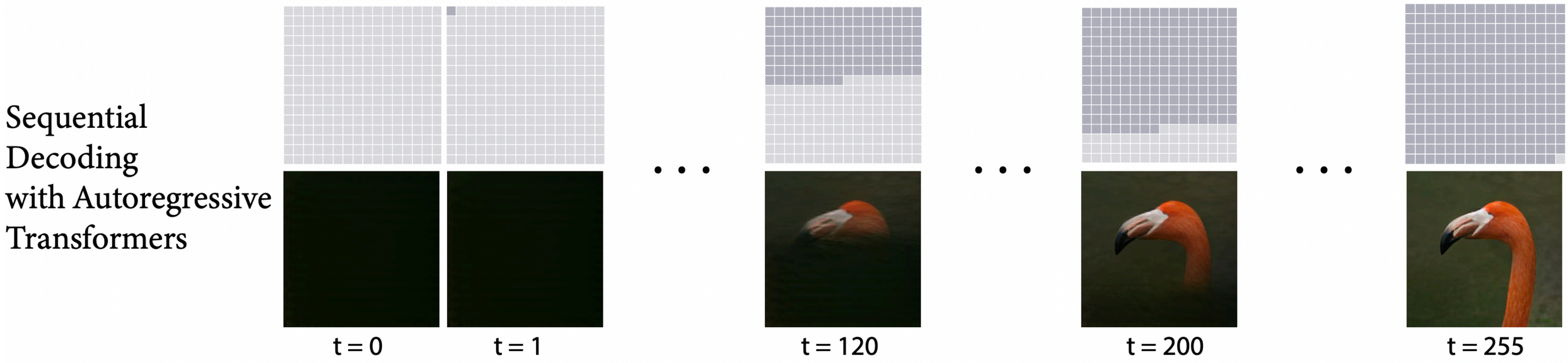
Likelihood of image x

Probability of i'th pixel value given all previous pixels



# Fully-visible Belief Networks (FVBNs)

## Autoregressive Decoders




Chang et al. (2022), MaskGIT: Masked Generative Image Transformer

# Fully-visible Belief Networks (FVBNs)

## Autoregressive Decoders

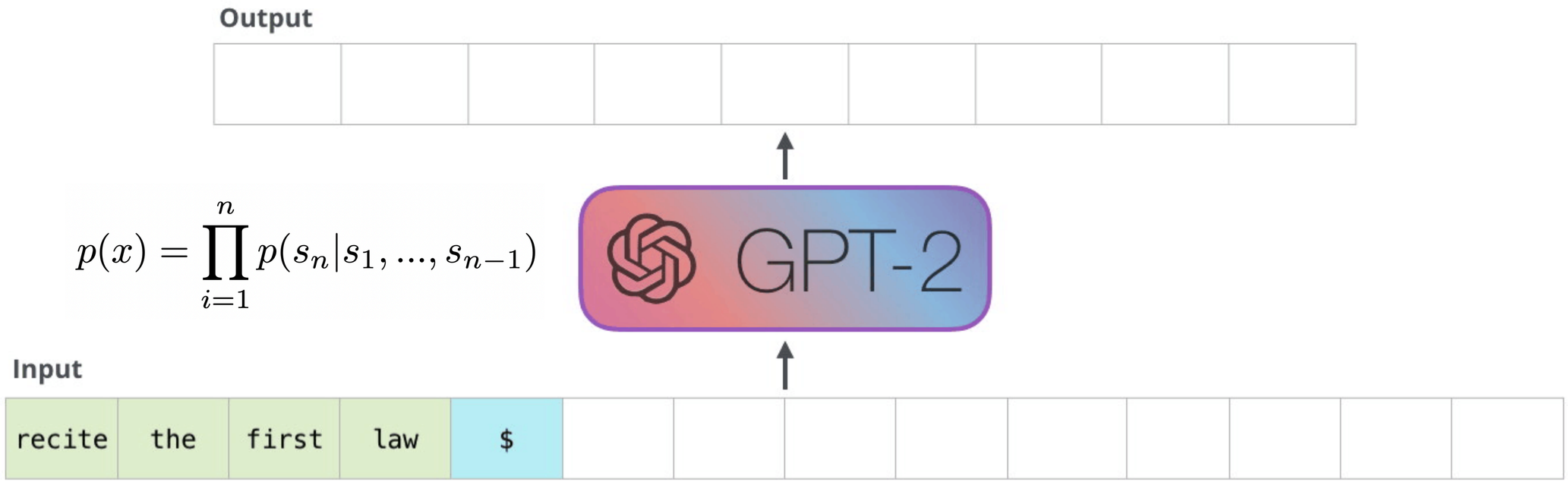



Figure taken from https://jalammar.github.io/illustrated-gpt2/

# Pros & Cons of Autoregressive Decoders

- Pros:
  - p(x) can be computed explicitly and exactly
  - Easy optimization via single objective function
  - High output quality
- Cons:
  - Slow O(n) decoding, n = output sequence length
  - Not controlled by latent variable (z)

# Today's Lecture

1. Supervised vs Unsupervised Learning
2. Generative Modeling
3. Applications of generative models
4. Fully-visible belief networks (FVBNs) & Autoregressive Decoders
5. **(Variational) Autoencoders**
6. Generative Adversarial Networks (GANs)

# Variational Autoencoders (VAEs)

- Autoregressive decoders define a tractable density function, directly optimised through MLE in the form of gradient descent

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

- VAEs define an intractable density function with latent variable z

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

**Yay, no chain dependencies, so we can sample the whole thing at once :))**

**Oh no, how are we supposed to optimize this though :((**

**And what is a latent variable z??**

# Autoencoders

- Unsupervised model
- Learns lower-dimensional feature representation **z** from **unlabelled** training data **x**
- Bottleneck architecture, i.e., dim(z) < dim(x)

**Why?**

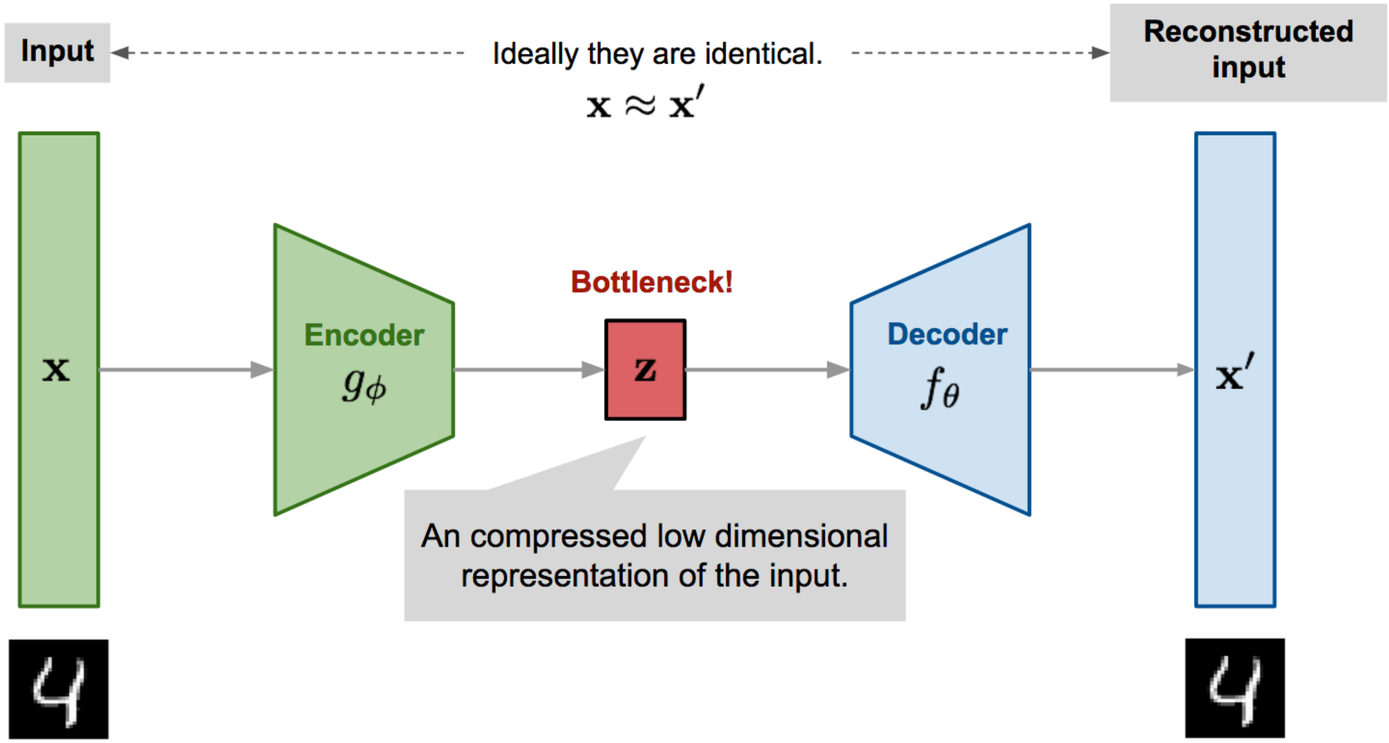



Figure taken from https://lilianweng.github.io/posts/2018-08-12-vae/

# Autoencoders

- Unsupervised model
- Learns lower-dimensional feature representation **z** from **unlabelled** training data **x**
- Bottleneck architecture, i.e., dim(z) < dim(x)

**Why?**

To learn meaningful factors of variation rather than just memorizing

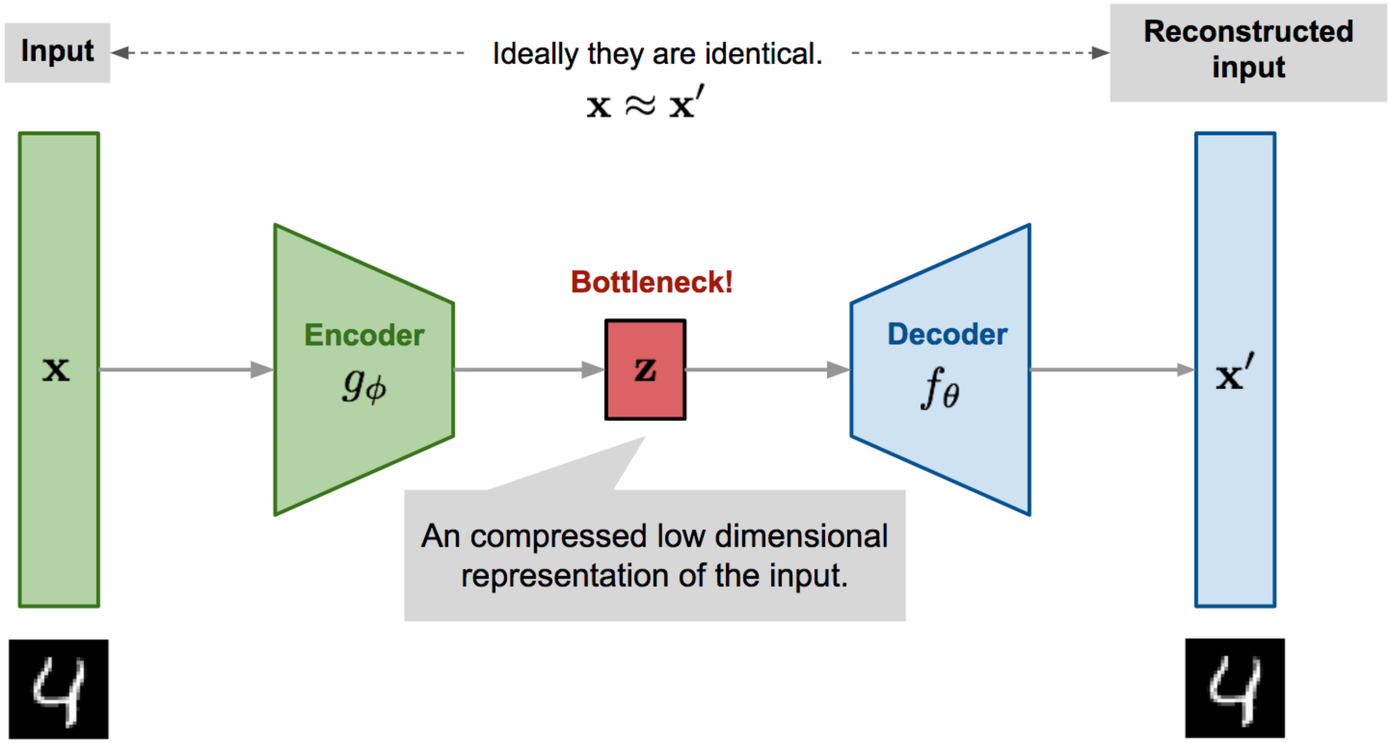



Figure taken from https://lilianweng.github.io/posts/2018-08-12-vae/

# Autoencoders

- Unsupervised model
- Learns lower-dimensional feature representation **z** from **unlabelled** training data **x**
- Bottleneck architecture, i.e., dim(z) < dim(x)

**Why?**

To learn meaningful factors of variation rather than just memorizing

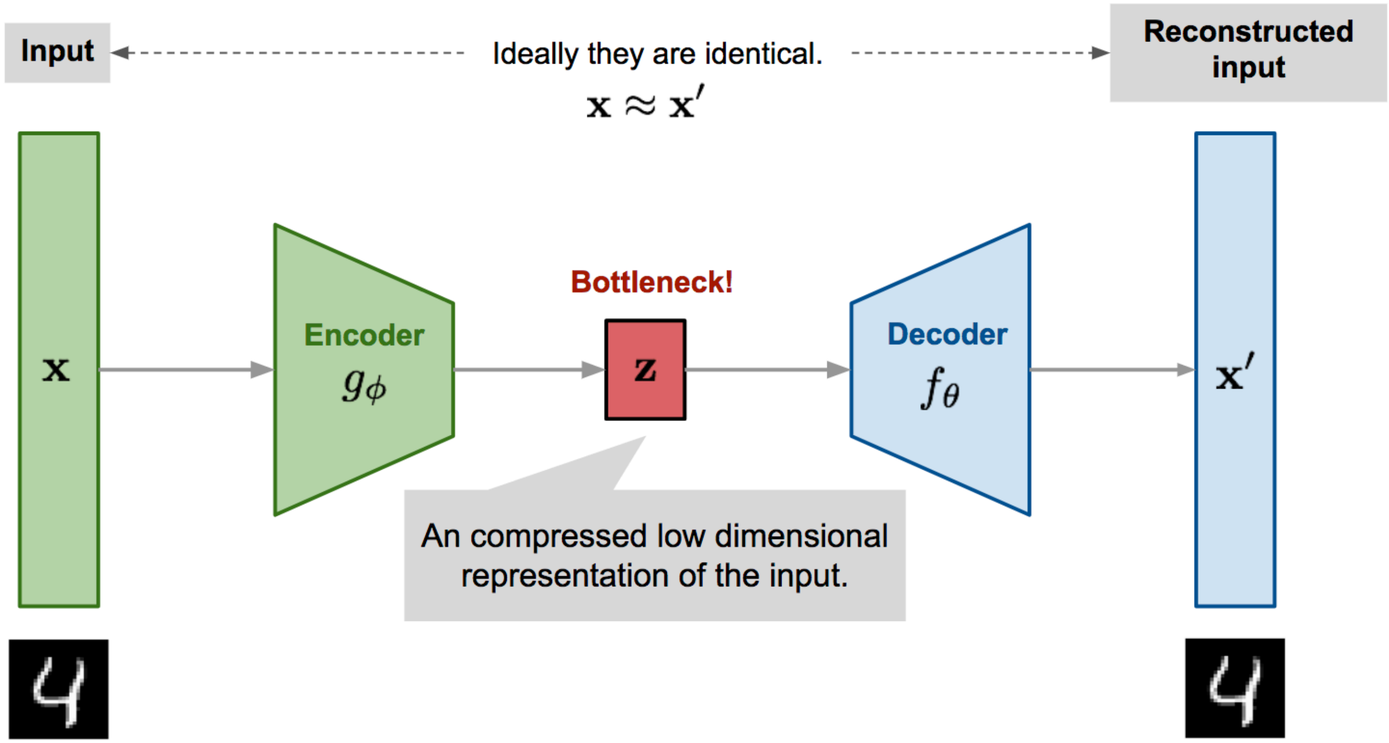



Figure taken from https://lilianweng.github.io/posts/2018-08-12-vae/

$$L_{\text{AE}}(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}^{(i)} - f_\theta(g_\phi(\mathbf{x}^{(i)})))^2$$

# Autoencoders

- Unsupervised model
- Learns lower-dimensional feature representation **z** from **unlabelled** training data **x**
- Bottleneck architecture, i.e., dim(z) < dim(x)

**Why?**

To learn meaningful factors of variation rather than just memorizing

**Transfer learning:** We can replace the decoder with a classifier and train the full network jointly on a downstream task with supervision using the learned latent features **z**

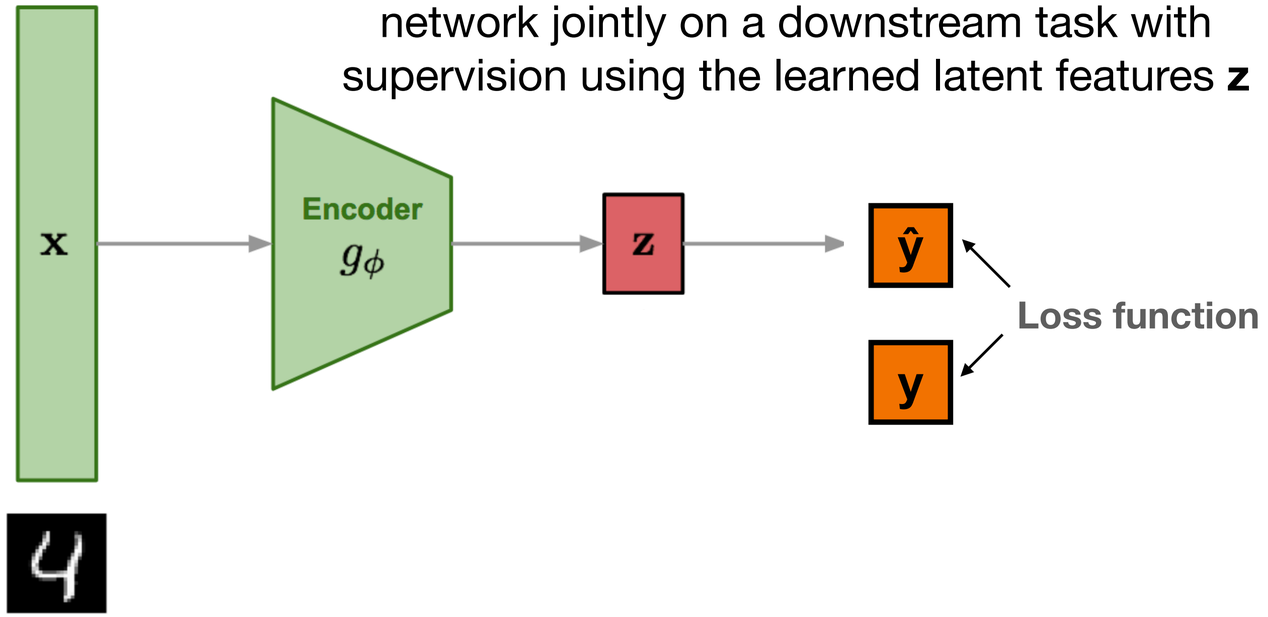



Figure modified from https://lilianweng.github.io/posts/2018-08-12-vae/

*Slide modified from Lilian Weng's Blog*

# Variational Autoencoders

- We cannot sample from **z** in the vanilla Autoencoder
- To achieve this, let's map our inputs **x** to a parametrized distribution $\mathbf{p_\theta(z)}$ rather than a *fixed* vector **z**
- Relationship between **x** and **z** defined in Bayesian terms:

Prior $p_\theta(\mathbf{z})$

Likelihood $p_\theta(\mathbf{x}|\mathbf{z})$

Posterior $p_\theta(\mathbf{z}|\mathbf{x})$

*Slide modified from Lilian Weng's Blog*

# Variational Autoencoders

Assuming we have found a good set of parameters **θ**, we can **generate an example x(i)** as follows:

1. Sample **z(i)** from **pθ(z)**
2. Decode **x(i)** via conditional probability distribution

   **pθ(x | z = z(i))**

The optimal parameters **θ*** maximize the probability of generating a real example **x(i)**:

$$\theta^* = \arg\max_\theta \prod_{i=1}^{n} p_\theta(\mathbf{x}^{(i)}) \rightarrow \theta^* = \arg\max_\theta \sum_{i=1}^{n} \log p_\theta(\mathbf{x}^{(i)})$$

(Use **log likelihood** because sum is easier to work with than product)

Prior $p_\theta(\mathbf{z})$

Likelihood $p_\theta(\mathbf{x}|\mathbf{z})$

Posterior $p_\theta(\mathbf{z}|\mathbf{x})$

$$p_\theta(\mathbf{x}^{(i)}) = \int p_\theta(\mathbf{x}^{(i)}|\mathbf{z}) p_\theta(\mathbf{z}) d\mathbf{z}$$

**This is cool but intractable because we can't really sum over all possible z**

*Slide modified from Lilian Weng's Blog*

# Variational Autoencoder (VAE)

## Graphical Model Illustration

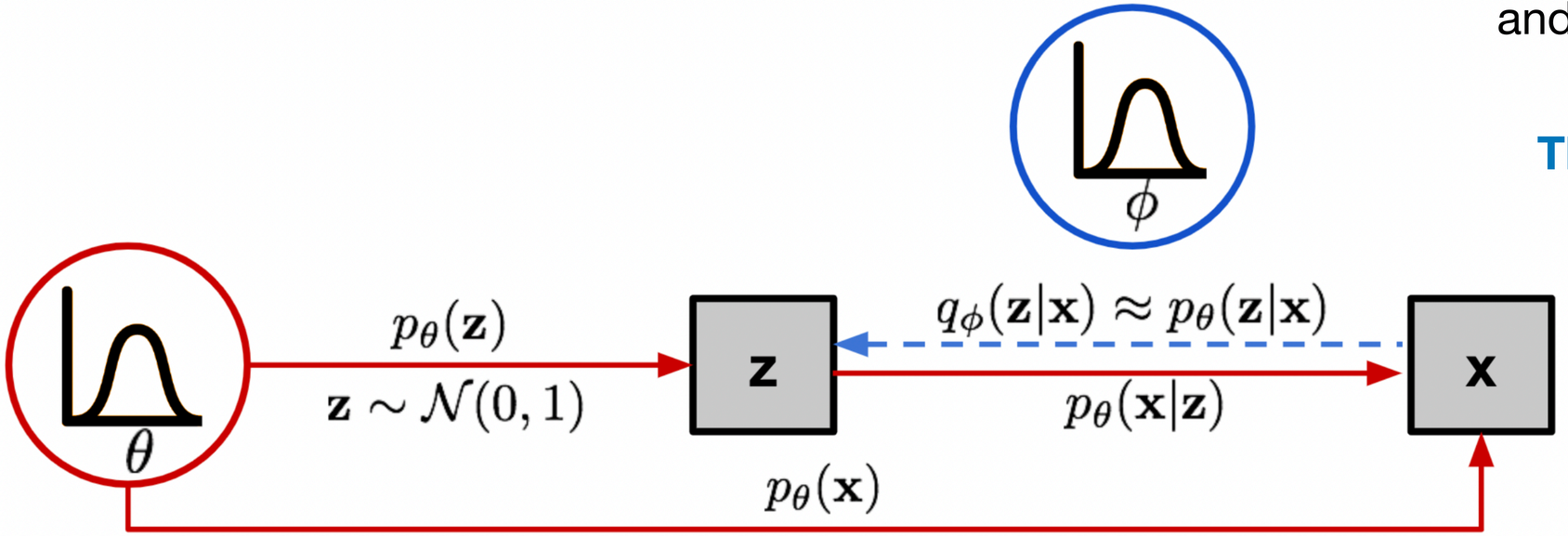



Fig. 6. The graphical model involved in Variational Autoencoder. Solid lines denote the generative distribution $p\_\theta(.)$ and dashed lines denote the distribution $q\_\phi(\mathbf{z}|\mathbf{x})$ to approximate the intractable posterior $p\_\theta(\mathbf{z}|\mathbf{x})$.

The **integral over z** is **intractable**, and the **posterior $p_\theta(z \mid x)$** is also **intractable**

**Therefore, we approximate the posterior via $q_\phi(z \mid x)$**

We call **$q_\phi(z \mid x)$** the **probabilistic encoder** and **$p_\theta(x \mid z)$** the **probabilistic decoder**

*Slide modified from Lilian Weng's Blog*

# Variational Autoencoder (VAE)

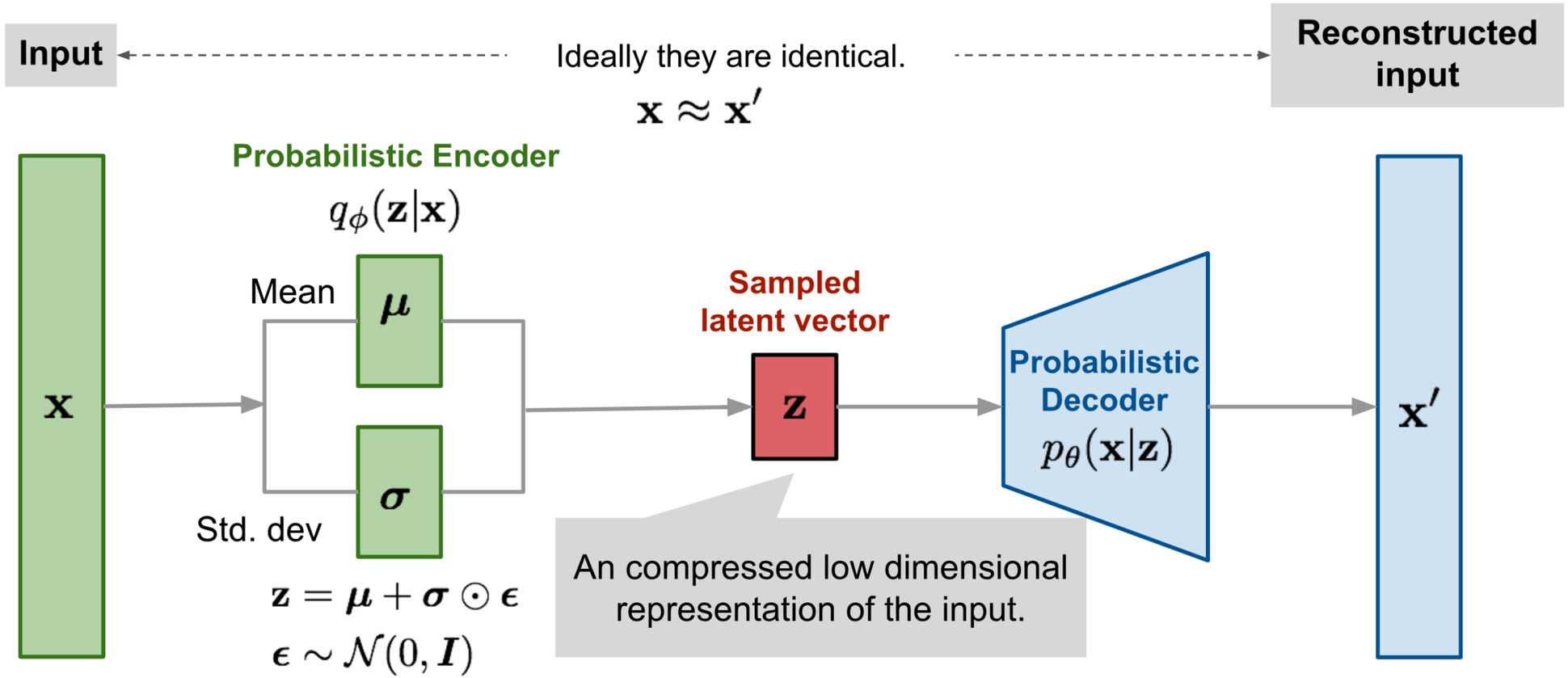



Figure taken from https://lilianweng.github.io/posts/2018-08-12-vae/

*Slide modified from Lilian Weng's Blog*

# Variational Autoencoder (VAE)

## Loss Function — Evidence Lower Bound (ELBO)

We need our probabilistic decoder distribution $\mathbf{q_\phi(z \mid x)}$ to be close to the posterior distribution $\mathbf{p_\theta(z \mid x)}$

(Otherwise our latent space may be something completely different)

We can make the distributions similar by **minimizing** the Kullback-Leibler (KL) Divergence

$$D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})|p_\theta(\mathbf{z}|\mathbf{x}))$$

*KL-Divergence quantifies how much information is lost when representing the left-hand distribution by the right-hand distribution (it is not symmetrical!)

*Slide modified from Lilian Weng's Blog*

# Variational Autoencoder (VAE)

## Loss Function — Evidence Lower Bound (ELBO)

$$
\begin{aligned}
& D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z}|\mathbf{x})) \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})} d\mathbf{z} \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x}) p_\theta(\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} d\mathbf{z} && ; \text{Because } p(z|x)=p(z,x)/p(x) \\
&= \int q_\phi(\mathbf{z}|\mathbf{x}) \big( \log p_\theta(\mathbf{x}) + \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} \big) d\mathbf{z} \\
&= \log p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}, \mathbf{x})} d\mathbf{z} && ; \text{Because } \textstyle\int q(z|x)dz=1 \\
&= \log p_\theta(\mathbf{x}) + \int q_\phi(\mathbf{z}|\mathbf{x}) \log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{x}|\mathbf{z}) p_\theta(\mathbf{z})} d\mathbf{z} && ; \text{Because } p(z,x)=p(x|z)p(z) \\
&= \log p_\theta(\mathbf{x}) + \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} [\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z})} - \log p_\theta(\mathbf{x}|\mathbf{z})] \\
&= \log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x}) \| p_\theta(\mathbf{z})) - \mathbb{E}_{\mathbf{z} \sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z})
\end{aligned}
$$

*Slide modified from Lilian Weng's Blog*

# Variational Autoencoder (VAE)

## Loss Function — Evidence Lower Bound (ELBO)

We end up with this equation:

$$D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) = \log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z})$$

Which we can arrange to the following one:

$$\log p_\theta(\mathbf{x}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}))$$

**This is exactly what we want to maximize:**
The **log likelihood of our data x**
**minus**
the difference between the **probabilistic decoder**
and the **true posterior** distributions

We can **maximize the left-hand side** by **maximizing the right-hand side**, or in other words, we can take the **negative of the right-hand side as our loss function and minimize it**

*Slide modified from Lilian Weng's Blog*

# Variational Autoencoder (VAE)

## Loss Function — Evidence Lower Bound (ELBO)

We end up with this equation:

$$D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) = \log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) - \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z})$$

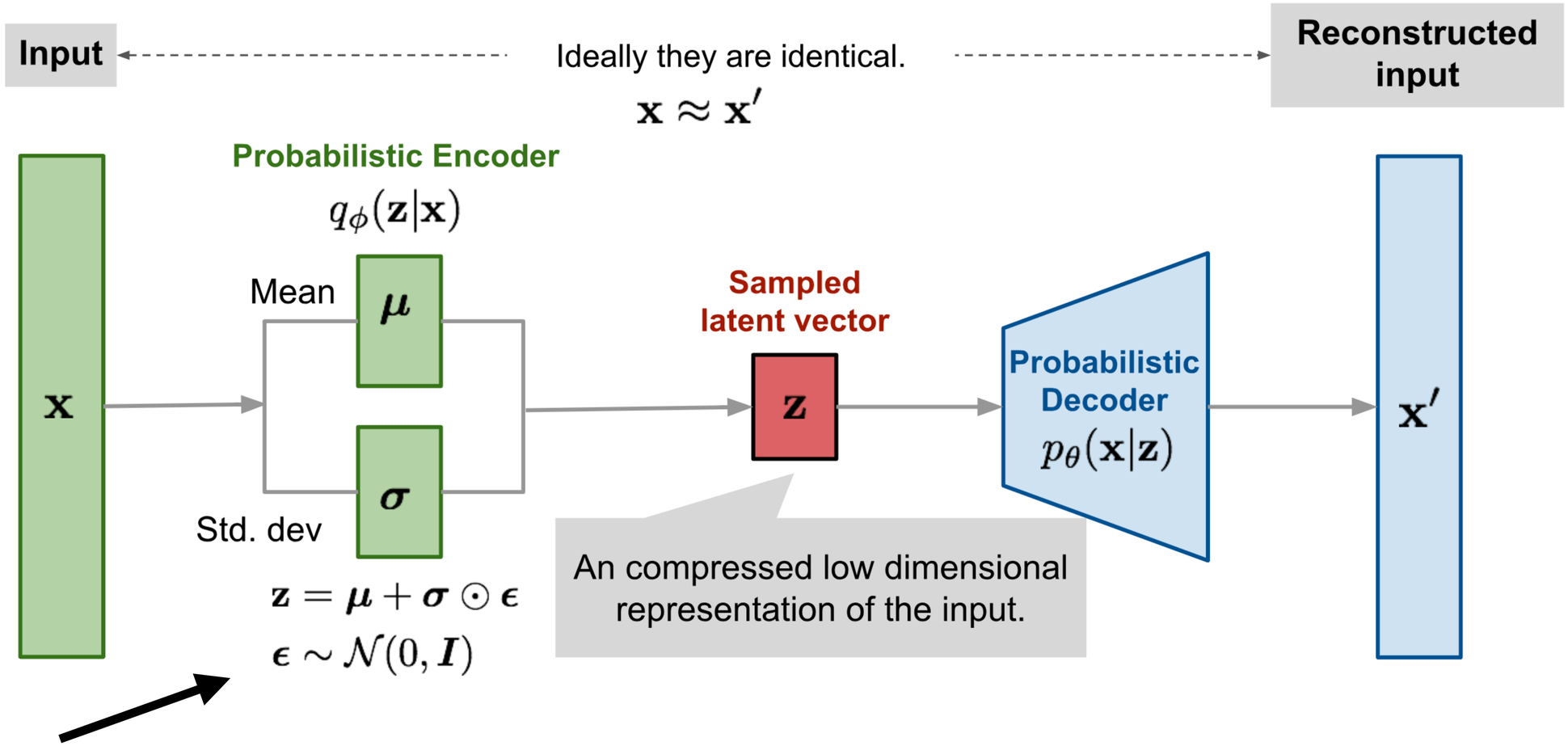
Which we can arrange to the following one:

$$\log p_\theta(\mathbf{x}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) = \mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) - D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}))$$

$$\begin{aligned} L_{\mathrm{VAE}}(\theta,\phi) &= -\log p_\theta(\mathbf{x}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z}|\mathbf{x})) \\ &= -\mathbb{E}_{\mathbf{z}\sim q_\phi(\mathbf{z}|\mathbf{x})} \log p_\theta(\mathbf{x}|\mathbf{z}) + D_{\mathrm{KL}}(q_\phi(\mathbf{z}|\mathbf{x})\|p_\theta(\mathbf{z})) \\ \theta^*,\phi^* &= \arg\min_{\theta,\phi} L_{\mathrm{VAE}} \end{aligned}$$

*Slide modified from Lilian Weng's Blog*

# Variational Autoencoder (VAE)

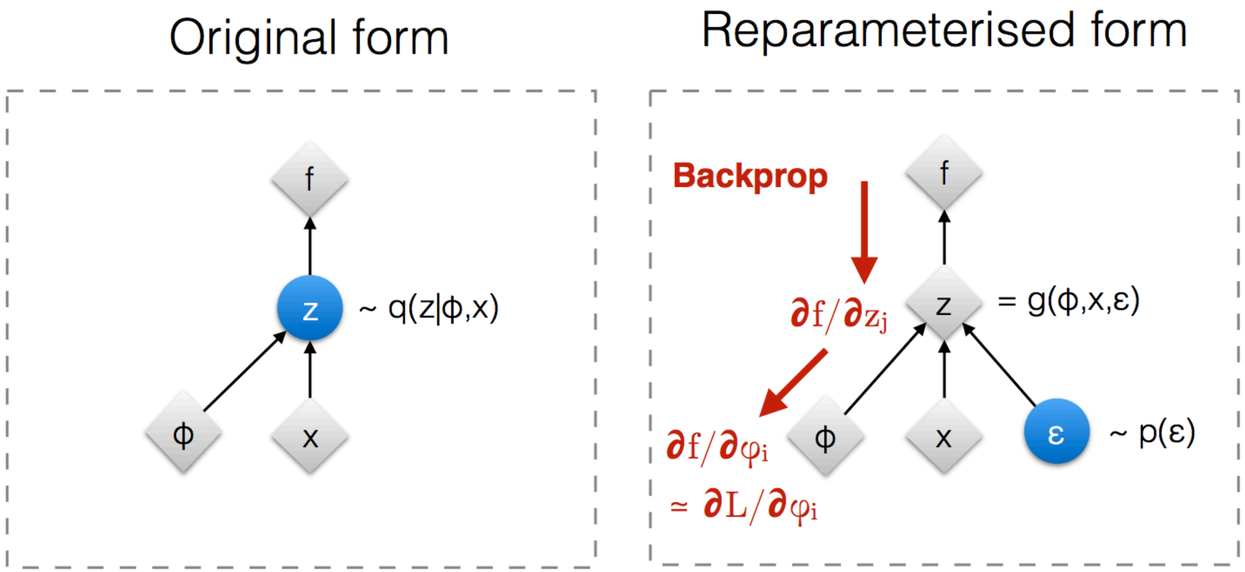



Note: **Sampling z is not a differentiable operation**, so we apply the **reparametrization trick**

Figure taken from https://lilianweng.github.io/posts/2018-08-12-vae/

# Variational Autoencoder (VAE)

## Reparametrization Trick



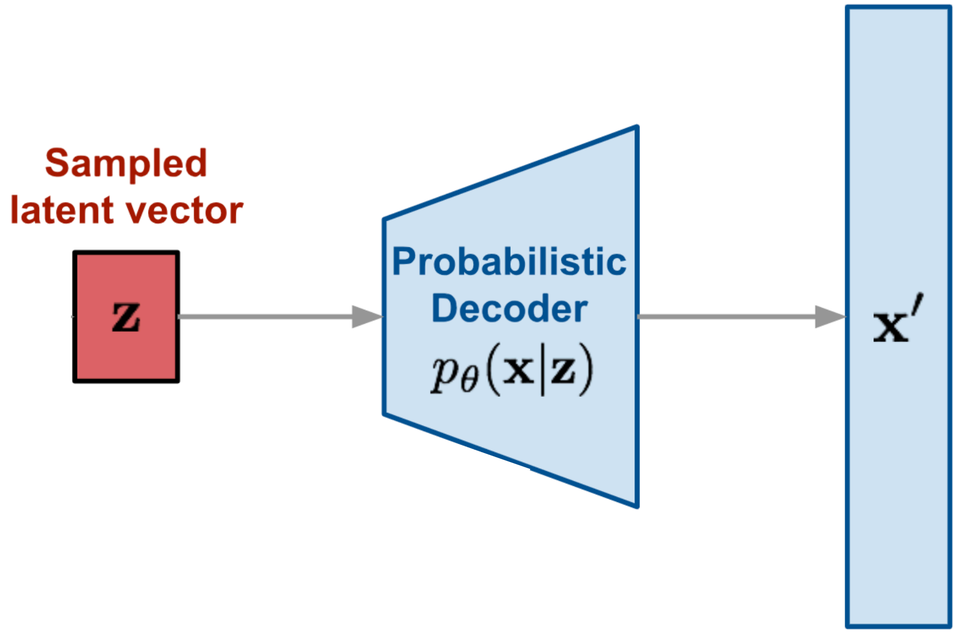

[Kingma, 2013]
[Bengio, 2013]
[Kingma and Welling 2014]
[Rezende et al 2014]

Original image source: http://dpkingma.com/wordpress/wp-content/uploads/2015/12/talk_nips_workshop_2015.pdf

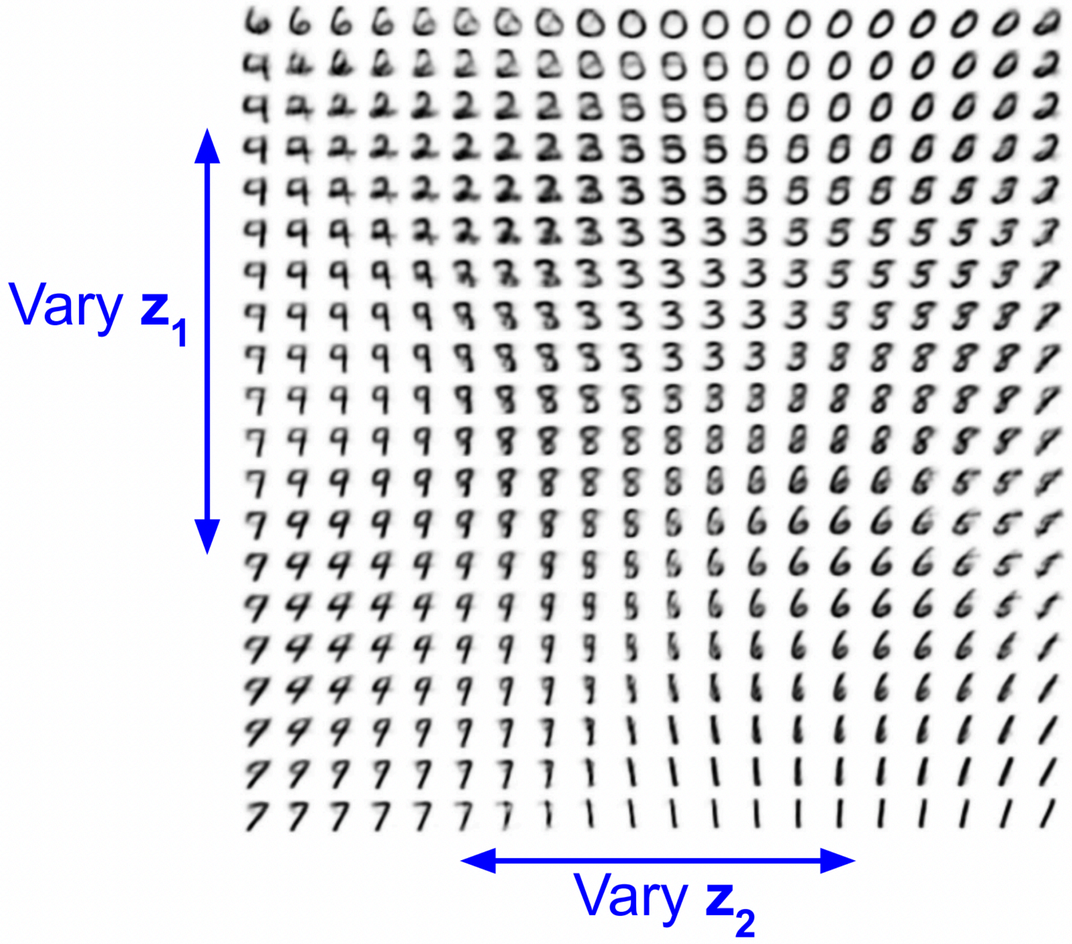
# Variational Autoencoder (VAE)

## Generating Data

Just sample z ~ N(0, 1) and decode



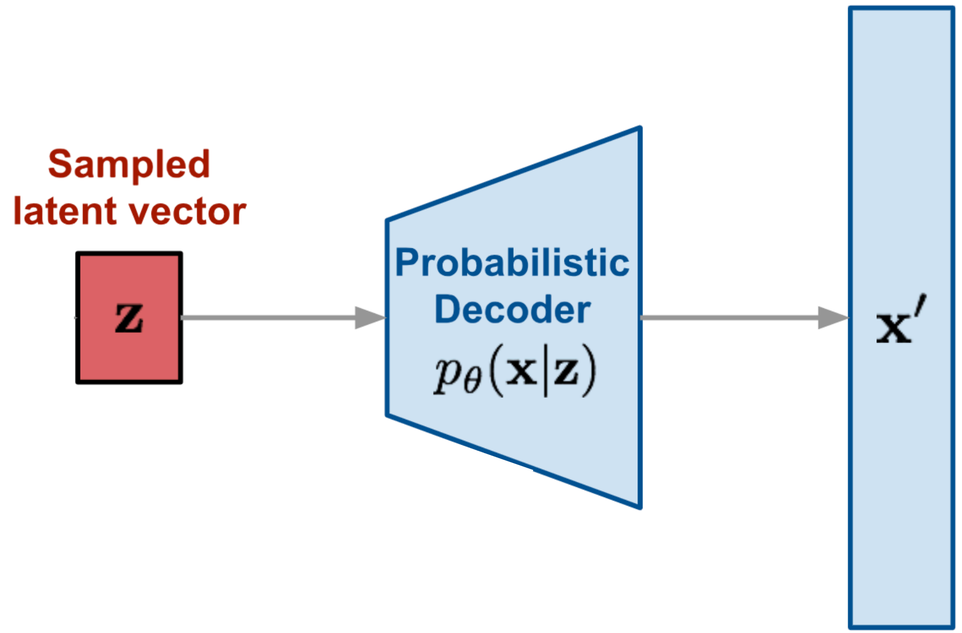

Data manifold for 2-d **z**

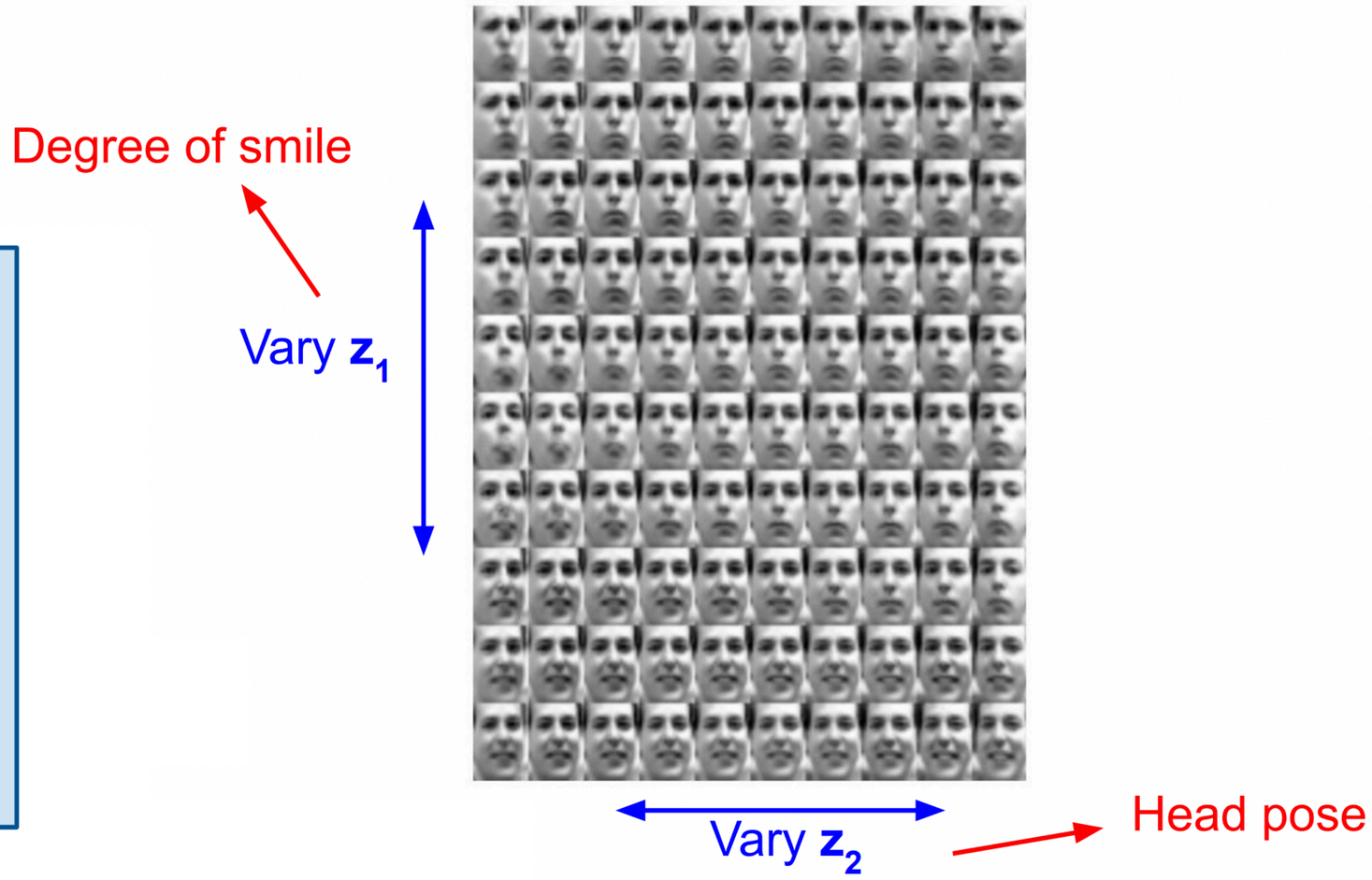


# Variational Autoencoder (VAE)

## Generating Data

Just sample z ~ N(0, 1) and decode



# Variational Autoencoder

## Pros and Cons

- **Pros:**
  - Interpretable latent space enabling e.g. interpolation
  - Probabilistic encoder can yield feature representations for transfer to supervised downstream tasks
- **Cons:**
  - We only optimize a lower bound on the data likelihood
  - Sample quality is worse than that of GANs (which we will now talk about)

# Today's Lecture

1. Supervised vs Unsupervised Learning
2. Generative Modeling
3. Applications of generative models
4. Fully-visible belief networks (FVBNs) & Autoregressive Decoders
5. (Variational) Autoencoders
6. **Generative Adversarial Networks (GANs)**

# Generative models — Taxonomy

**Maximum likelihood estimation**:
Find the set of parameters **θ*** that maximizes the likelihood of our data **x**

$$\boldsymbol{\theta}^* = \arg\max_{\boldsymbol{\theta}} \mathbb{E}_{x \sim p_{\text{data}}} \log p_{\text{model}}(\boldsymbol{x} \mid \boldsymbol{\theta})$$

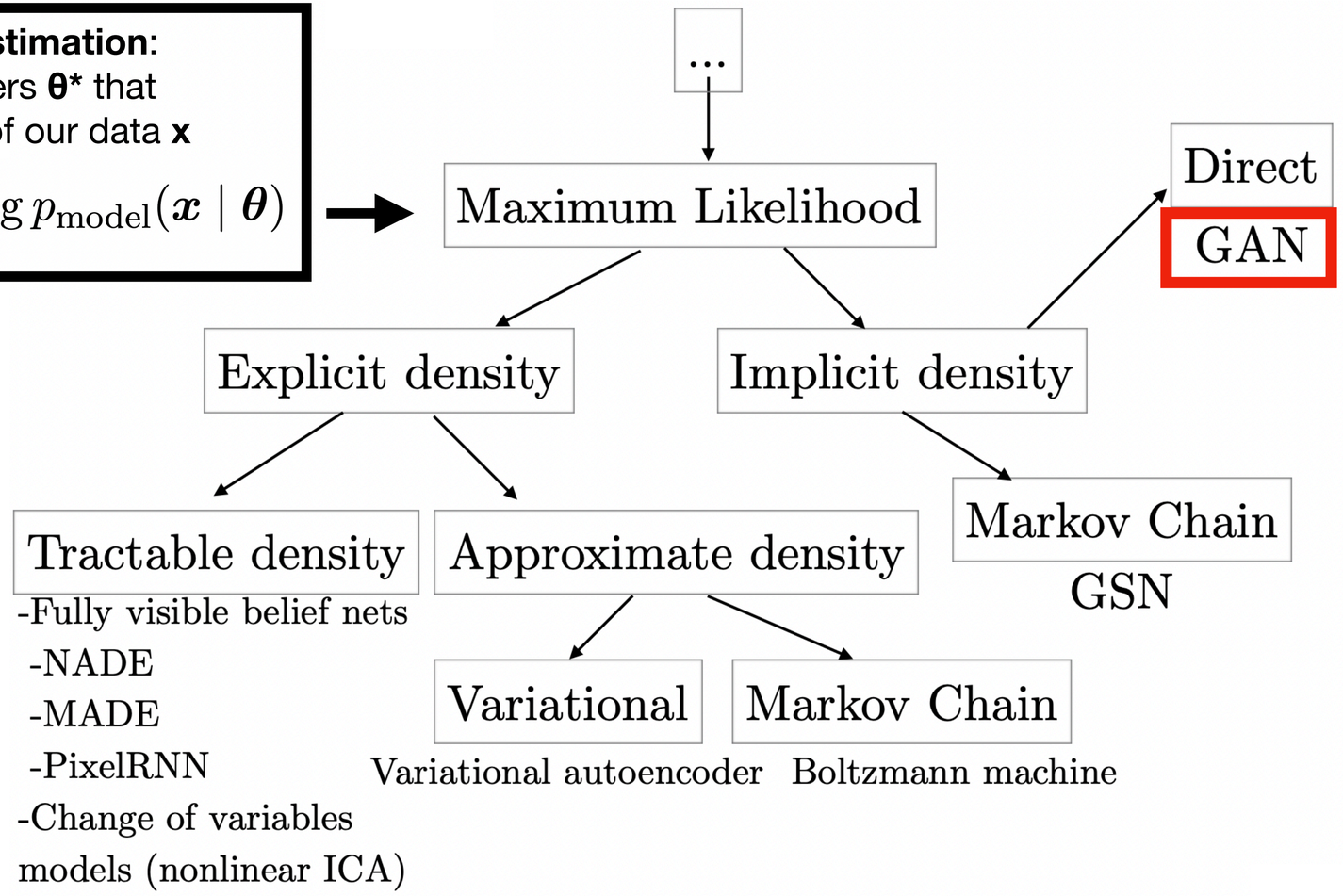


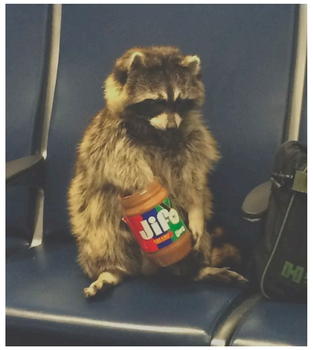

Figure taken from https://www.iangoodfellow.com/slides/2016-12-04-NIPS.pdf

# Generative Adversarial Networks (GANs)

- **Implicit** density estimation, i.e., we are **only** interested in l**earning to sample** from the data distribution **without actually knowing it**
- **Game-theoretic approach** that involves a **generator** and a **discriminator** network with **competing training objectives**

*Slide modified from Serena Yeung & Fei-Fei Li*

# Generative Adversarial Networks (GANs)



**Objective**: Generate images that look **real**

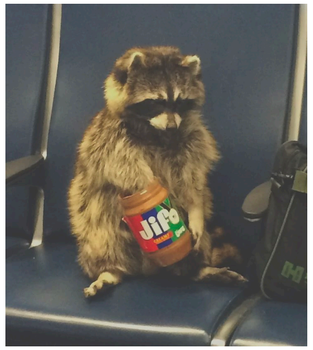
Generator Network

**Input: Sample random noise**

z

*Slide modified from Serena Yeung & Fei-Fei Li*

# Generative Adversarial Networks (GANs)



Discriminator Network

Real or Fake?

Gradients

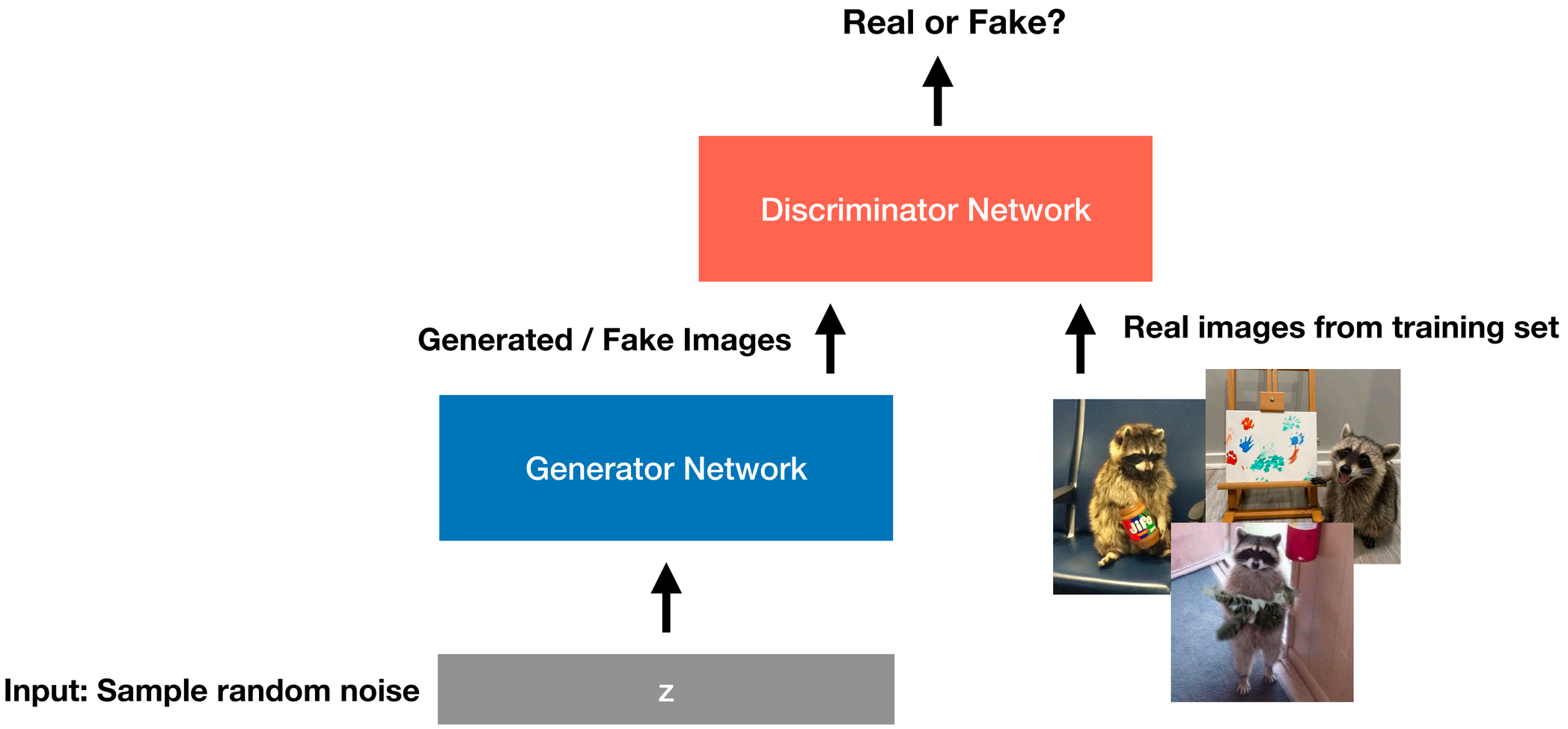
Generator Network

Input: Sample random noise

z

*Slide modified from Serena Yeung & Fei-Fei Li*

# Generative Adversarial Networks (GANs)



# Generative Adversarial Networks (GANs)

## Two-Player Minimax Game

- **Generator Objective**: Fool discriminator by generating images that look real
- **Discriminator Objective**: Learn to distinguish between “fake” and “real” images

**Discriminator outputs likelihood in (0, 1) of real image**

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

**Generator Objective**

**Discriminator Objective**

**Discriminator output for real image x**

**Discriminator output for fake image created by generator from sampled noise**

# Generative Adversarial Networks (GANs)

## Two-Player Minimax Game

Discriminator outputs likelihood in (0, 1) of real image

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

**Generator Objective**

**Discriminator Objective**

**Discriminator output for real image x**

**Discriminator output for fake image created by generator from sampled noise**

- **Discriminator objective**: D(x) close to 1, D(G(z)) close to 0
- **Generator objective:** D(G(z)) close to 1

# Generative Adversarial Networks (GANs)

## Two-Player Minimax Game

$$\min_{\theta_g} \max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between two optimization problems:

1. **Gradient ascent** on **discriminator**

$$\max_{\theta_d} \left[ \mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient ascent** on **generator**

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Note: **Maximizing** the probability of the discriminator being wrong is more stable than **minimizing E[log(1-D(G(z))]** (the probability of the discriminator being right) because of gradient flow for bad generated samples in the beginning

# Generative Adversarial Networks (GANs)

## Two-Player Minimax Game

**After enough training**, the generator and discriminator reach the **Nash-Equilibrium** at which **neither can improve**

Now we can throw away the discriminator and happily generate samples with the generator

# Generative Adversarial Networks (GANs)

## Samples

- Try it out yourself: https://thispersondoesnotexist.com/
- Interpolation:


Sauer et al. (2022), StyleGAN-XL: Scaling StyleGAN to Large Diverse Datasets

# Suggested and Further Resources

- **The Illustrated GPT-2**: https://jalammar.github.io/illustrated-gpt2/
- **Autoencoder Blog Post** (Includes **Vector-Quantized VAE**, a state-of-the-art VAE used in 2022): https://lilianweng.github.io/posts/2018-08-12-vae/
- **NIPS 2016 Tutorial: Generative Adversarial Networks**, Ian Goodfellow
- **StyleGAN Paper** (Karras et al., 2019)
- **FAIR PyTorch GAN Zoo:** https://github.com/facebookresearch/pytorch_GAN_zoo