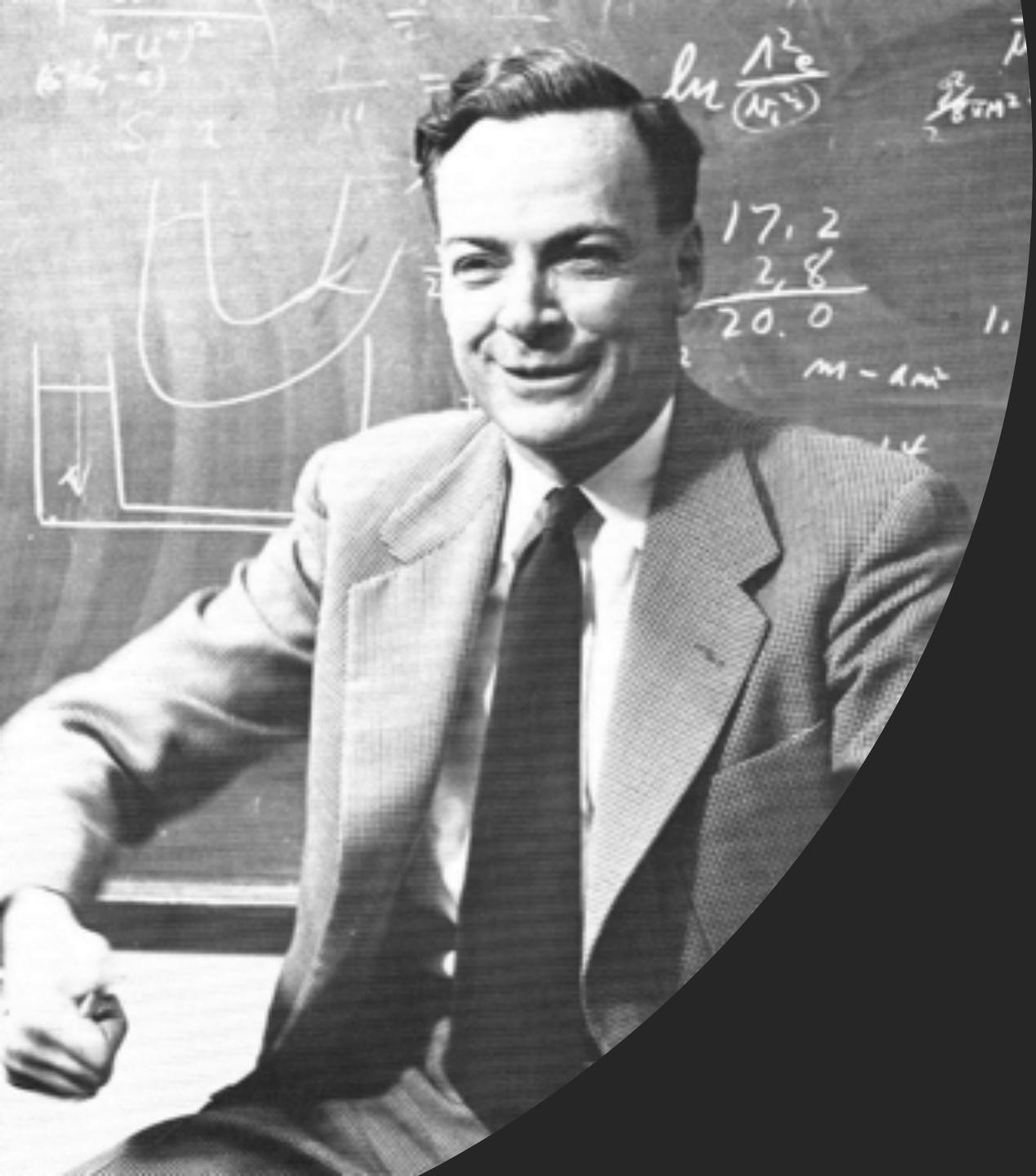




Interacting with Deep Generative Models for Content Creation

Bolei Zhou

The Chinese University of Hong Kong

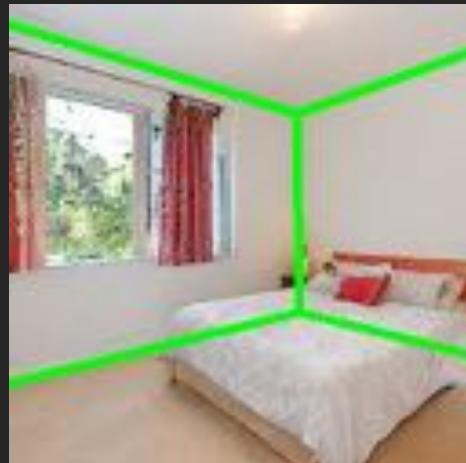


Richard Feynman:
“What I cannot create,
I do not understand”

Inference and Generation

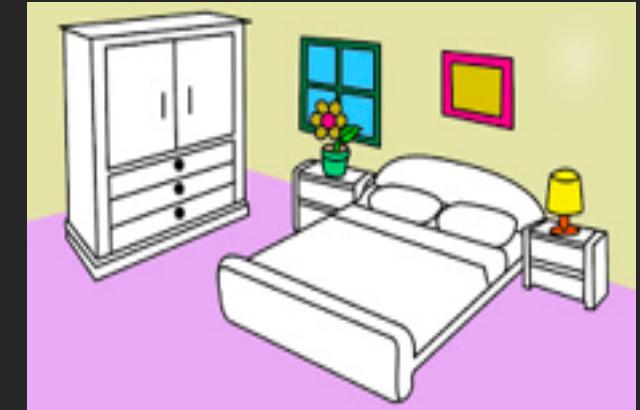
Predictions:
bedroom
wood
foliage

Inference



layout

Generation



Statistical Generative Models

- Data: samples such as images of bedrooms
- Prior knowledge: distribution functions (Gaussian, or other non-linear?), loss function (e.g., maximum likelihood, adversarial loss?),



Learn

probability
distribution $p(x)$

Sample



Progress for Image Generation



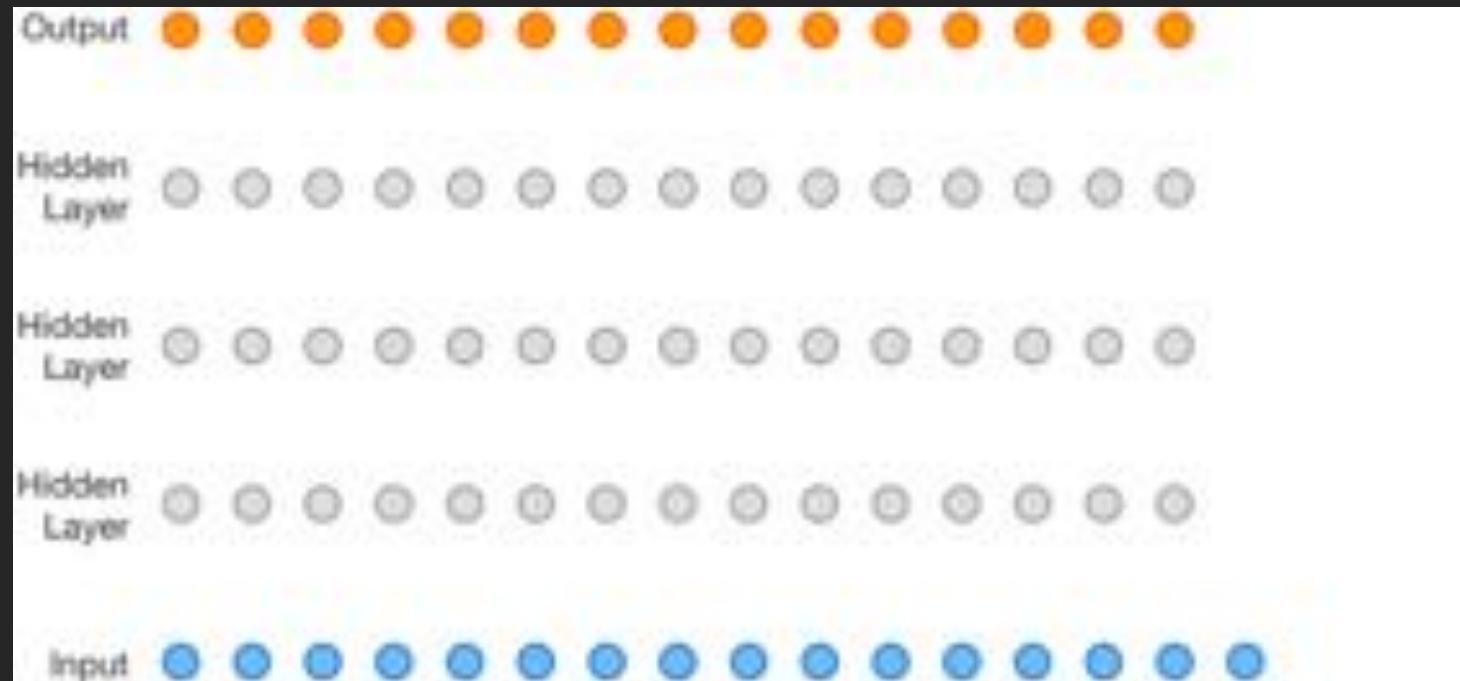
PG-GAN

StyleGAN

BigGAN

Progress for Speech Synthesis

WaveNet



Text to Speech

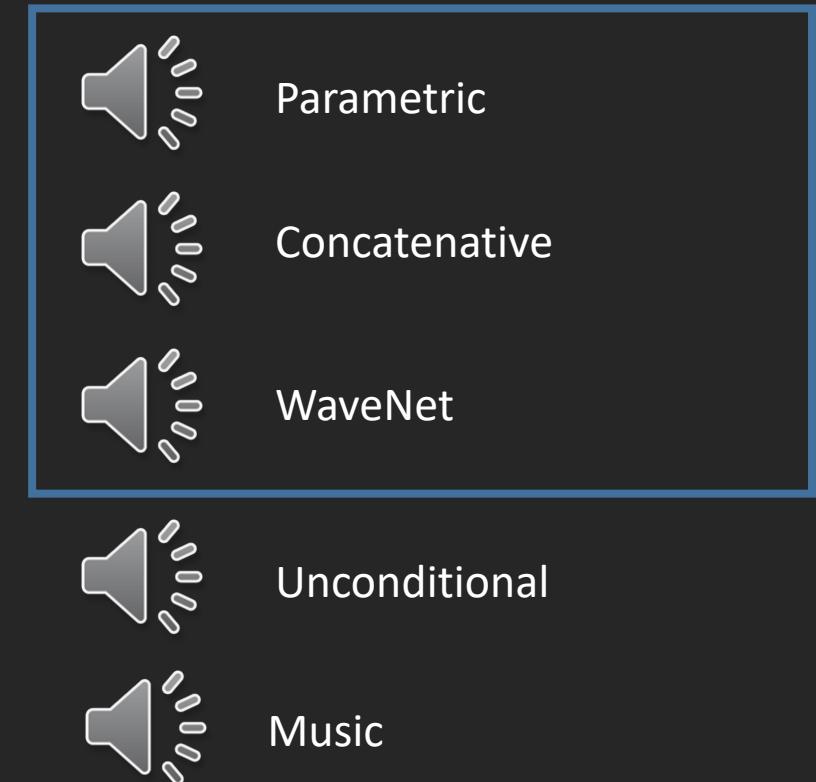
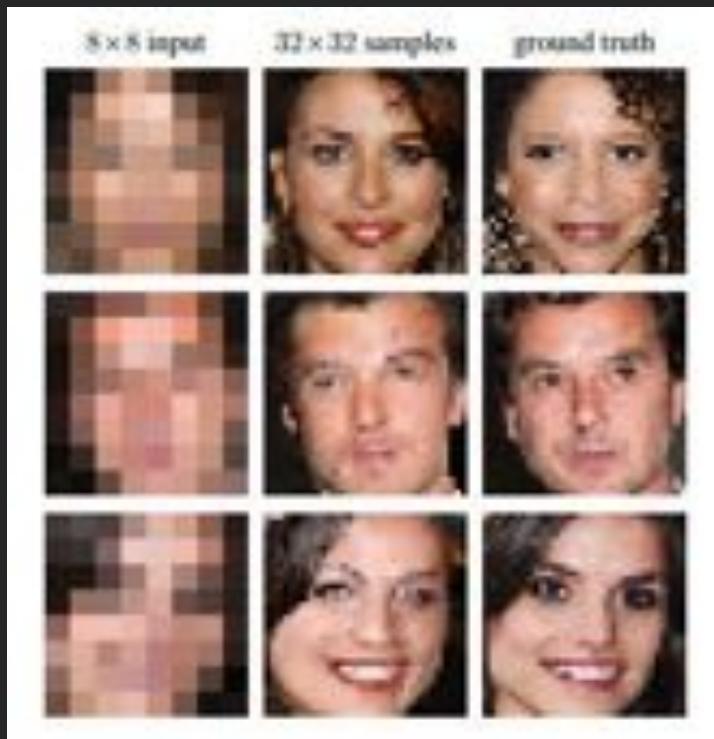


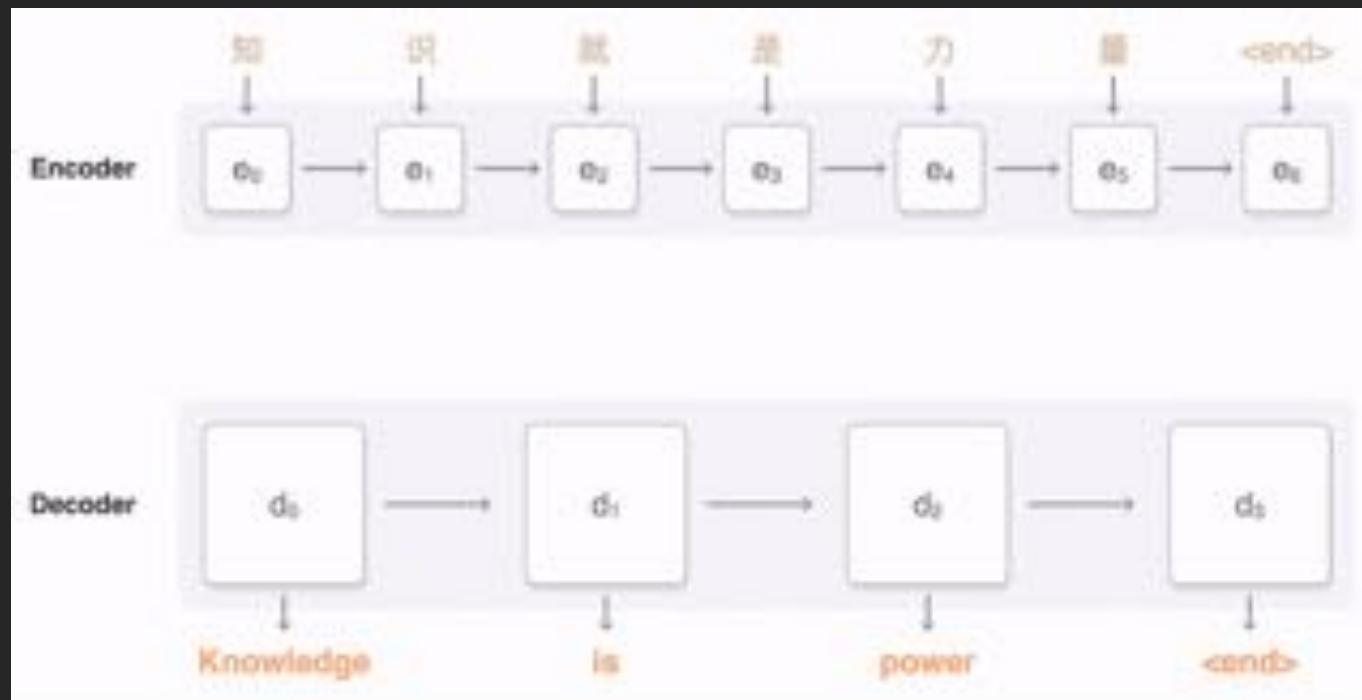
Image Super Resolution

Conditional generative model $P(\text{high res image} \mid \text{low res image})$

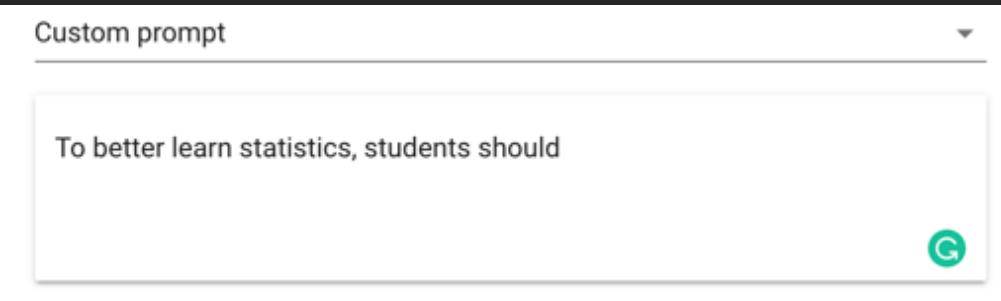


Machine Translation

Conditional generative model $P(\text{ English text} | \text{ Chinese text})$



Paragraph Generation



Completion

To better learn statistics, students should understand the functions of probability, statistics, and probability distributions. They should also understand the basic logic that underlies statistics and find their own ideas and methods for applying the logic to new problems. Their basic understanding of probability and statistics should provide the foundation they will need to think creatively and critically when dealing with critical situations and problems. They should understand the information we can find out by running statistical tests, and they should understand the implications and limitations of such information. They should be able to apply basic math concepts, statistics, probability and mathematics to their own particular circumstances and to teach other students. They should be able to analyze and critique the many issues of statistics that are in our society. They should be able to tell about possible sources

DeepFakes (video+audio)



Image Translation

Conditional generative model $P(\text{ zebra images} | \text{ horse images})$



Tutorial Outline

- Session 1: Overview of the generative models
- Session 2: Interacting with the generative models

Session 1: Overview of Generative Models

- Likelihood-based models: Autoregressive models
- Latent variable models: Variational Autoencoder (VAE)
- Implicit generative models: Generative Adversarial Networks (GANs)

Autoregressive generative model

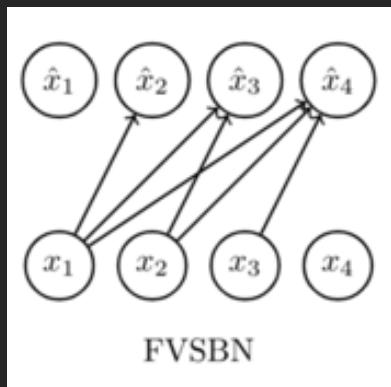
- Given a dataset of handwritten digits (binarized MNIST)



- Each image has $n = 28 \times 28 = 784$ pixels, with each pixel either being black (0) or white (1)
- Goal: Learn a probability distribution $p(x) = p(x_1, \dots, x_{784})$

Autoregressive generative model

- We can use chain rule for factorization:
 - $p(x_1, \dots, x_{784}) = p(x_1)p(x_2|x_1)p(x_3|x_1,x_2)\dots p(x_n|x_1,\dots,x_{n-1})$
- Then we can have the following simple autoregressive model
 - $P(x_1=1;a_1) = a_1, P(x_1=0;a_1) = 1-a_1$
 - $P(x_2=1|x_1;a_2) = \text{logit}(a_{02}+a_{12}x_1)$
 - $P(x_3=1|x_1,x_2;a_3) = \text{logit}(a_{03}+a_{13}x_1+a_{23}x_2)$

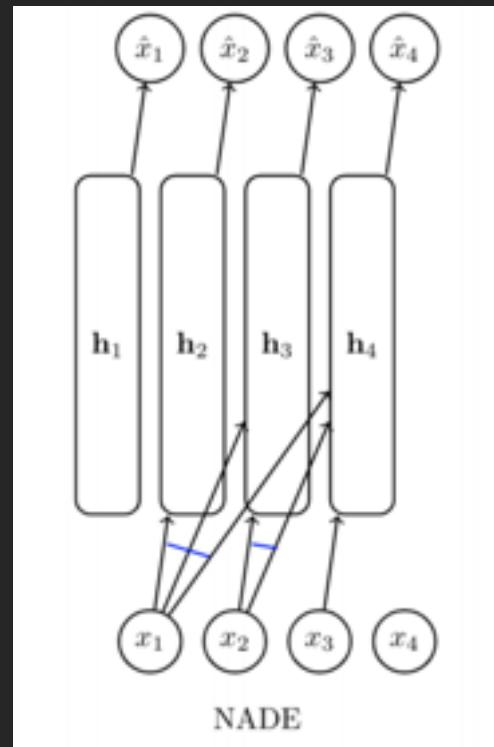


$$\hat{x}_i = p(X_i = 1|x_1, \dots, x_{i-1}; \boldsymbol{\alpha}^i) = p(X_i = 1|x_{<i}; \boldsymbol{\alpha}^i) = \sigma(\alpha_0^i + \sum_{j=1}^{i-1} \alpha_j^i x_j)$$

Fully visible sigmoid belief network (FVSBN)

NADE: Neural Autoregressive Density Estimation

To improve the factorization model: use one-layer neural network instead of logistic regression



$$\hat{x}_i = p(x_i | x_1, \dots, x_{i-1}; \underbrace{A_i, \mathbf{c}_i, \boldsymbol{\alpha}_i, b_i}_{\text{parameters}}) = \sigma(\boldsymbol{\alpha}_i \mathbf{h}_i + b_i)$$
$$\mathbf{h}_i = \sigma(A_i \mathbf{x}_{<i} + \mathbf{c}_i)$$

$$\mathbf{h}_2 = \sigma \left(\underbrace{\begin{pmatrix} \vdots \\ \mathbf{w}_1 \\ \vdots \end{pmatrix}}_{A_2} x_1 \right) \quad \mathbf{h}_3 = \sigma \left(\underbrace{\begin{pmatrix} \vdots & \vdots \\ \mathbf{w}_1 & \mathbf{w}_2 \\ \vdots & \vdots \end{pmatrix}}_{A_3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} \right) \quad \mathbf{h}_4 = \sigma \left(\underbrace{\begin{pmatrix} \vdots & \vdots & \vdots \\ \mathbf{w}_1 & \mathbf{w}_2 & \mathbf{w}_3 \\ \vdots & \vdots & \vdots \end{pmatrix}}_{A_3} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} \right)$$

Training data



Samples from the probability



Figure from The Neural Autoregressive Distribution Estimator, 2011.

RNN: Recurrent Neural Networks

Suppose $x_i \in \{h, e, l, o\}$. Use one-hot encoding:

- h encoded as $[1, 0, 0, 0]$, e encoded as $[0, 1, 0, 0]$, etc.

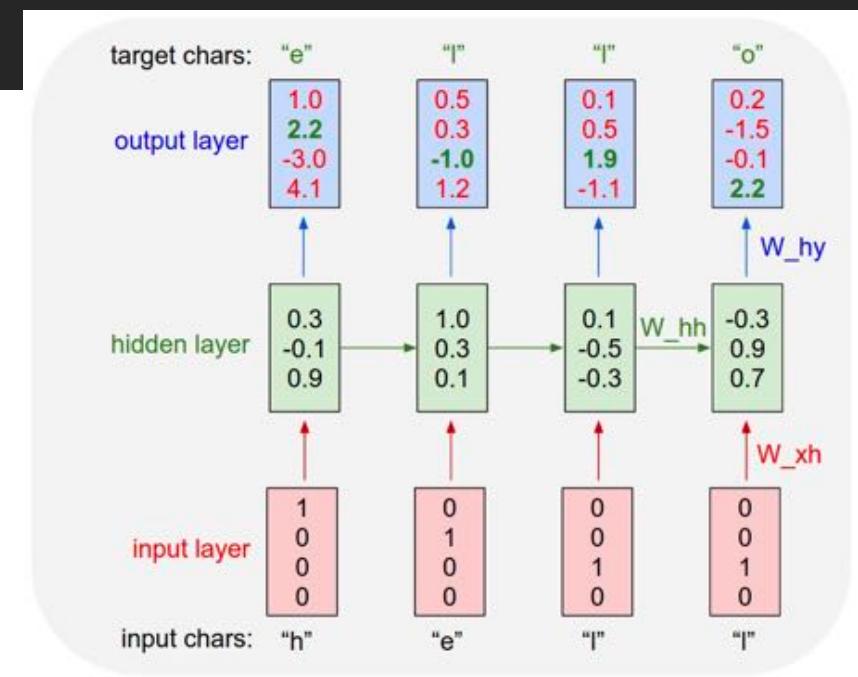
Autoregressive: $p(x = \text{hello}) = p(x_1 = h)p(x_2 = e|x_1 = h)p(x_3 = l|x_1 = h, x_2 = e) \cdots p(x_5 = o|x_1 = h, x_2 = e, x_3 = l, x_4 = l)$

For example,

$$p(x_2 = e|x_1 = h) = \text{softmax}(o_1) = \frac{\exp(2.2)}{\exp(1.0) + \cdots + \exp(4.1)}$$

$$o_1 = W_{hy} h_1$$

$$h_1 = \tanh(W_{hh} h_0 + W_{xh} x_1)$$



RNN: Recurrent Neural Networks

RNN model on Wikipedia

Naturalism and decision for the majority of Arab countries' capitalide was grounded by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal in the [[Protestant Immineners]], which could be said to be directly in Cantonese Communication, which followed a ceremony and set inspired prison, training. The emperor travelled back to [[Antioch, Perth, October 25|21]] to note, the Kingdom of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known in western [[Scotland]], near Italy to the conquest of India with the conflict. Copyright was the succession of independence in the slop of Syrian influence that was a famous German movement based on a more popular servicious, non-doctrinal and sexual power post. Many governments recognize the military housing of the [[Civil Liberalization and Infantry Resolution 265 National Party in Hungary]], that is sympathetic to be to the [[Punjab Resolution]] (PJS)[<http://www.humah.yahoo.com/guardian>.

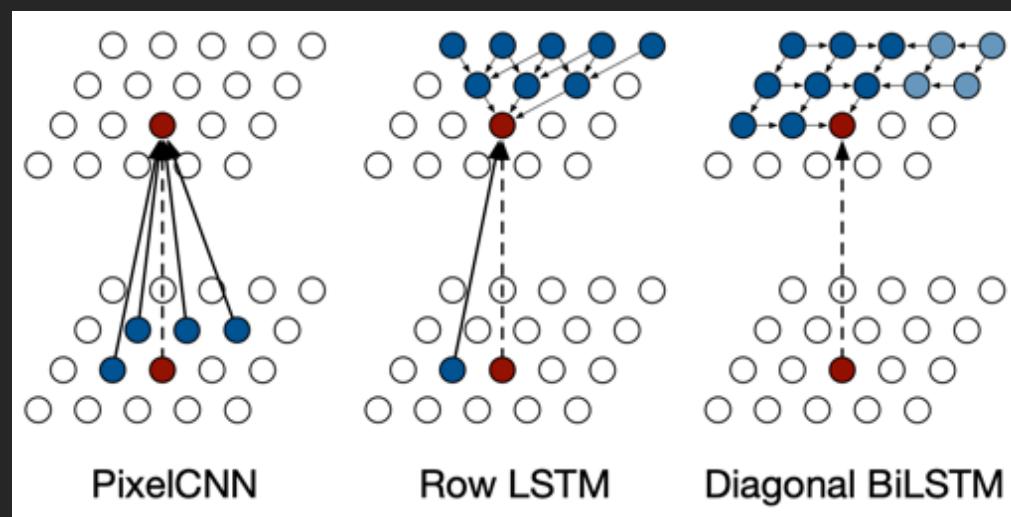
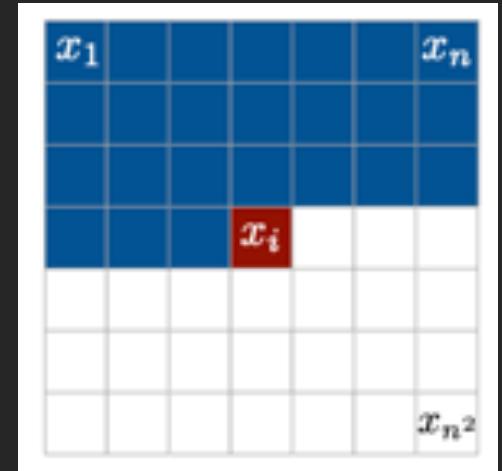
RNN model on Linux source code

```
/*
 * Increment the size file of the new incorrect UI_FILTER group information
 * of the size generatively.
 */
static int indicate_policy(void)
{
    int error;
    if (fd == MARN_EPT) {
        /*
         * The kernel blank will coeld it to userspace.
         */
        if (ss->segment < mem_total)
            unblock_graph_and_set_blocked();
        else
            ret = 1;
        goto bail;
    }
    segaddr = in_SB(in.addr);
    selector = seg / 16;
    setup_works = true;
    for (i = 0; i < blocks; i++) {
        seq = buf[i++];
        bpf = bd->bd.next + i * search;
        if (fd) {
            current = blocked;
        }
    }
}
```

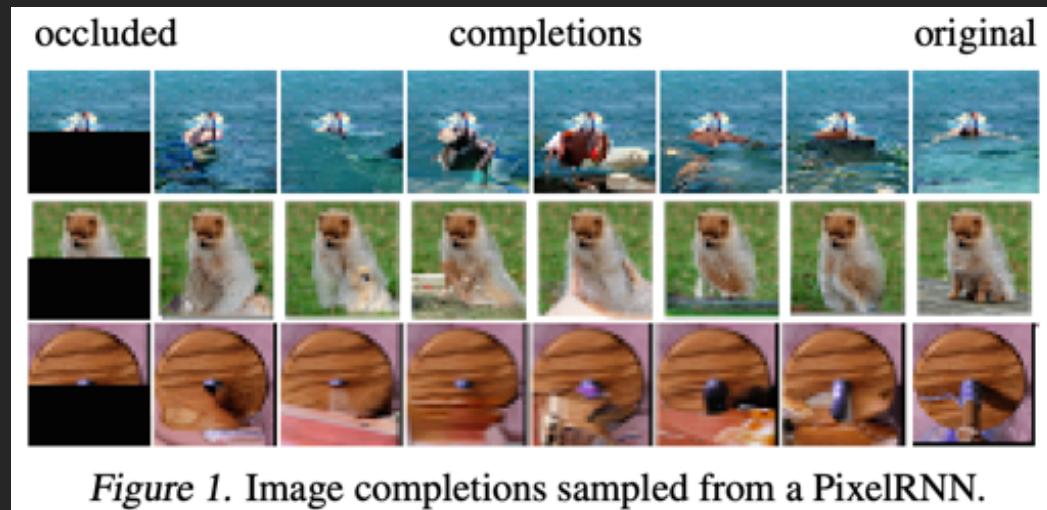
PixelRNN and PixelCNN (Oord et al. 2016)

Model image pixel by pixel using raster scan order

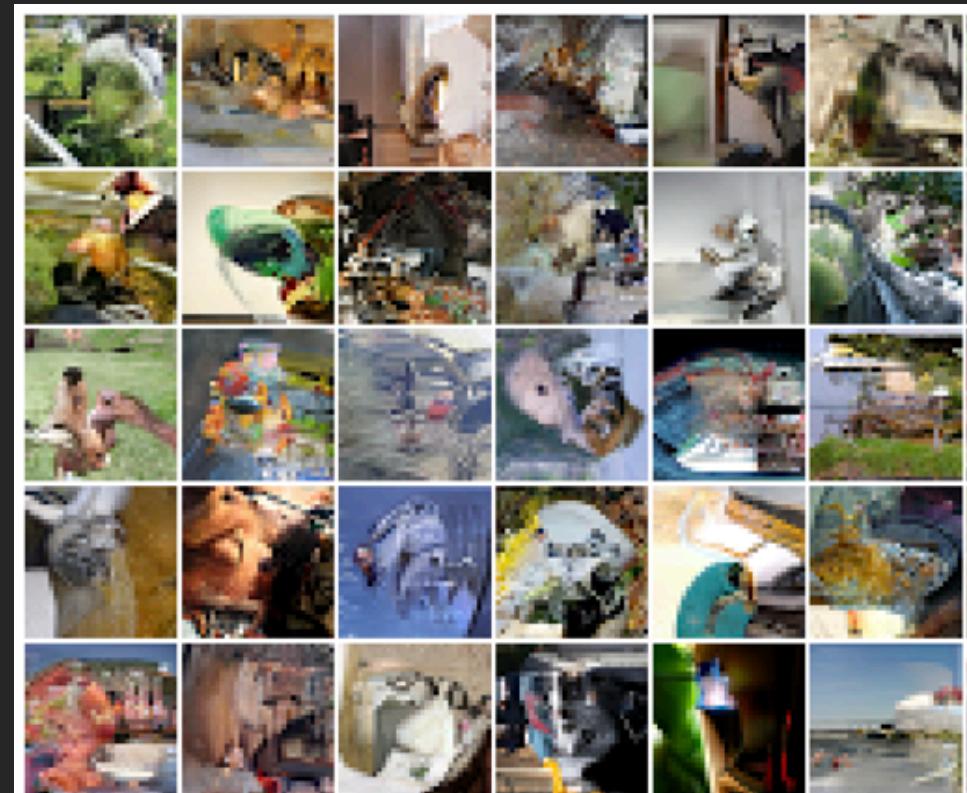
$$p(x_t \mid x_{1:t-1}) = p(x_t^{\text{red}} \mid x_{1:t-1})p(x_t^{\text{green}} \mid x_{1:t-1}, x_t^{\text{red}})p(x_t^{\text{blue}} \mid x_{1:t-1}, x_t^{\text{red}}, x_t^{\text{green}})$$



PixelCNN and PixelRNN (Oord et al. 2016)

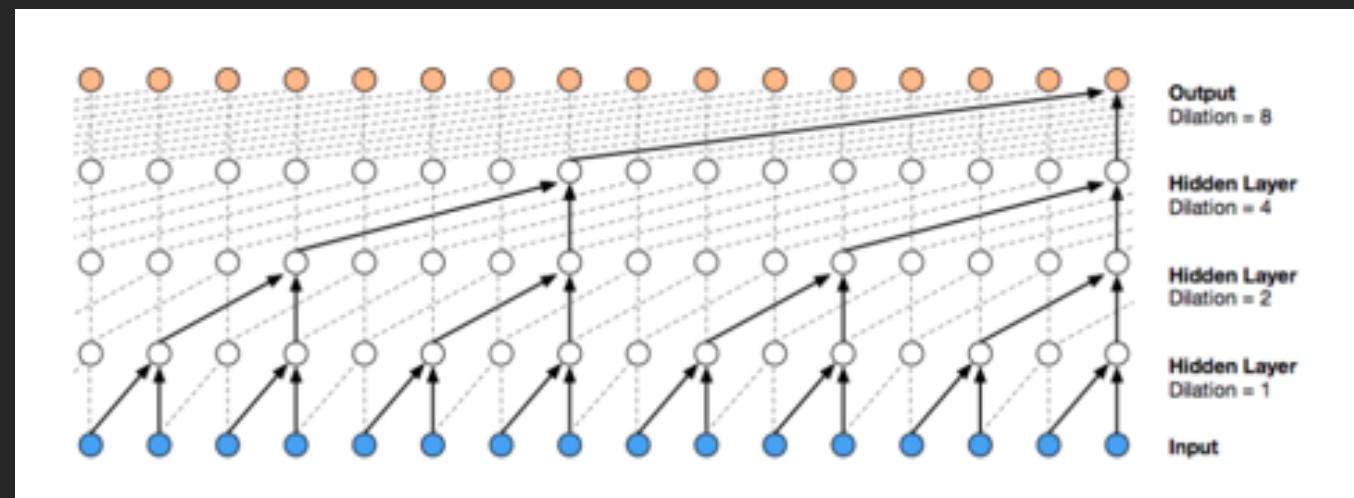
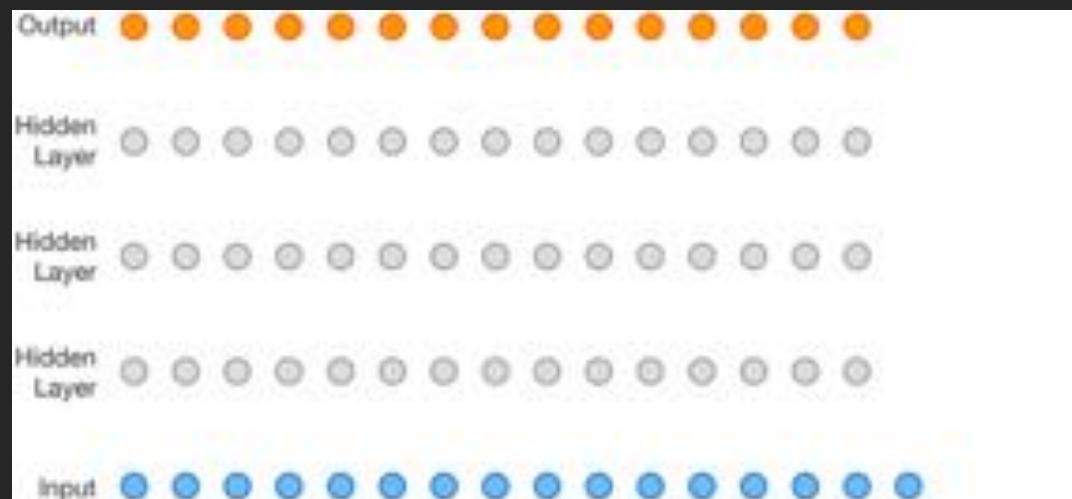


Samples from PixelRNN trained on ImageNet



WaveNet (Oord et al. 2016)

- CNN with 1D dilated convolutions



Summary of Autoregressive Models

- Autoregressive models:
 - Chain-rule factorization is fully general
 - Compact representation via conditional independence and/or neural parameterizations
- Pros:
 - Easy to evaluate likelihoods
 - Easy to train
- Cons:
 - Requires an ordering
 - Generation is sequential
 - Cannot learn features in an unsupervised way

Latent Variable Models

- Gender, eye color, race, pose are the variation factors in images, unless annotated, they are not explicitly available
- Then we can model them as (low-dimensional) **latent** variables

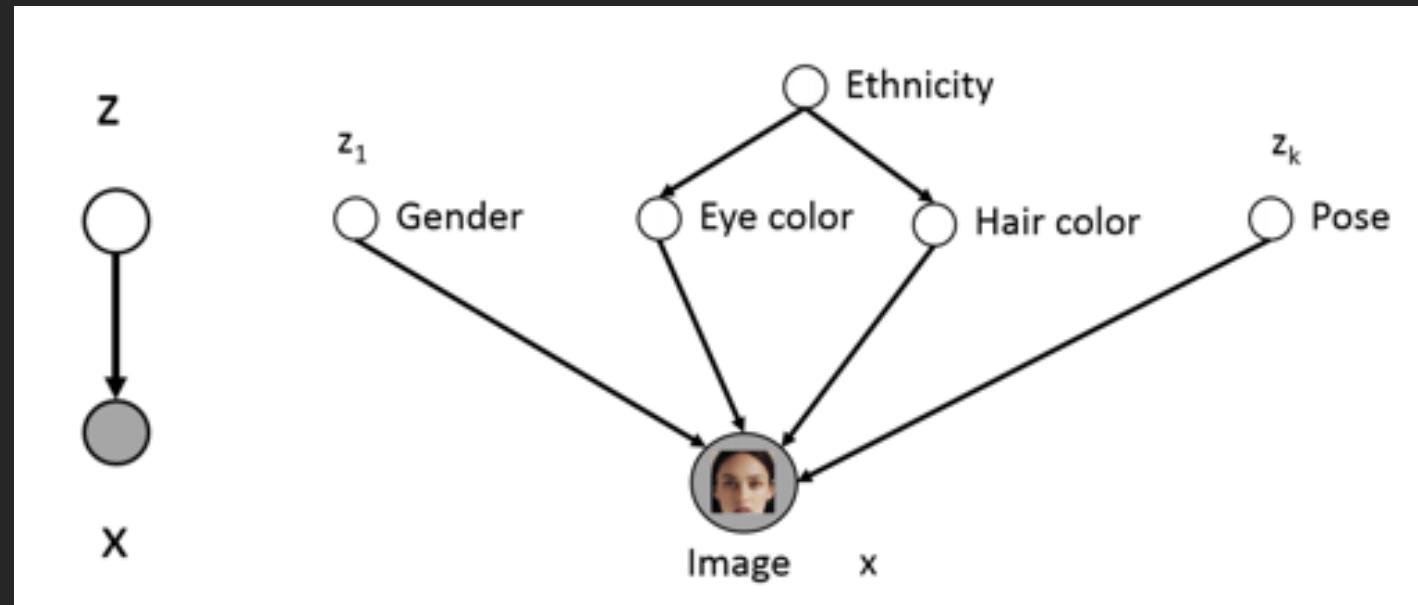


Latent Variable Models

Data generation process could be driven by the latent variables

Latent variables

Observation



Latent Variable Models

- We can define complex models $p(x)$ in terms of simple building blocks $p(x|z)$
- It can be applied for unsupervised learning tasks easily
- Challenge: much more difficult to learn due to the latent variables.

A mixture of an infinite number of Gaussians:

① $\mathbf{z} \sim \mathcal{N}(0, I)$

② $p(\mathbf{x} | \mathbf{z}) = \mathcal{N}(\mu_\theta(\mathbf{z}), \Sigma_\theta(\mathbf{z}))$ where $\mu_\theta, \Sigma_\theta$ are neural networks

③ Even though $p(\mathbf{x} | \mathbf{z})$ is simple, the marginal $p(\mathbf{x})$ is very complex/flexible

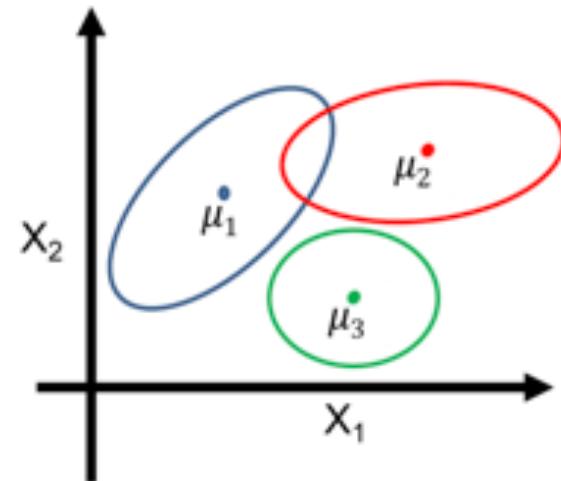
Mixture of Gaussians as a simple latent variable model

Generative process:

- Pick a mixture component k by sampling z
- Generate a data point by sampling from that Gaussian

Mixture of Gaussians. Bayes net: $z \rightarrow x$.

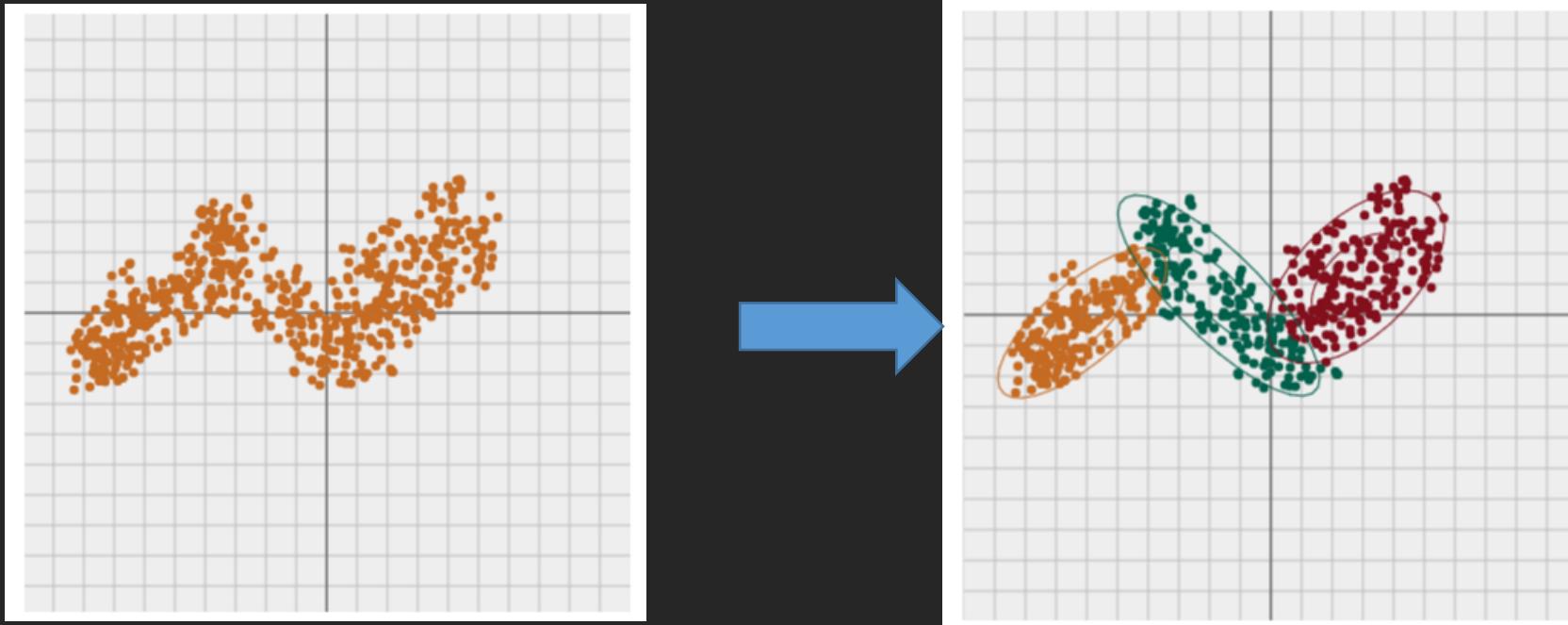
- ➊ $z \sim \text{Categorical}(1, \dots, K)$
- ➋ $p(x | z = k) = \mathcal{N}(\mu_k, \Sigma_k)$



Mixture of Gaussians as a simple latent variable model

- Unsupervised learning: the posterior $p(z|x)$ identifies the mixture component

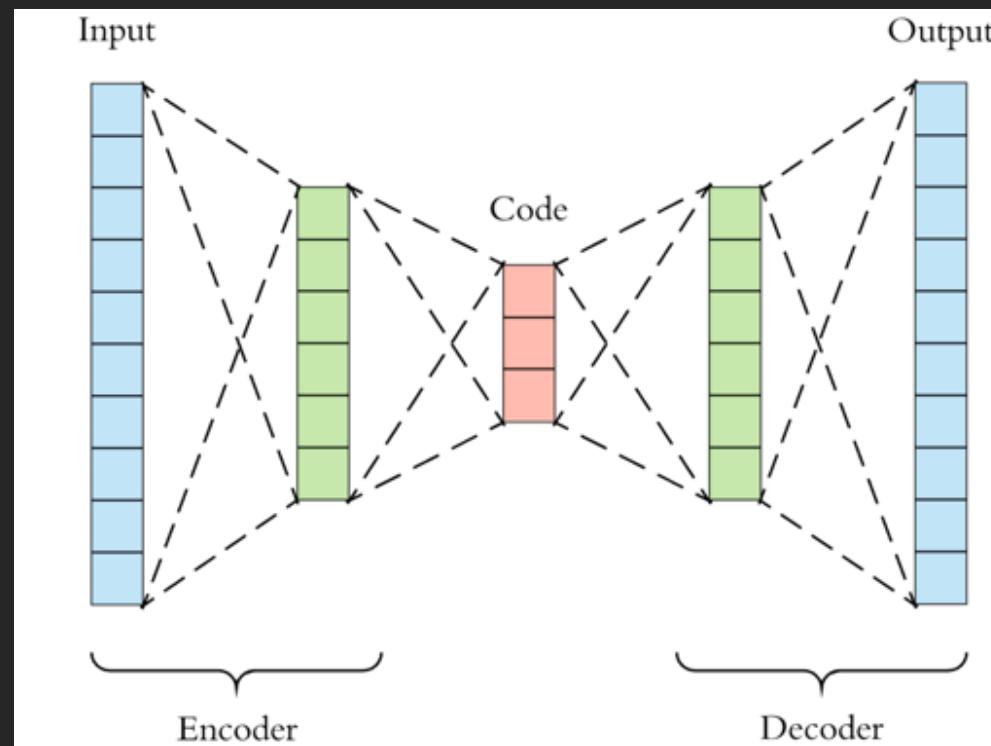
$$i = \operatorname{argmax} p(z=i|x)$$



$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) = \sum_{\mathbf{z}} p(\mathbf{z})p(\mathbf{x} | \mathbf{z}) = \sum_{k=1}^K p(\mathbf{z} = k) \underbrace{\mathcal{N}(\mathbf{x}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}_{\text{component}}$$

Autoencoder

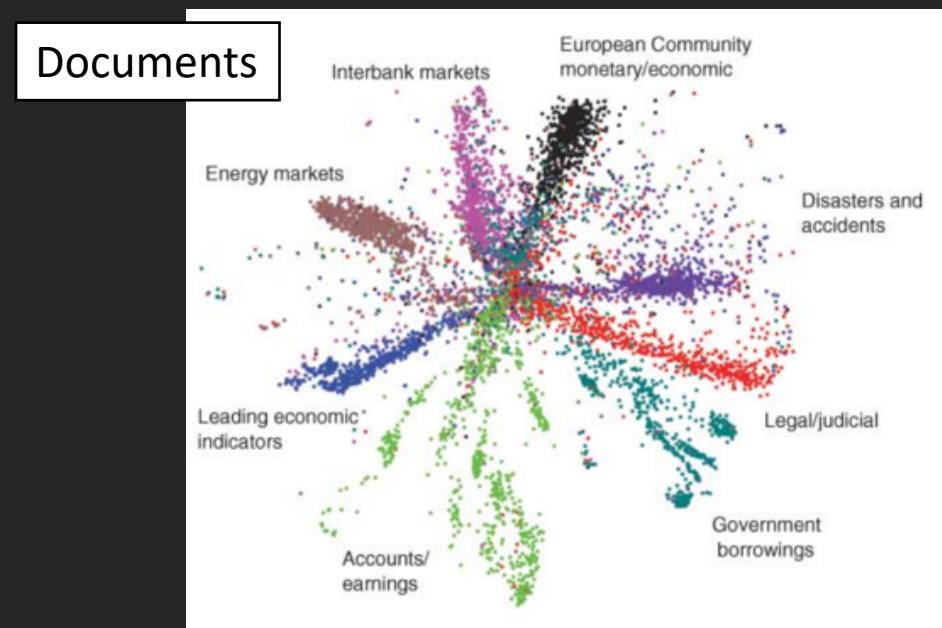
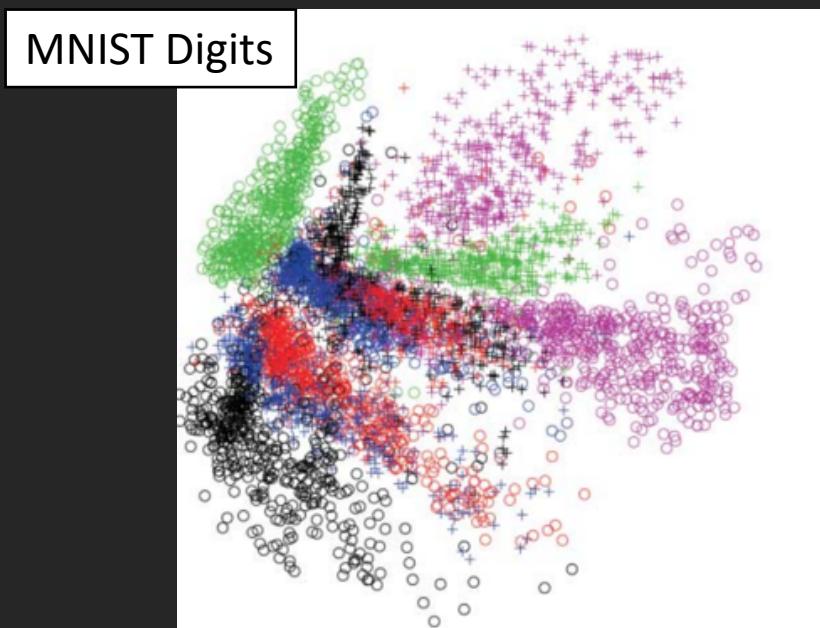
Autoencoder is originally proposed for dimensionality reduction and feature learning



Autoencoder

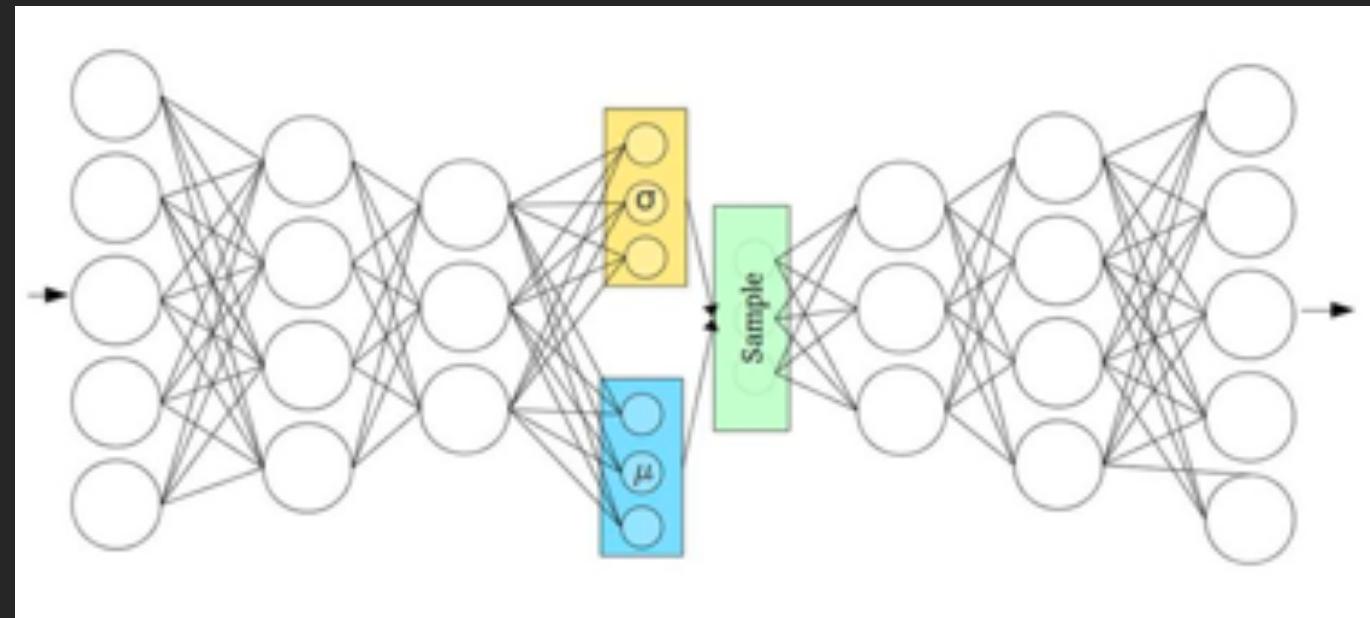
Autoencoder is originally proposed for dimensionality reduction and feature learning

Clustering using the latent features



Variational Autoencoder

- Problem with the vanilla autoencoder is that we cannot sample the latent code to generate new image
- Thus variational autoencoder is introduced



<https://github.com/pytorch/examples/blob/master/vae/main.py>

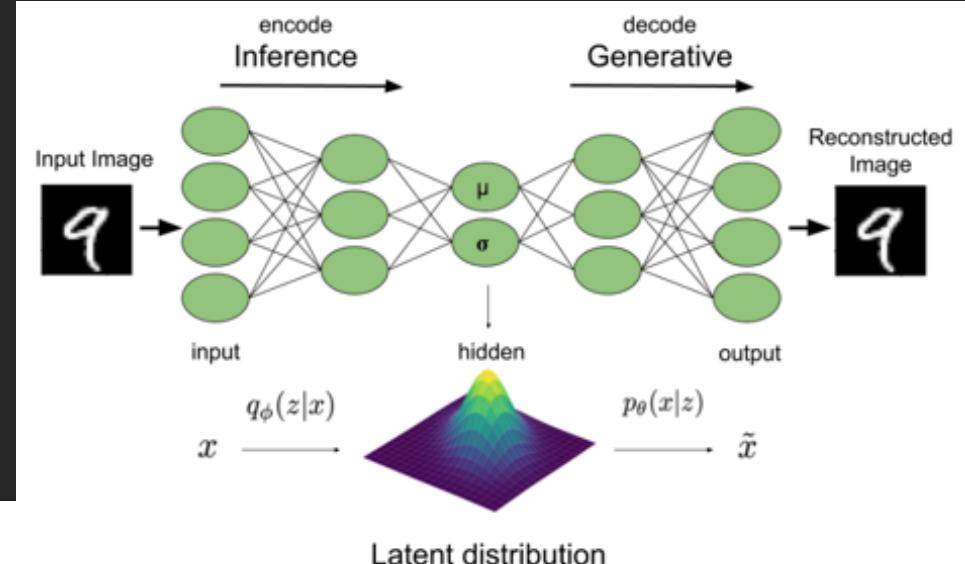
Variational Inference

Evidence Lower Bound (ELBO) holds for any q

$$\begin{aligned}\log p(\mathbf{x}; \theta) &\geq \sum_{\mathbf{z}} q(\mathbf{z}) \log \left(\frac{p_{\theta}(\mathbf{x}, \mathbf{z})}{q(\mathbf{z})} \right) \\&= \underbrace{\sum_{\mathbf{z}} q(\mathbf{z}) \log p_{\theta}(\mathbf{x}, \mathbf{z})}_{\text{Entropy } H(q) \text{ of } q} - \sum_{\mathbf{z}} q(\mathbf{z}) \log q(\mathbf{z}) \\&= \sum_{\mathbf{z}} q(\mathbf{z}) \log p_{\theta}(\mathbf{x}, \mathbf{z}) + H(q) \\&= \mathcal{L}(\mathbf{x}; \theta, \phi) + D_{KL}(q(\mathbf{z}; \phi) \| p(\mathbf{z}|\mathbf{x}; \theta))\end{aligned}$$

The better $q(\mathbf{z}; \phi)$ can approximate the posterior $p(\mathbf{z}|\mathbf{x}; \theta)$, the smaller $D_{KL}(q(\mathbf{z}; \phi) \| p(\mathbf{z}|\mathbf{x}; \theta))$ we can achieve, the closer ELBO will be to $\log p(\mathbf{x}; \theta)$. Next: jointly optimize over θ and ϕ to maximize the ELBO over a dataset

Variational Inference



$$\begin{aligned}\mathcal{L}(\mathbf{x}; \theta, \phi) &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}, \mathbf{x}; \theta) - \log p(\mathbf{z}) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z}|\mathbf{x})] \\ &= E_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z}; \theta)] - D_{KL}(q_\phi(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}))\end{aligned}$$

- ① Take a data point \mathbf{x}^i
- ② Map it to $\hat{\mathbf{z}}$ by sampling from $q_\phi(\mathbf{z}|\mathbf{x}^i)$ (*encoder*)
- ③ Reconstruct $\hat{\mathbf{x}}$ by sampling from $p(\mathbf{x}|\hat{\mathbf{z}}; \theta)$ (*decoder*)

What does the training objective $\mathcal{L}(\mathbf{x}; \theta, \phi)$ do?

- First term encourages $\hat{\mathbf{x}} \approx \mathbf{x}^i$ (\mathbf{x}^i likely under $p(\mathbf{x}|\hat{\mathbf{z}}; \theta)$)
- Second term encourages $\hat{\mathbf{z}}$ to be likely under the prior $p(\mathbf{z})$

https://deepgenerativemodels.github.io/assets/slides/cs236_lecture5.pdf

https://deepgenerativemodels.github.io/assets/slides/cs236_lecture6.pdf

Variational Autoencoder

Synthesis quality looked good, but not good for recent two-year standard



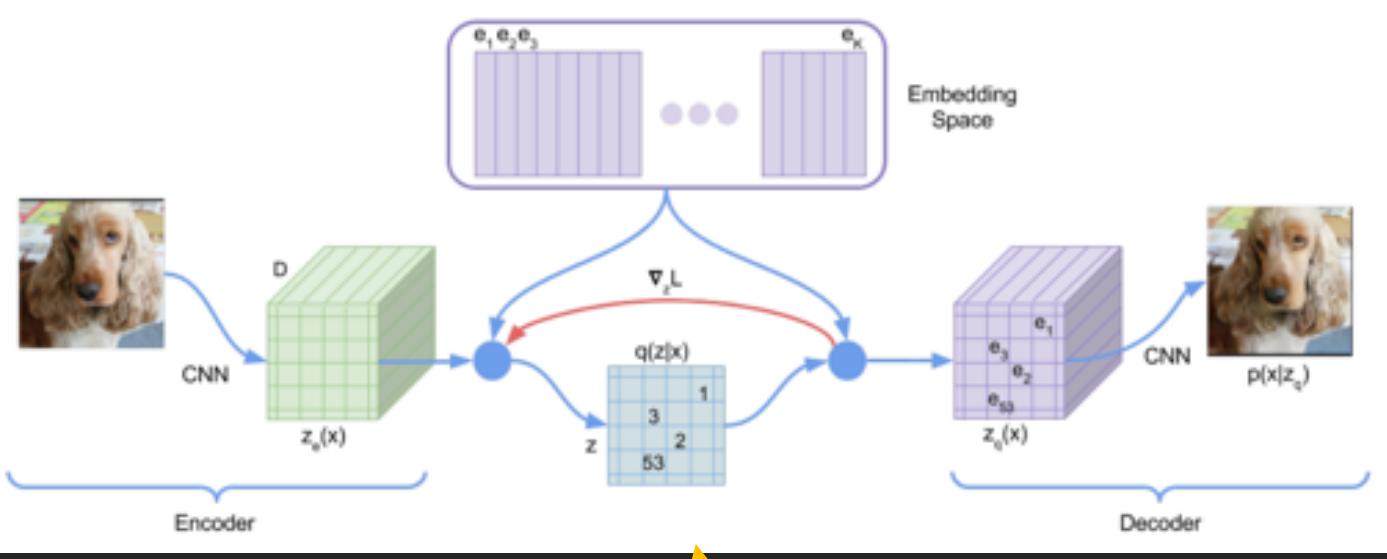
VQ-VAE and VQ-VAE-2

- Vector Quantized Variational AutoEncoder (VQ-VAE)
- Discrete latent representation + PixelCNN for handling posterior collapse



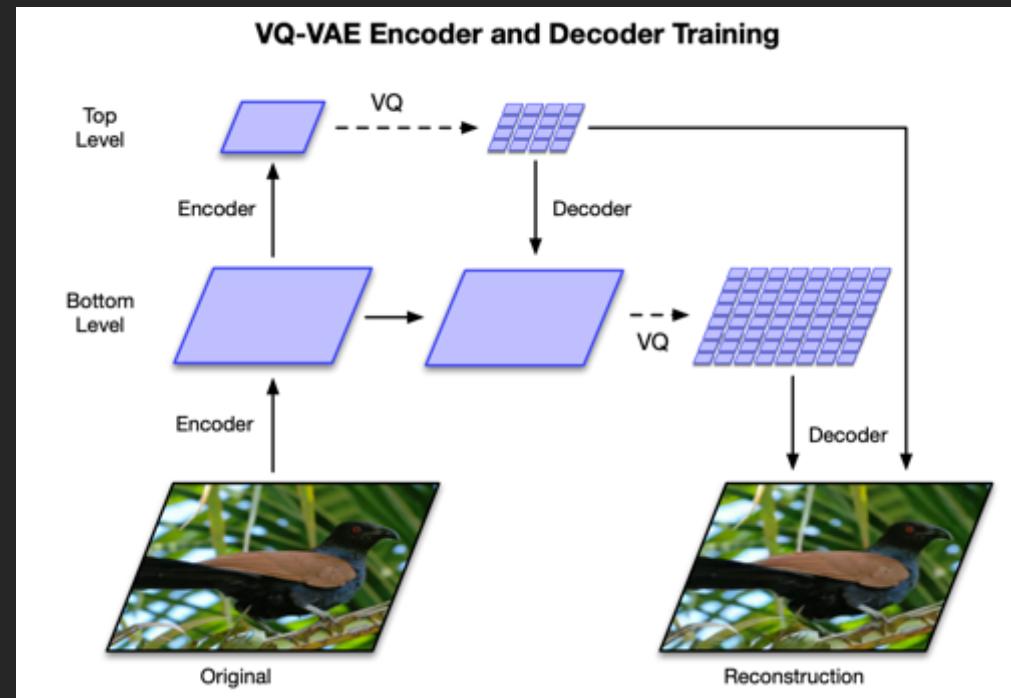
Figure 1: Class-conditional 256x256 image samples from a two-level model trained on ImageNet.

VQ-VAE-1



PixeCNN is learned on the discrete latent space

VQ-VAE-2



VQ-VAE and VQ-VAE-2

Very well handle mode-collapse issue in generative models

VQ-VAE on ImageNet's Tinca class

BigGAN



Summary of Latent Variable Models

Pros:

- Easy to build flexible models
- Suitable for unsupervised learning

Cons:

- Hard to evaluate likelihood
- Hard to train via maximum-likelihood, as it is difficult to compute the posterior inference $p(z|x)$ which requires variational approximations

Autoregressive models and Variational Autoencoders

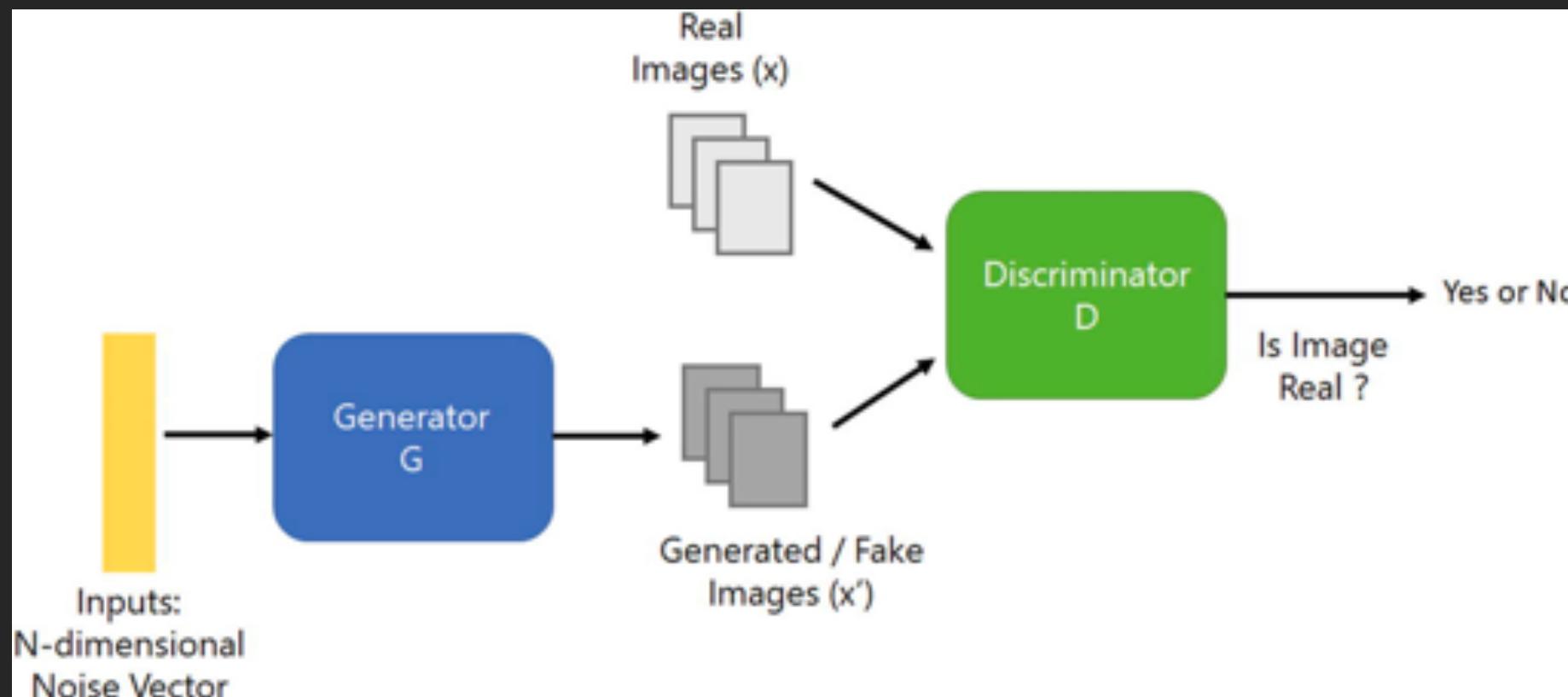
- Both model families are based on maximizing likelihoods (or approximation)

$$\begin{aligned} \text{Autoregressive Models: } p_{\theta}(\mathbf{x}) &= \prod_{i=1}^n p_{\theta}(x_i | \mathbf{x}_{<i}) \\ \text{Variational Autoencoders: } p_{\theta}(\mathbf{x}) &= \int p_{\theta}(\mathbf{x}, \mathbf{z}) d\mathbf{z} \end{aligned}$$

- Is that likelihood a good indicator? Any other likelihood-free generative modeling?
- Can we throw away KL-divergence and likelihood???

Generative Adversarial Networks

A two-player minimax game between a generator and a discriminator



Training objective for discriminator:

$$\max_D V(G, D) = E_{\mathbf{x} \sim p_{\text{data}}} [\log D(\mathbf{x})] + E_{\mathbf{x} \sim p_G} [\log(1 - D(\mathbf{x}))]$$

For a fixed generator G , the discriminator is performing binary classification with the cross entropy objective

- Assign probability 1 to true data points $\mathbf{x} \sim p_{\text{data}}$
- Assign probability 0 to fake samples $\mathbf{x} \sim p_G$

Optimal discriminator

$$D_G^*(\mathbf{x}) = \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})}$$

Training objective for generator:

$$\min_G V(G, D) = E_{\mathbf{x} \sim p_{\text{data}}} [\log D(\mathbf{x})] + E_{\mathbf{x} \sim p_G} [\log(1 - D(\mathbf{x}))]$$

For the optimal discriminator $D_G^*(\cdot)$, we have

$$\begin{aligned} & V(G, D_G^*(\mathbf{x})) \\ &= E_{\mathbf{x} \sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})} \right] + E_{\mathbf{x} \sim p_G} \left[\log \frac{p_G(\mathbf{x})}{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})} \right] \\ &= E_{\mathbf{x} \sim p_{\text{data}}} \left[\log \frac{p_{\text{data}}(\mathbf{x})}{\frac{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})}{2}} \right] + E_{\mathbf{x} \sim p_G} \left[\log \frac{p_G(\mathbf{x})}{\frac{p_{\text{data}}(\mathbf{x}) + p_G(\mathbf{x})}{2}} \right] - \log 4 \\ &= \underbrace{D_{KL} \left[p_{\text{data}}, \frac{p_{\text{data}} + p_G}{2} \right] + D_{KL} \left[p_G, \frac{p_{\text{data}} + p_G}{2} \right]}_{2 \times \text{Jenson-Shannon Divergence (JSD)}} - \log 4 \\ &= 2D_{JSD}[p_{\text{data}}, p_G] - \log 4 \quad p_G = p_{\text{data}} \end{aligned}$$

Symmetric KL divergence

Optimal generator

Joint loss:

$$\min_{\theta} \max_{\phi} V(G_{\theta}, D_{\phi}) = E_{\mathbf{x} \sim p_{\text{data}}} [\log D_{\phi}(\mathbf{x})] + E_{\mathbf{z} \sim p(\mathbf{z})} [\log(1 - D_{\phi}(G_{\theta}(\mathbf{z})))]$$

GAN training algorithm

Repeat:

- Sample minibatch of m training points $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \dots, \mathbf{x}^{(m)}$ from \mathcal{D}
- Sample minibatch of m noise vectors $\mathbf{z}^{(1)}, \mathbf{z}^{(2)}, \dots, \mathbf{z}^{(m)}$ from p_z
- Update the generator parameters θ by stochastic gradient **descent**

$$\nabla_{\theta} V(G_{\theta}, D_{\phi}) = \frac{1}{m} \nabla_{\theta} \sum_{i=1}^m \log(1 - D_{\phi}(G_{\theta}(\mathbf{z}^{(i)})))$$

- Update the discriminator parameters ϕ by stochastic gradient **ascent**

$$\nabla_{\phi} V(G_{\theta}, D_{\phi}) = \frac{1}{m} \nabla_{\phi} \sum_{i=1}^m [\log D_{\phi}(\mathbf{x}^{(i)}) + \log(1 - D_{\phi}(G_{\theta}(\mathbf{z}^{(i)})))]$$

Example code of training GAN

```
for i, data in enumerate(dataloader, 0):
    #####
    # (1) Update D network: maximize log(D(x)) + log(1 - D(G(z)))
    #####
    # train with real
    netD.zero_grad()
    real_cpu = data[0].to(device)
    batch_size = real_cpu.size(0)
    label = torch.full((batch_size,), real_label, device=device)

    output = netD(real_cpu)
    errD_real = criterion(output, label)
    errD_real.backward()
    D_x = output.mean().item()

    # train with fake
    noise = torch.randn(batch_size, nz, 1, 1, device=device)
    fake = netG(noise)
    label.fill_(fake_label)
    output = netD(fake.detach())
    errD_fake = criterion(output, label)
    errD_fake.backward()
    D_G_z1 = output.mean().item()
    errD = errD_real + errD_fake
    optimizerD.step()

    #####
    # (2) Update G network: maximize log(D(G(z)))
    #####
    netG.zero_grad()
    label.fill_(real_label) # fake labels are real for generator cost
    output = netD(fake)
    errG = criterion(output, label)
    errG.backward()
    D_G_z2 = output.mean().item()
    optimizerG.step()
```

GenForce: Generative Modeling for Everyone

- <https://github.com/genforce/genforce>
- Open-source PyTorch library for generative modeling
- 60+ pretrained GAN models with Colab live demo



https://colab.research.google.com/github/genforce/genforce/blob/master/docs/synthesize_demo.ipynb



Session 1 ends here

- Try the Google Colab for BigGAN:

https://colab.research.google.com/github/tensorflow/hub/blob/master/examples/colab/biggan_generation_with_tf_hub.ipynb

- Try the Google Colab for StyleGAN from GenForce:

https://colab.research.google.com/github/genforce/genforce/blob/master/docs/synthesize_demo.ipynb