10-423/623: Generative AI Lecture 6 – Generative Adversarial Networks and Variational Autoencoders

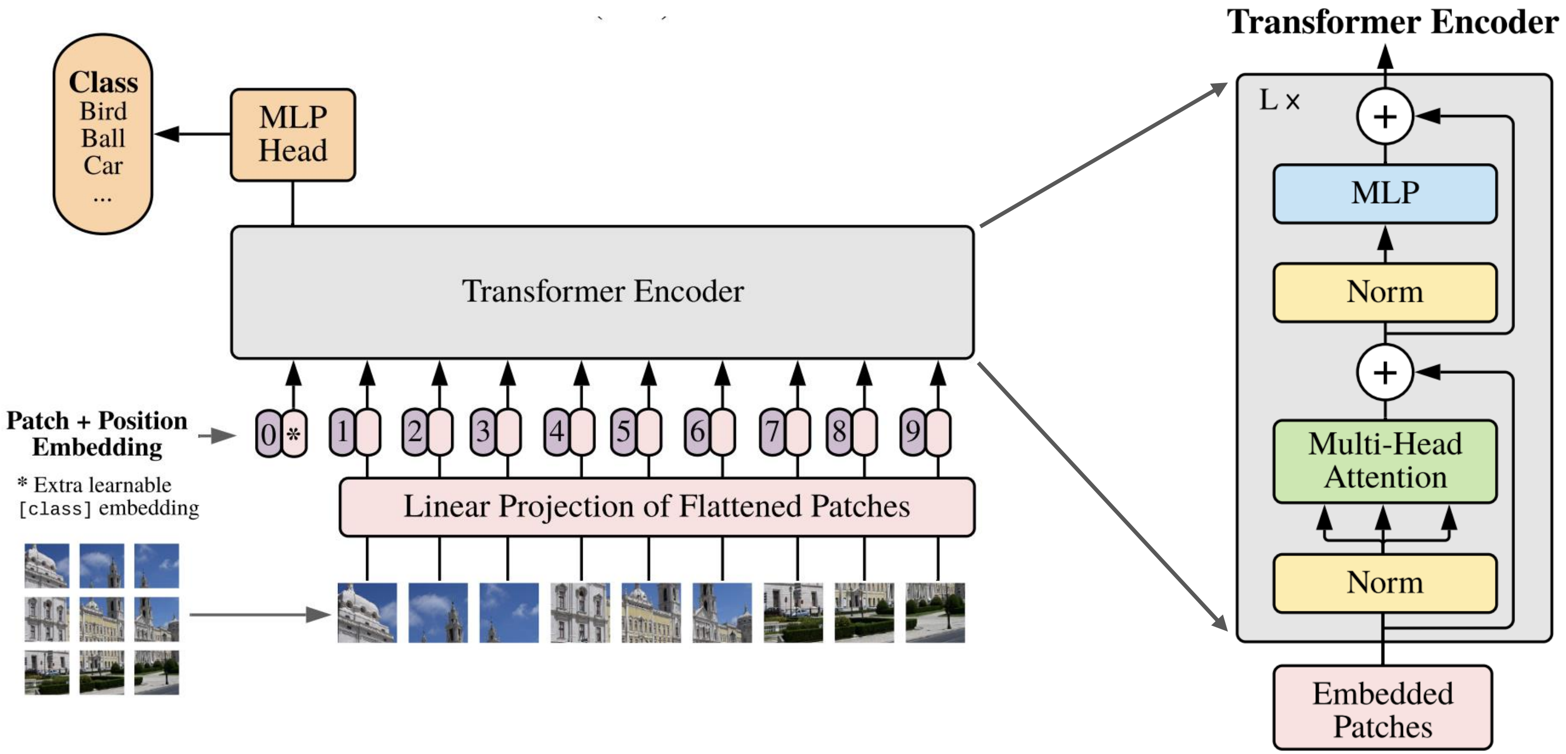
Henry Chai & Matt Gormley

9/16/24

Front Matter

- Announcements:
  - HW1 released 9/9, due 9/23 at 11:59 PM

Recall: Vision Transformer (ViT)

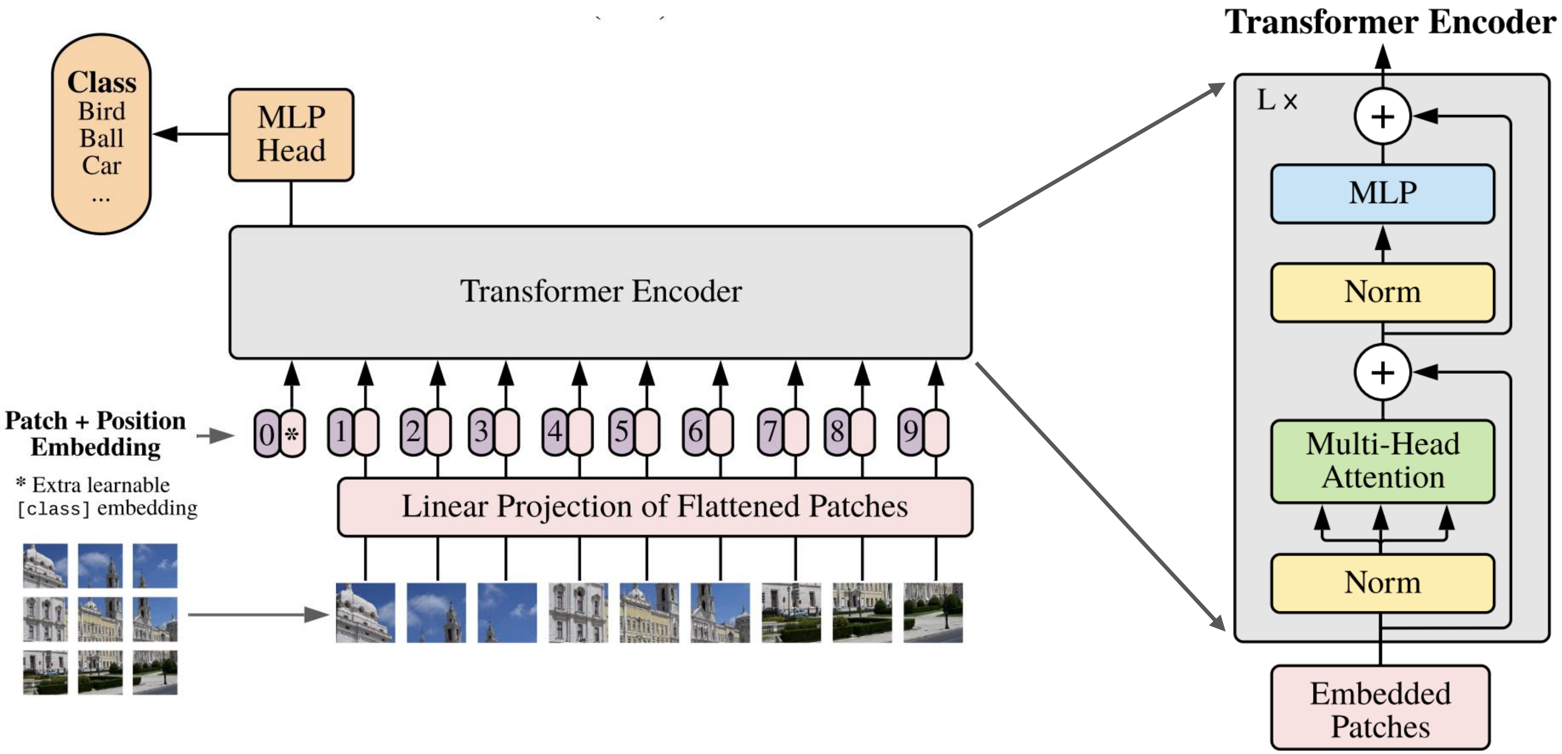


- Instead of words as input, the inputs are $P \times P$ pixel *patches*
- Each patch is embedded linearly into a vector of size 1024
- Uses 1D positional embeddings
- Pre-trained on a large, supervised dataset (e.g., ImageNet 21K, JFT-300M)

Source: https://arxiv.org/pdf/2010.11929

Is this even a generative model?

Not inherently…



- Instead of words as input, the inputs are $P \times P$ pixel *patches*
- Each patch is embedded linearly into a vector of size 1024
- Uses 1D positional embeddings
- Pre-trained on a large, supervised dataset (e.g., ImageNet 21K, JFT-300M)

Source: https://arxiv.org/pdf/2010.11929

Common Tasks in Computer Vision

- Image Classification
- Object Localization
- Object Detection
- Semantic Segmentation
- Instance Segmentation
- Image Captioning
- **Image Generation**

- Class-conditional generation
- Super resolution
- Image Editing
- Style transfer
- Text-to-image (TTI) generation

Image Generation

sea anemone

brain coral

slug

- Given a class label, sample a new image from that class
  - Image classification takes an image and predicts its label using $p(y|\boldsymbol{x})$
  - Class-conditional generation does this in reverse with $p(\boldsymbol{x}|y)$

- **Class-conditional generation**
- Super resolution
- Image Editing
- Style transfer
- Text-to-image (TTI) generation

Source: https://arxiv.org/pdf/1906.00446.pdf

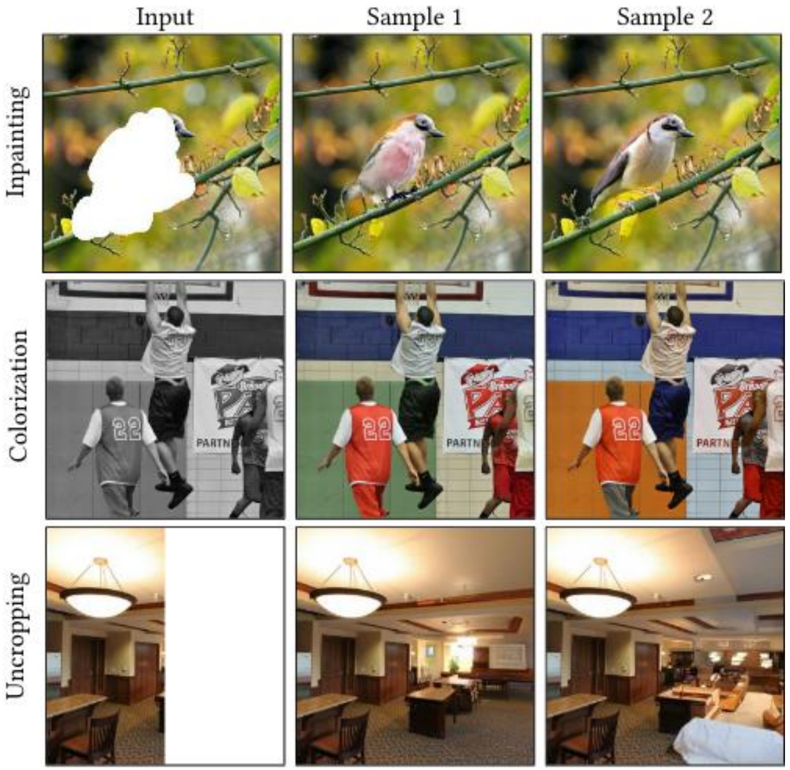
Image Generation

- Given a low-resolution image, generate a high-resolution reconstruction of the image

- Class-conditional generation
- **Super resolution**
- Image Editing
- Style transfer
- Text-to-image (TTI) generation

Source: https://arxiv.org/pdf/2104.14951.pdf

Image Generation

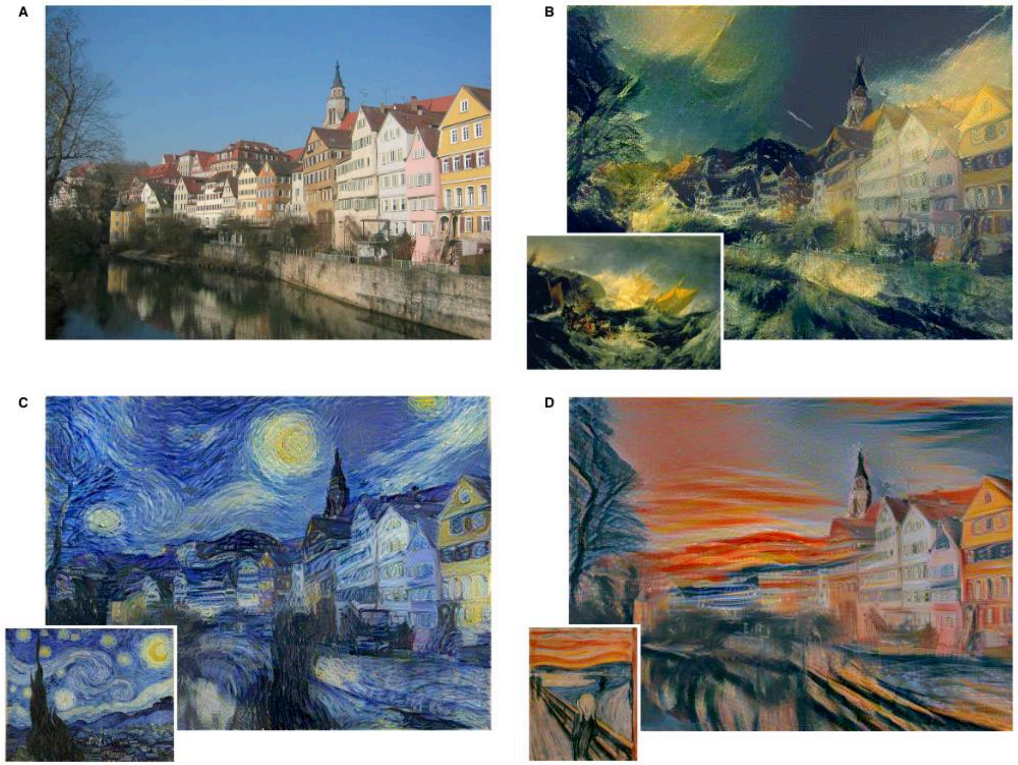


- Class-conditional generation
- Super resolution
- **Image Editing**

- **Inpainting** fills in the (pre-specified) missing pixels
- **Colorization** restores color to a greyscale image
- **Uncropping** creates a photo-realistic reconstruction of a missing side of an image

Source: https://arxiv.org/pdf/2111.05826.pdf

Image Generation



- Given two images, present the semantic content of the *source* image in the style of the *reference* image

- Class-conditional generation
- Super resolution
- Image Editing
- **Style transfer**
- Text-to-image (TTI) generation

Source: https://arxiv.org/pdf/1508.06576.pdf

Image Generation

Prompt: A propaganda poster depicting a cat dressed as french emperor napoleon holding a piece of cheese.

- Given a text description, sample an image that depicts the prompt

- Class-conditional generation
- Super resolution
- Image Editing
- Style transfer
- **Text-to-image (TTI) generation**

Source: https://arxiv.org/pdf/2307.01952.pdf

Image Generation

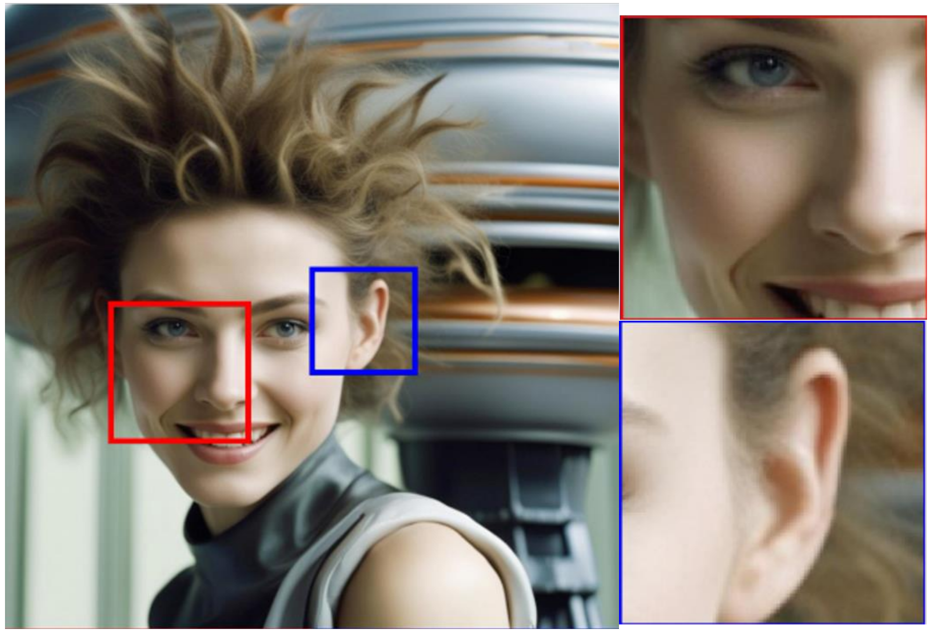
Prompt: Epic long distance cityscape photo of New York City flooded by the ocean and overgrown buildings and jungle ruins in rainforest, at sunset, cinematic shot, highly detailed, 8k, golden light

- Class-conditional generation
- Super resolution
- Image Editing
- Style transfer
- **Text-to-image (TTI) generation**

Source: https://arxiv.org/pdf/2307.01952.pdf

Image Generation

Prompt: close up headshot, futuristic young woman, wild hair sly smile in front of gigantic UFO, dslr, sharp focus, dynamic composition



- Class-conditional generation
- Super resolution
- Image Editing
- Style transfer
- **Text-to-image (TTI) generation**

Source: https://arxiv.org/pdf/2307.01952.pdf

Slide Generation?

Prompt: powerpoint slide explaining generative adversarial networks for a generative AI course, easy to follow, with a clear explanation of the objective function

- Class-conditional generation
- Super resolution
- Image Editing
- Style transfer
- **Text-to-image (TTI) generation**

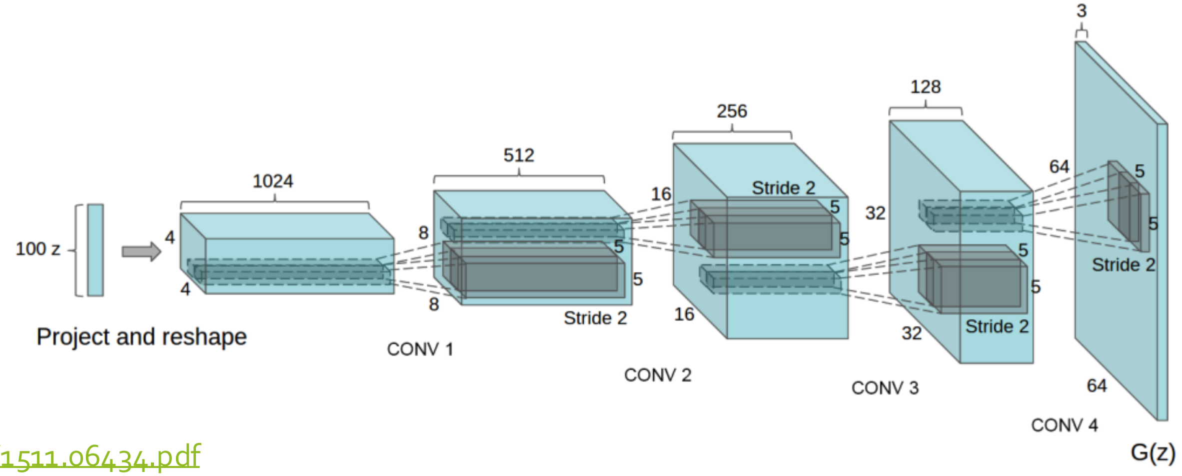
Source: https://stablediffusionweb.com/app/image-generator

Generative Adversarial Networks (GANs)

- A GAN consists of two (deterministic) models:
  - a **generator** that takes a vector of random noise as input, and generates an image
  - a **discriminator** that takes in an image classifies whether it is real (label = 1) or fake (label = 0)
  - Both models are typically (but not necessarily) neural networks

Generative Adversarial Networks (GANs)

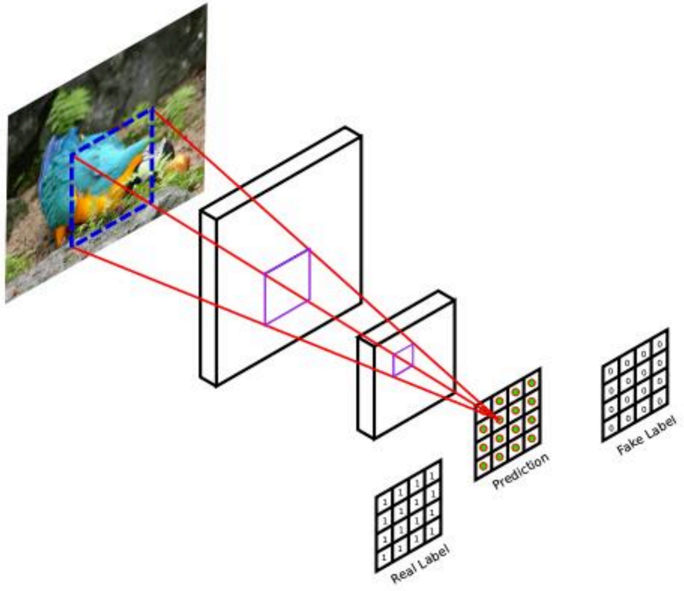
- A GAN consists of two (deterministic) models:
  - a **generator** that takes a vector of random noise as input, and generates an image
- Example generator: DCGAN
  - An inverted CNN with four *fractionally-strided* convolution layers that grow the size of the image from layer to layer; final layer has three channels to generate color images



Source: https://arxiv.org/pdf/1511.06434.pdf

Generative Adversarial Networks (GANs)

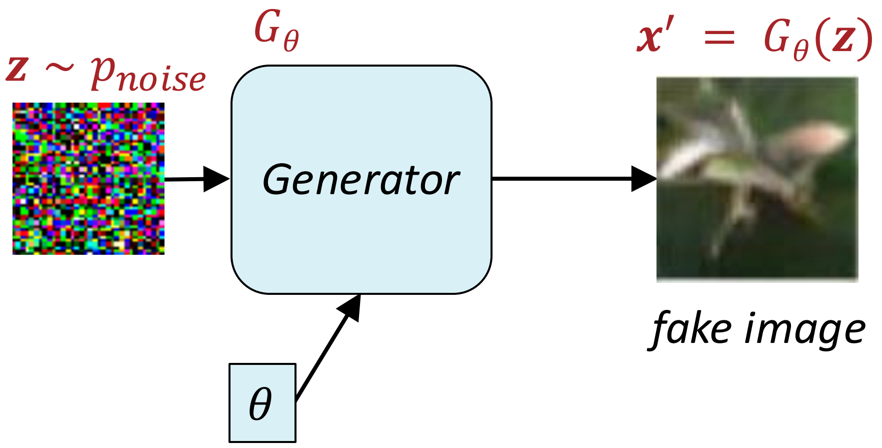
- A GAN consists of two (deterministic) models:
  - a **generator** that takes a vector of random noise as input, and generates an image
  - a **discriminator** that takes in an image classifies whether it is real (label = 1) or fake (label = 0)
- Example discriminator: PatchGAN
  - Traditional CNN that looks at each patch of the image and tries to predict whether it is real or fake; can help encourage to generator to avoid creating blurry images



Source: https://arxiv.org/pdf/1803.07422.pdf

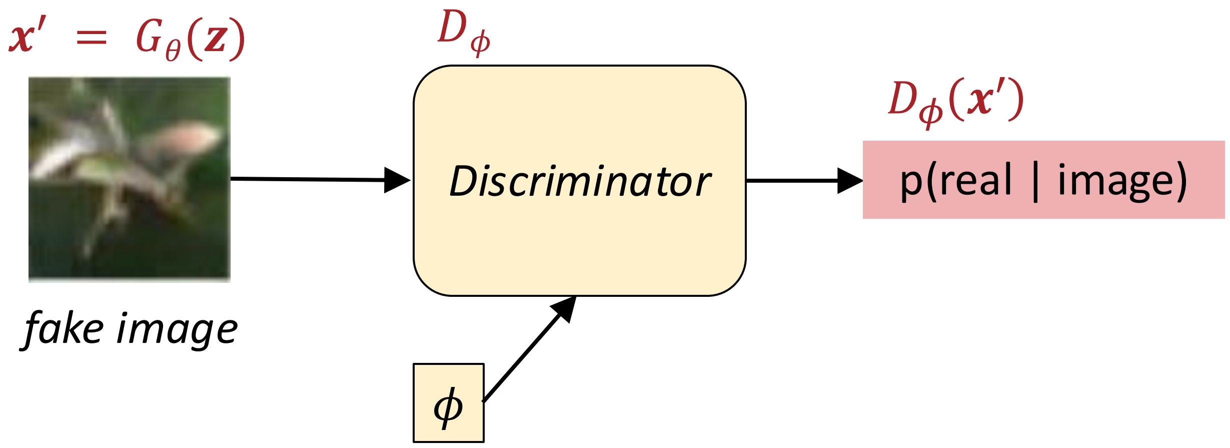
Generative Adversarial Networks (GANs): Training

- A GAN consists of two (deterministic) models:
  - a **generator** that takes a vector of random noise as input, and generates an image
  - a **discriminator** that takes in an image classifies whether it is real (label = 1) or fake (label = 0)
  - Both models are typically (but not necessarily) neural networks
- During training, the GAN plays a two-player minimax game: the generator tries to create realistic images to fool the discriminator and the discriminator tries to identify the real images from the fake ones

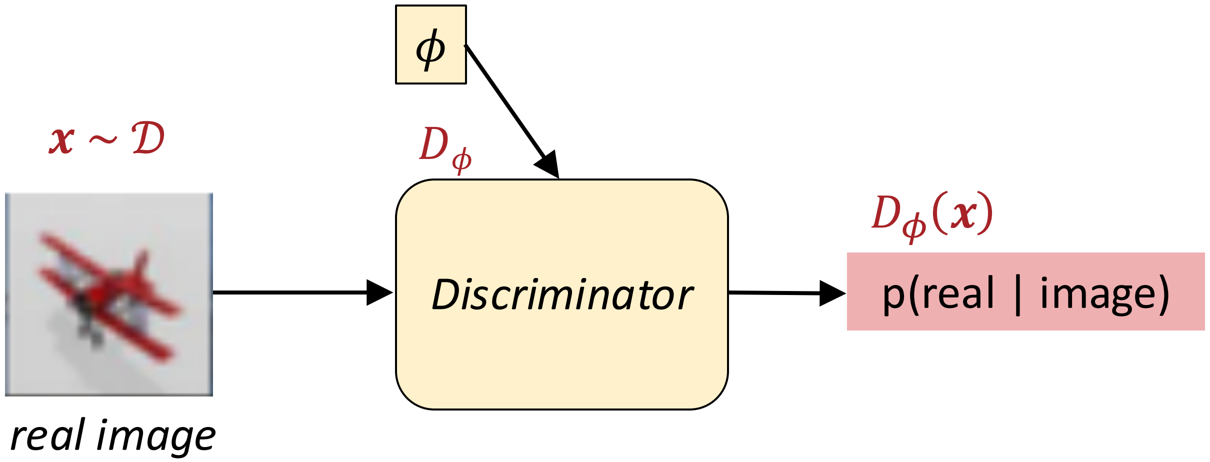


Typically, p_{noise} is a standard Gaussian i.e., $\mathcal{N}(\mathbf{0}, \sigma^2 I)$

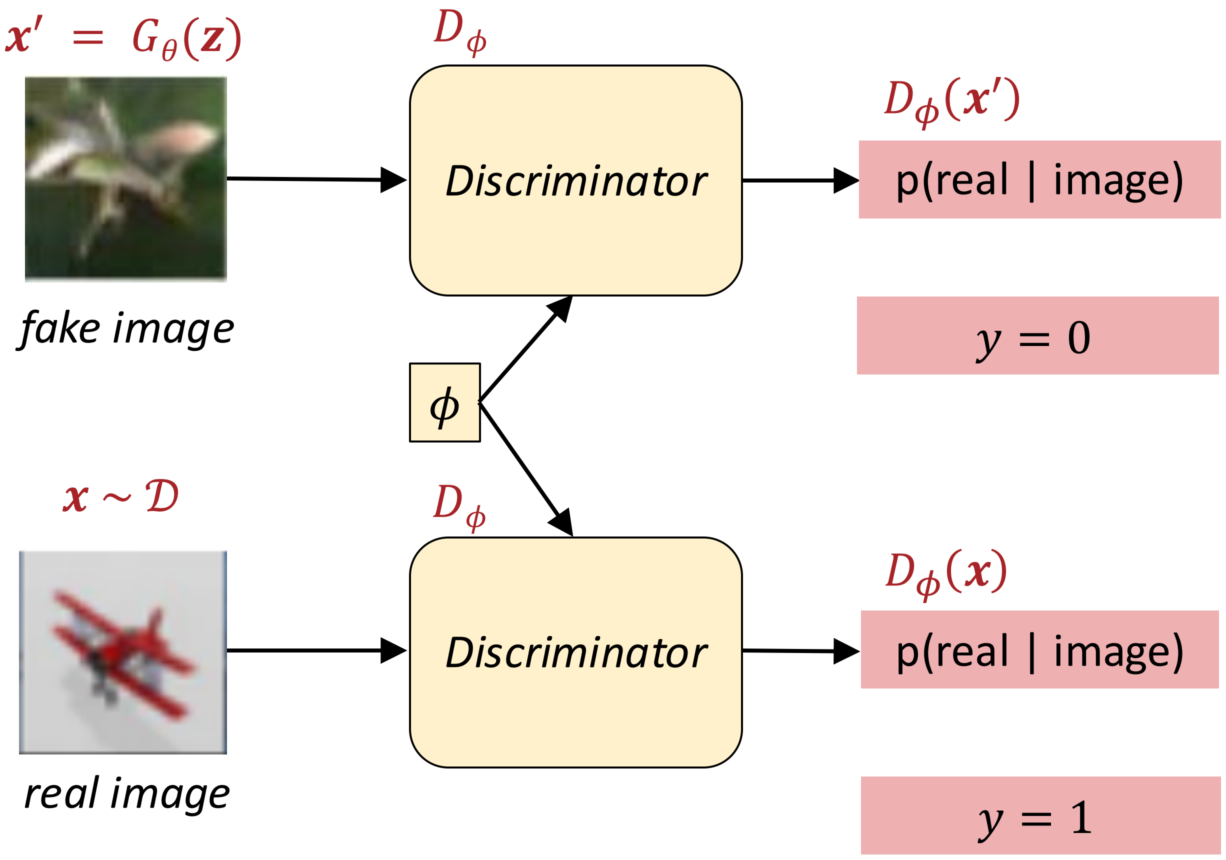
GANs: Architecture



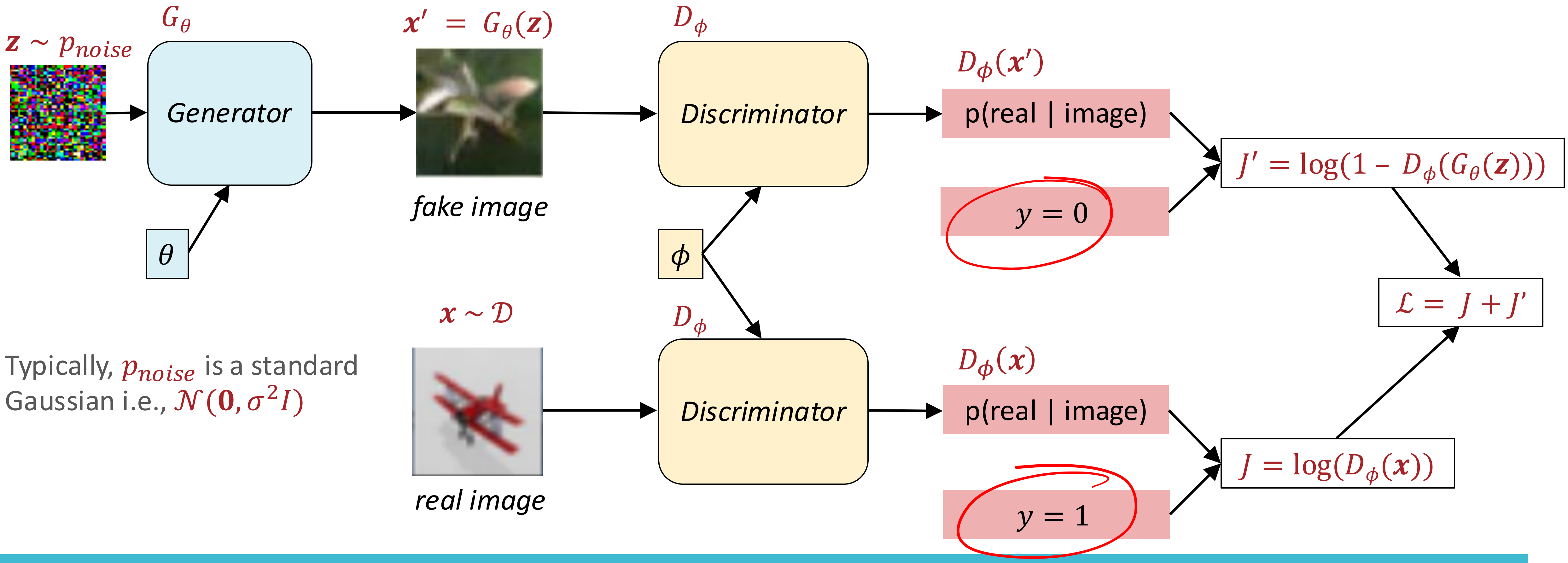
GANs: Architecture



GANs: Architecture

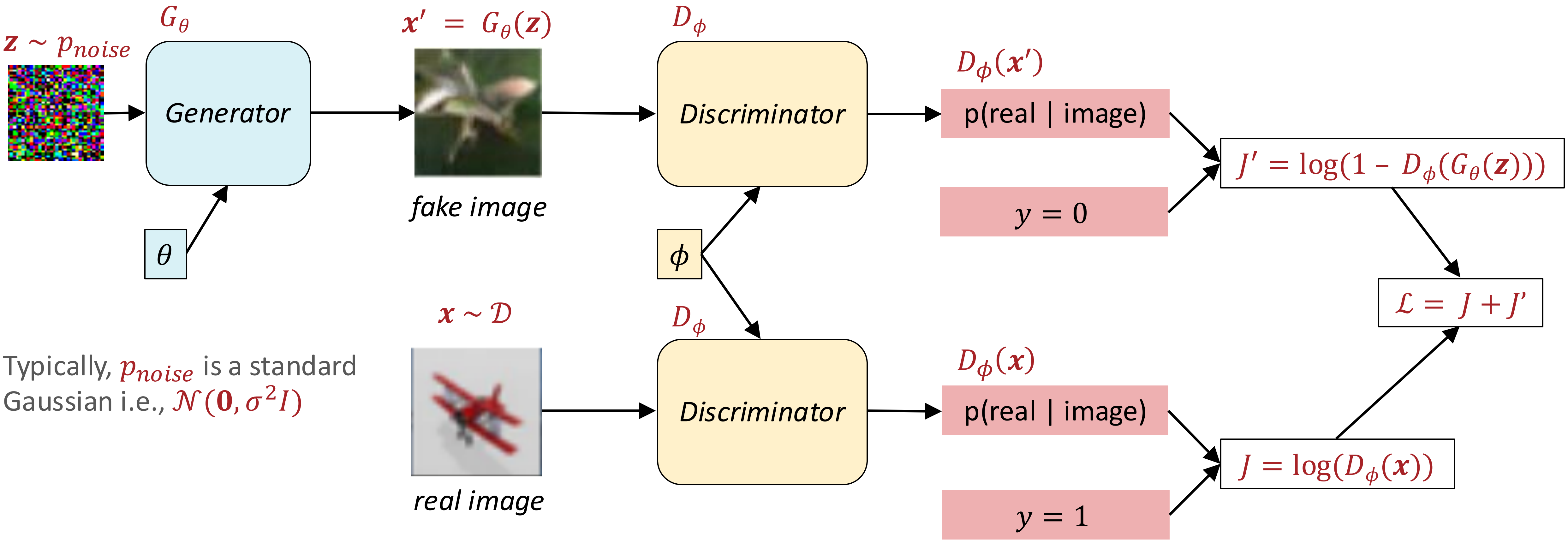


GANs: Architecture



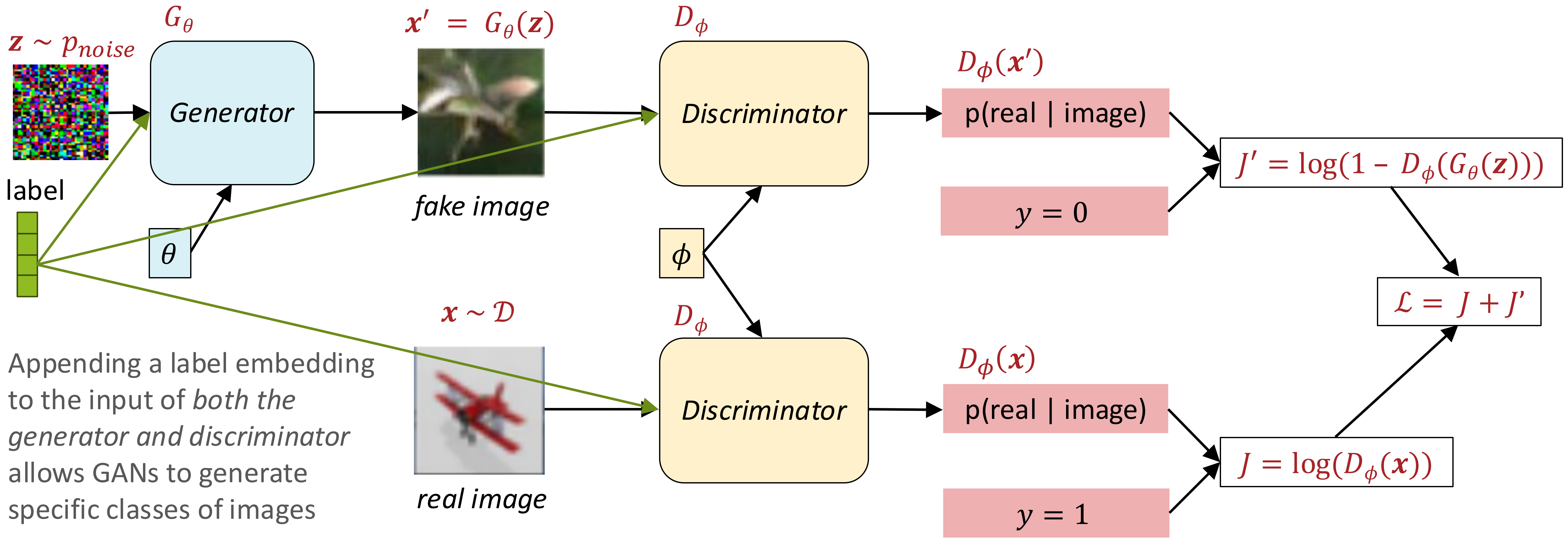
Typically, p_{noise} is a standard Gaussian i.e., $\mathcal{N}(\mathbf{0}, \sigma^2 I)$

GANs: Architecture



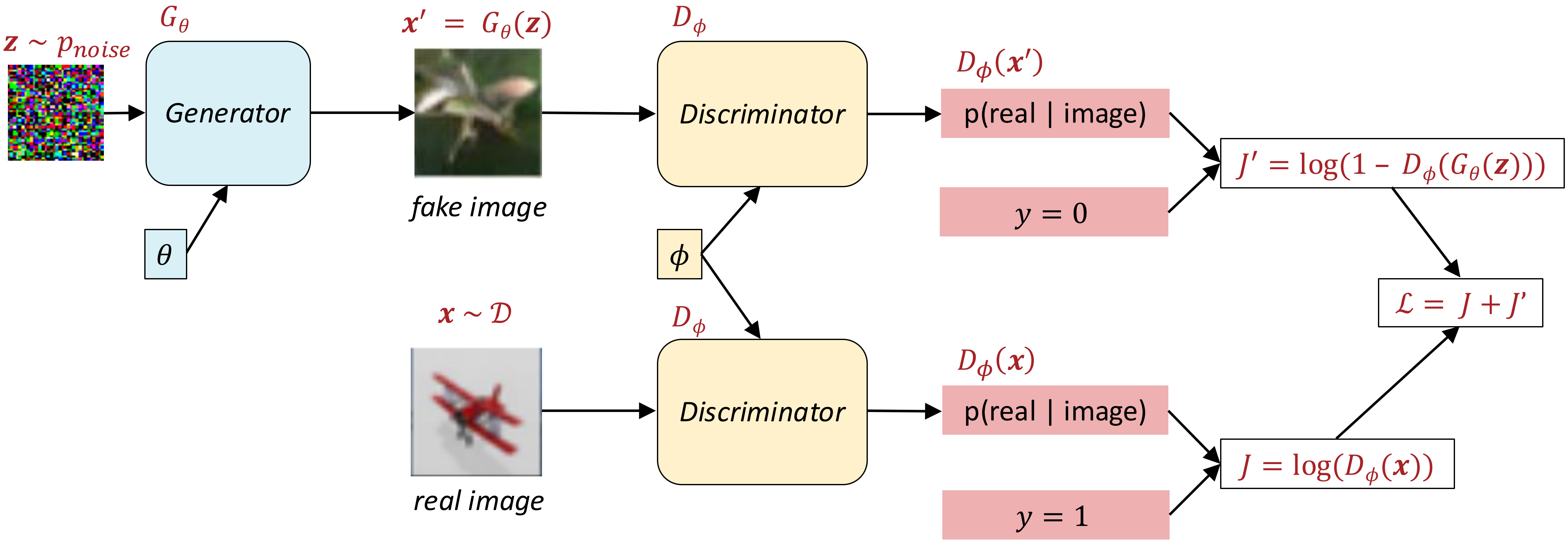
Typically, p_{noise} is a standard Gaussian i.e., $\mathcal{N}(\mathbf{0}, \sigma^2 I)$

Can we backpropagate through G_θ given that $\boldsymbol{z}$ is stochastic?



Appending a label embedding to the input of *both the generator and discriminator* allows GANs to generate specific classes of images

Class-conditional GANs



So how do we go about training one of these things?

The discriminator is trying to maximize the likelihood of the true labels {real = 1, fake = 0} for a fixed generator

$$\max_{\phi} \log\left(D_{\phi}(\mathbf{x}^{(i)})\right) + \log\left(1 - D_{\phi}(G_{\theta}(\mathbf{z}^{(i)}))\right)$$

$$\min_{\theta} \log\left(1 - D_{\phi}(G_{\theta}(\mathbf{z}^{(i)}))\right)$$

The generator is trying to minimize the likelihood of its generated (fake) image being classified as fake, according to a fixed discriminator

$J' = \log(1 - D_{\phi}(G_{\theta}(\boldsymbol{z})))$

$\mathcal{L} = J + J'$

$J = \log(D_{\phi}(\boldsymbol{x}))$

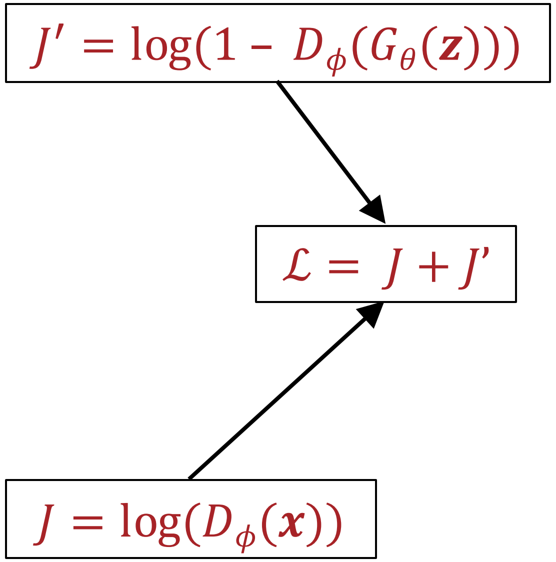
GANs: Training

GANs: Training

Both objectives (and hence, their sum) are differentiable!

$$\max_{\phi} \log\left(D_\phi(\mathbf{x}^{(i)})\right) + \log\left(1 - D_\phi(G_\theta(\mathbf{z}^{(i)}))\right)$$

$$\min_{\theta} \log\left(1 - D_\phi(G_\theta(\mathbf{z}^{(i)}))\right)$$



Training alternates between:

1. Keeping θ fixed and backpropagating through D_ϕ
2. Keeping ϕ fixed and backpropagating through G_θ

GANs: Training

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**

  for k steps **do**

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for

  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

- Optimization is like block coordinate descent but instead of exact optimization, we take a step of mini-batch SGD

Source: https://arxiv.org/pdf/1406.2661.pdf

But what about those Vision Transformer things we talked about last week?

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

for number of training iterations **do**
  for k steps **do**
    • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for
The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

- Optimization is like block coordinate descent but instead of exact optimization, we take a step of mini-batch SGD

Source: https://arxiv.org/pdf/1406.2661.pdf

TransGANs

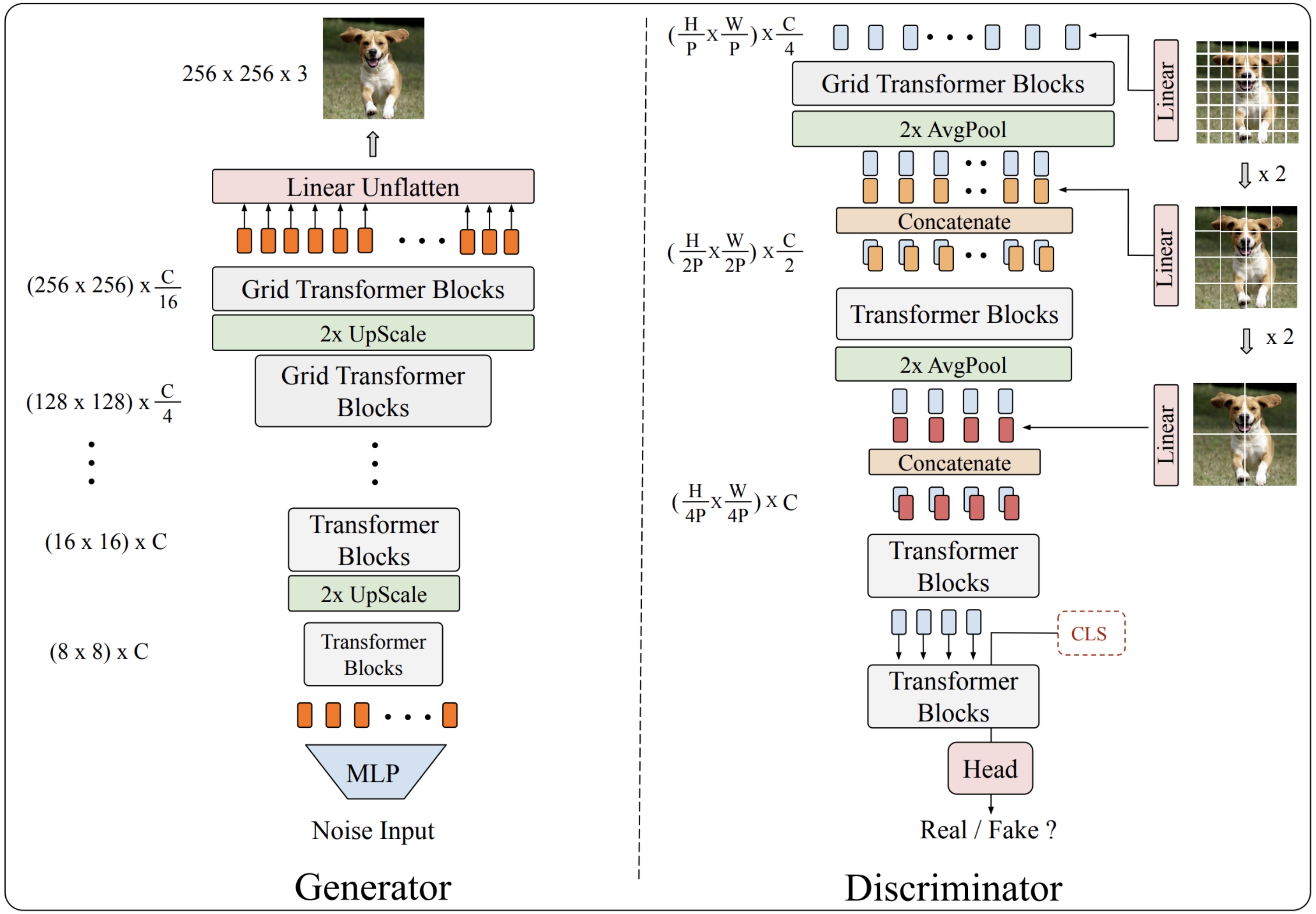


Figure 2: The pipeline of the pure transform-based generator and discriminator of TransGAN.

Source: https://arxiv.org/pdf/2102.07074

TransGANs

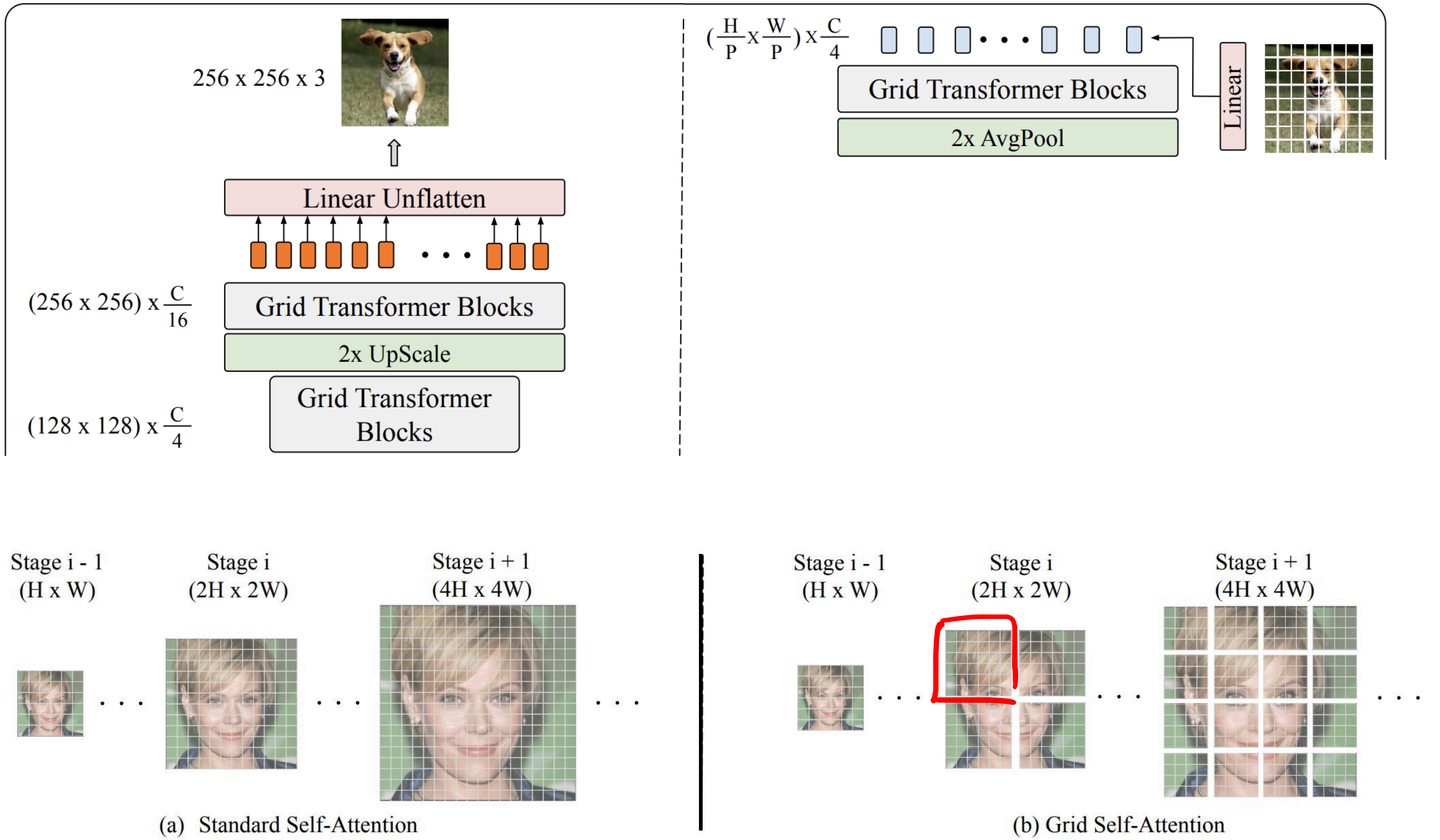


Figure 3: Grid Self-Attention across different transformer stages. We replace Standard Self-Attention with Grid Self-Attention when the resolution is higher than 32×32 and the grid size is set to be 16×16 by default.

Source: https://arxiv.org/pdf/2102.07074

ViTGANs

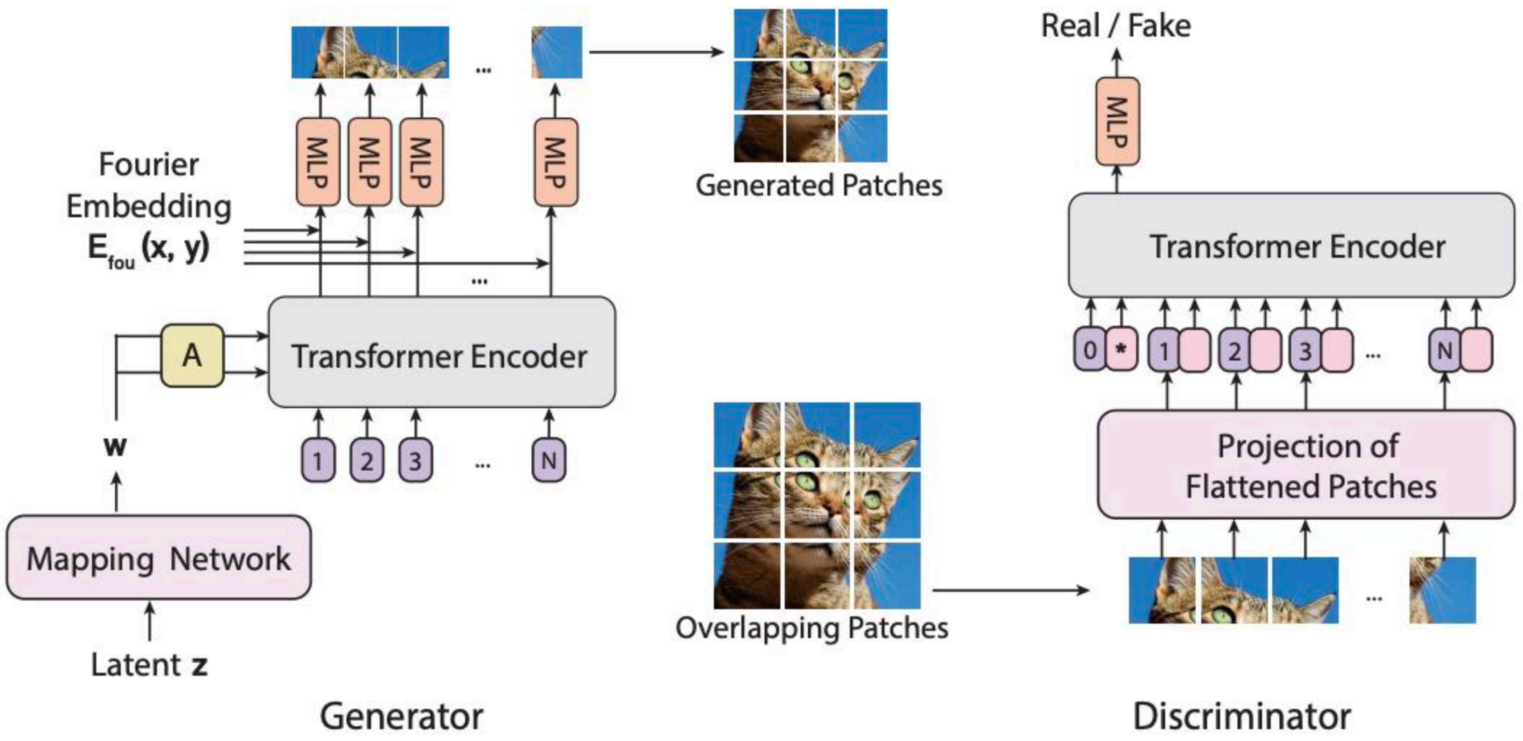
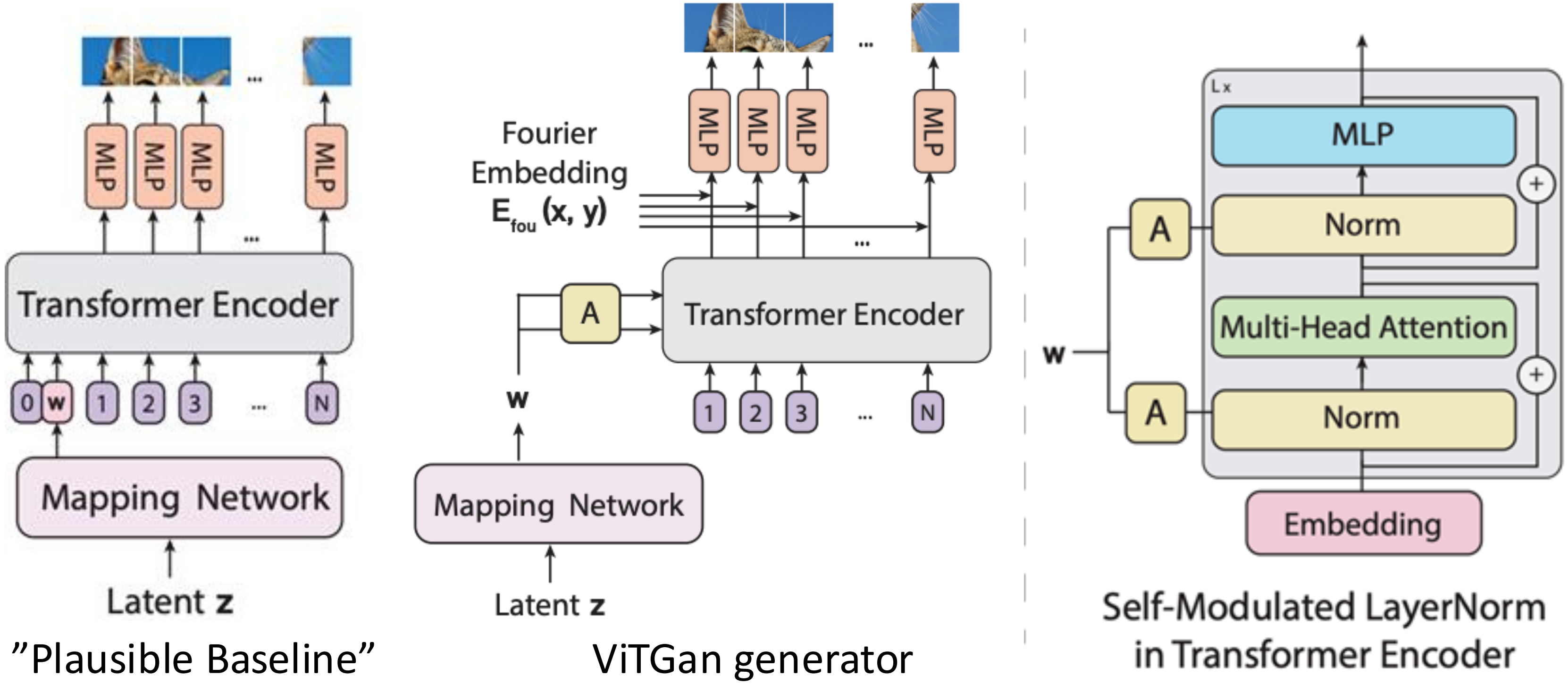


Figure 1: **Overview of the proposed ViTGAN framework.** Both the generator and the discriminator are designed based on the Vision Transformer (ViT). Discriminator score is derived from the classification embedding (denoted as [*] in the Figure). The generator generates pixels patch-by-patch based on patch embeddings.

Source: https://arxiv.org/pdf/2107.04589

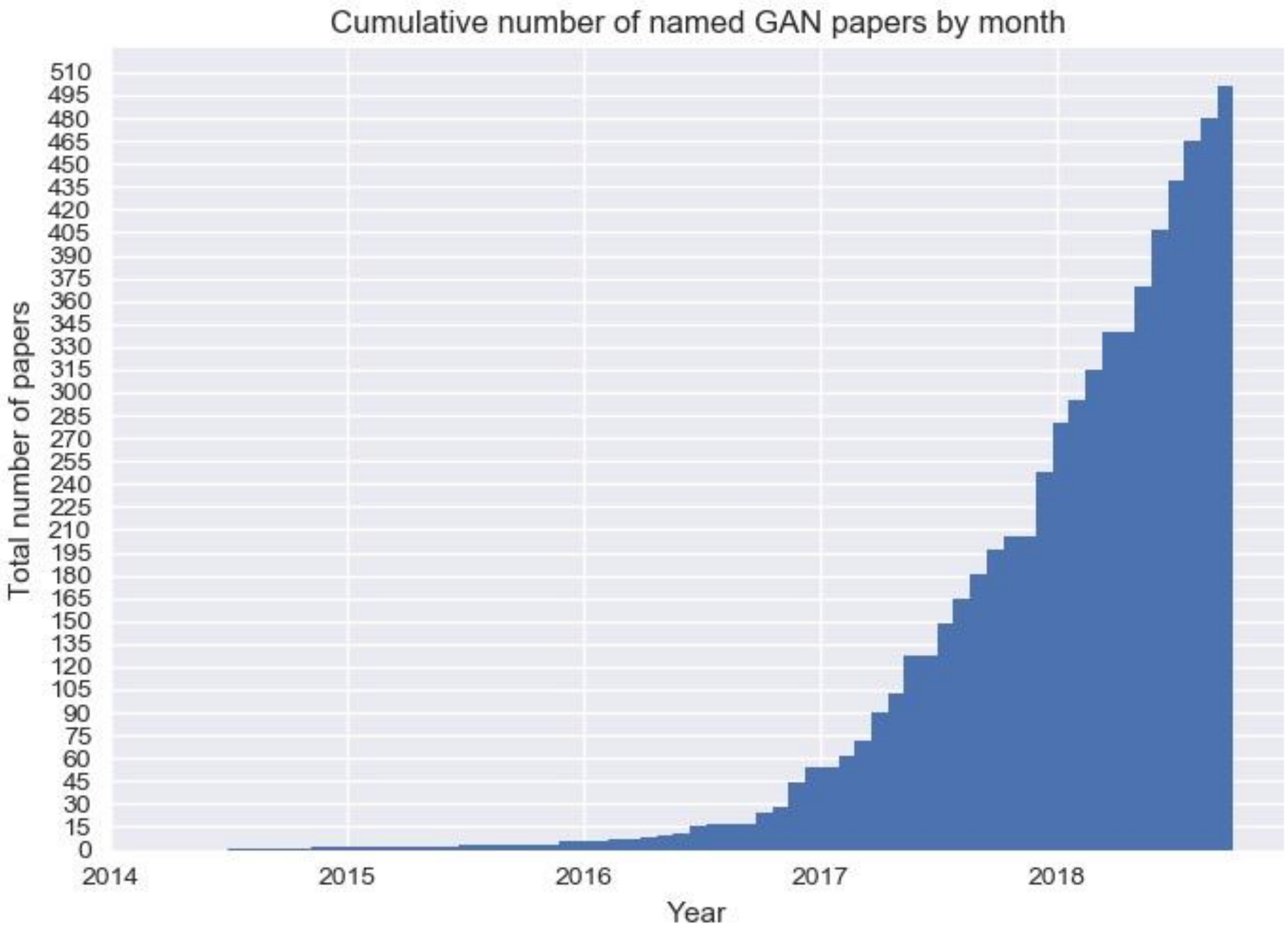
ViTGANs



"Plausible Baseline" ViTGan generator

Source: https://arxiv.org/pdf/2107.04589

GANs Everywhere!



Source: https://github.com/hindupuravinash/the-gan-zoo/tree/master

Recall: Computer Vision Timeline

- **1998**: LeNet
- **2009**: ImageNet
- **2012**: AlexNet
- **2013**: VAEs
- **2014**: VGG, R-CNN, **GANs**
- **2015**: Diffusion models, ResNet
- **2017**: Transformer
- **2020**: DDPM
- **2021**: ViT, Dall-E, CLIP
- **2022**: Dall-E 2, Imagen, Stable diffusion
- **2023**: SDXL, SDXL Turbo

Recall: Computer Vision Timeline

- **1998** — LeNet
- **2009** — ImageNet
- **2012** — AlexNet
- **2013** — VAEs
- **2014** — VGG, R-CNN, GANs
- **2015** — Diffusion models, ResNet
- **2017** — Transformer
- **2020** — **DDPM**
- **2021** — ViT, Dall-E, CLIP
- **2022** — Dall-E 2, Imagen, Stable diffusion
- **2023** — SDXL, SDXL Turbo

GANs vs. Diffusion

GAN generated images | Training images | Diffusion generated images

Source: https://medium.com/thedeephub/what-is-the-gan-generative-adversarial-networks-2ed6965c13fb

Recall: Computer Vision Timeline

1998
LeNet

2009
ImageNet

2012
AlexNet

2013
1. VAEs

2014
VGG
R-CNN
GANs

2015
Diffusion models
ResNet

2017
Transformer

2020
2. DDPM

2021
ViT
Dall-E
CLIP

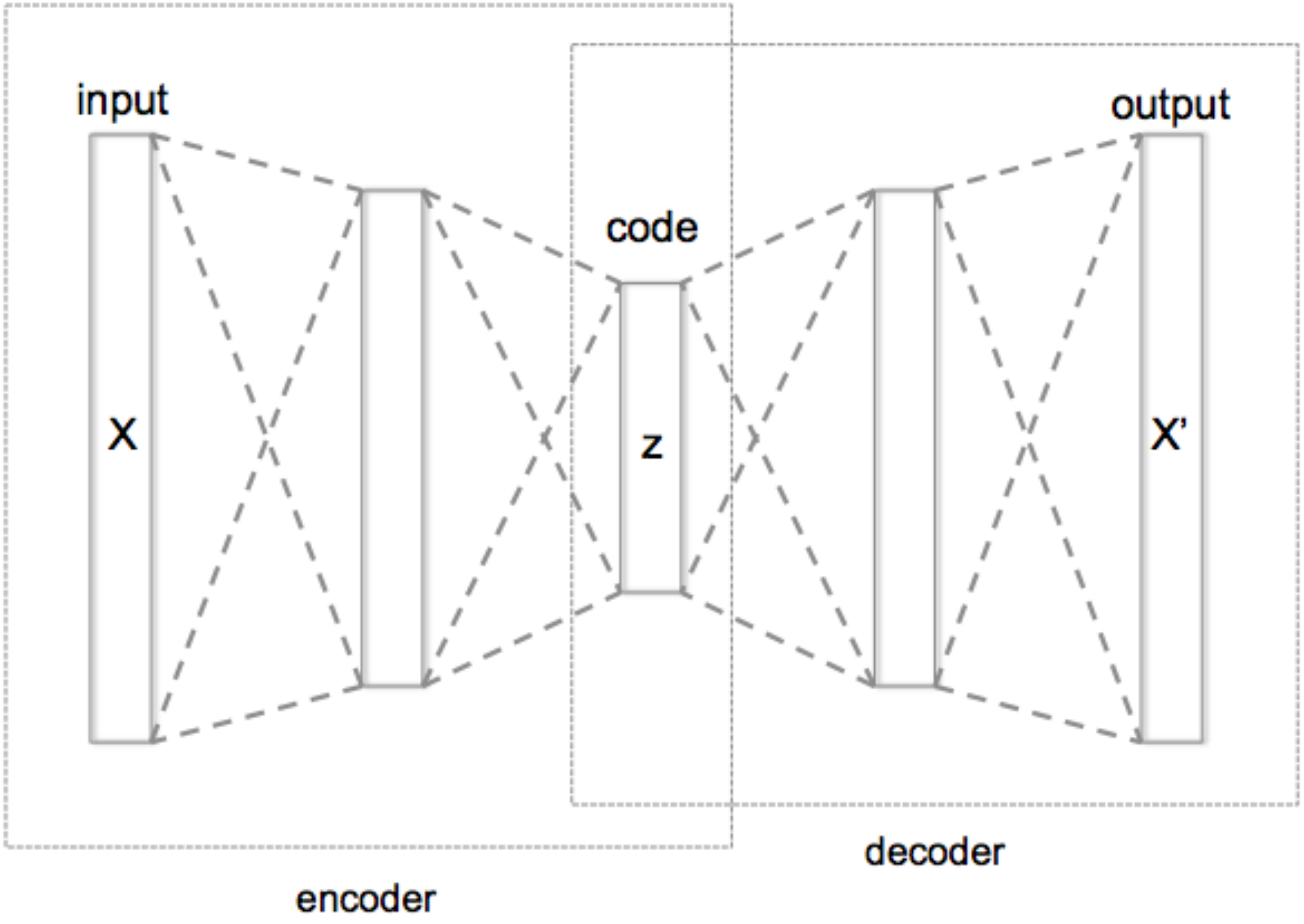
2022
Dall-E 2
Imagen
Stable diffusion

2023
SDXL
SDXL Turbo

Image Generation

- Fundamental challenge: images are incredibly high-dimensional objects with complex relationships between elements
- Idea: learn a low-dimensional representation of images, sample points in the low-dimensional space and project them up to the original image space

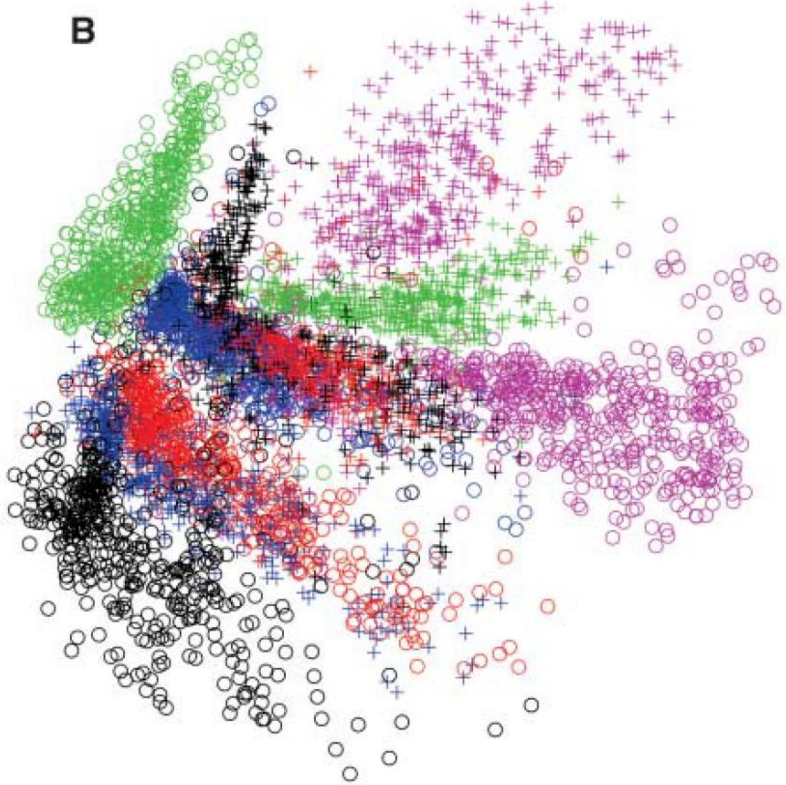
Recall: Autoencoders



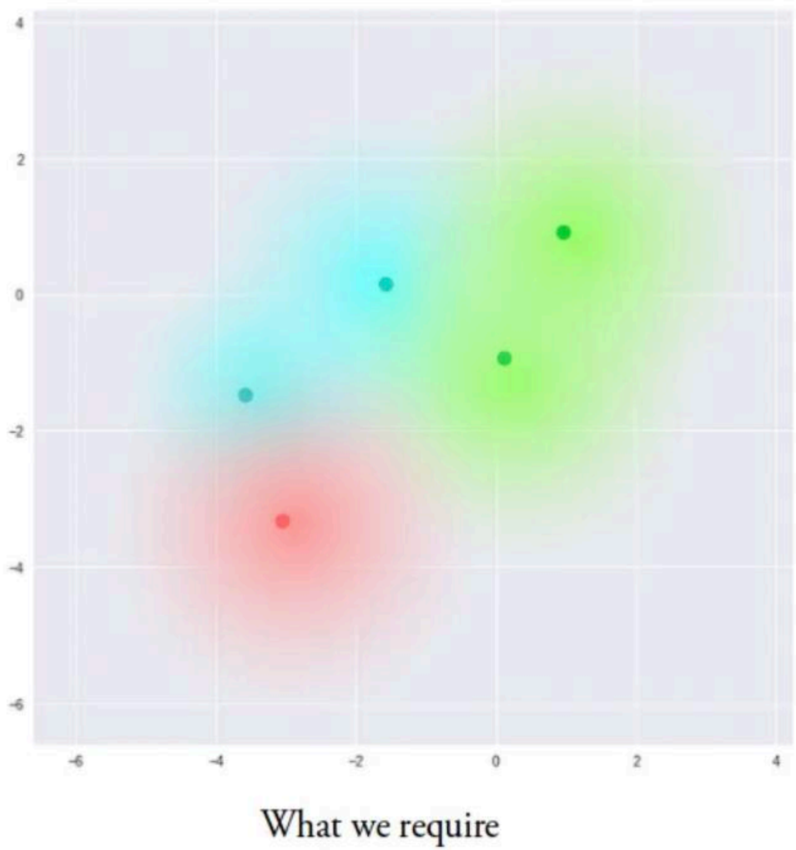
Source: https://en.wikipedia.org/wiki/Autoencoder#/media/File:Autoencoder_structure.png

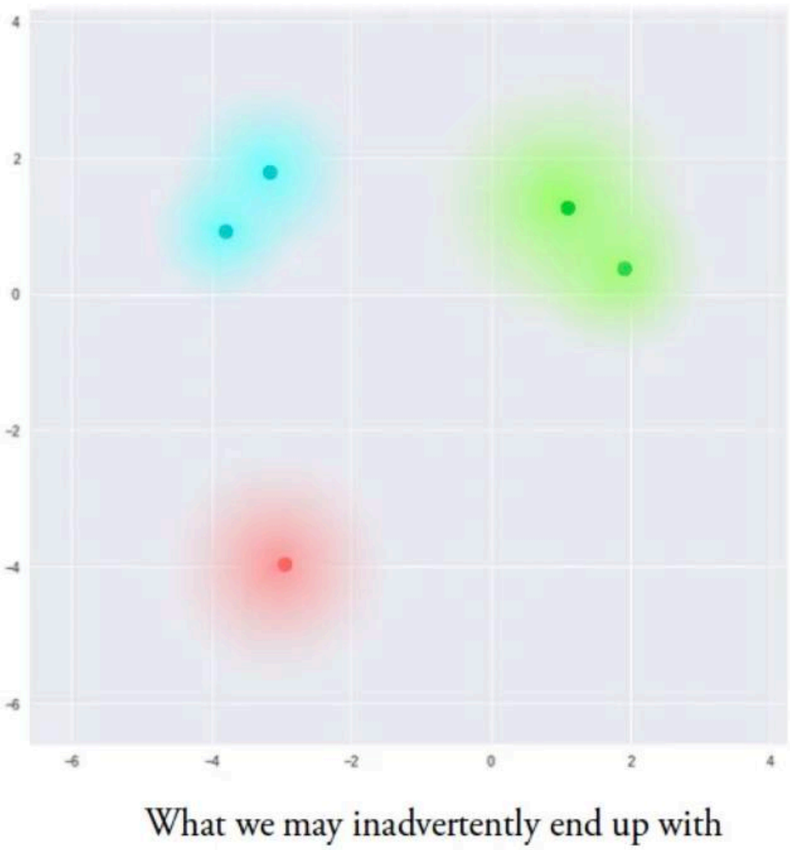
Autoencoder Latent Space

- Issue: latent space is sparse…
  - Sampling from latent space of an autoencoder creates outputs that are effectively identical to images in the training dataset



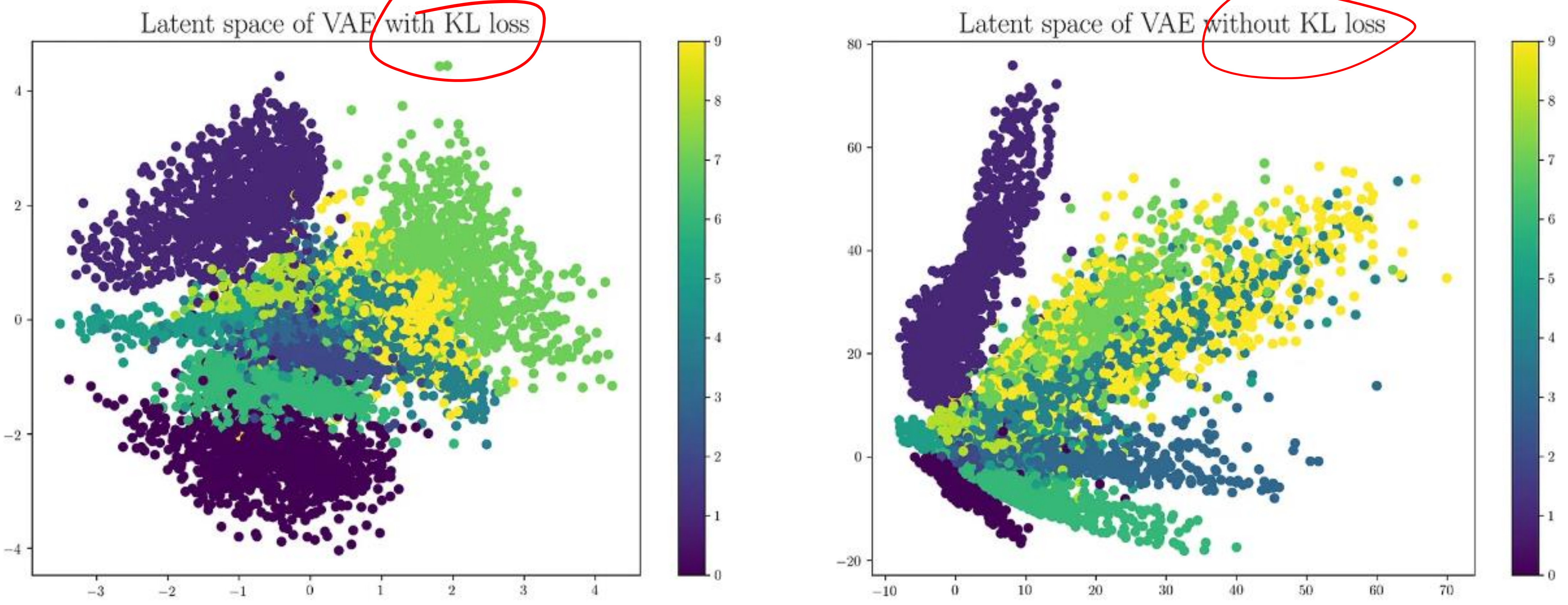
Source: https://www.science.org/doi/10.1126/science.1127647





Autoencoder Latent Space

Source: https://towardsdatascience.com/intuitively-understanding-variational-autoencoders-1bfe67eb5daf



Variational Autoencoder Latent Space

Source: https://towardsdatascience.com/difference-between-autoencoder-ae-and-variational-autoencoder-vae-ed7be1c038f2

Variational Autoencoder: Network Perspective

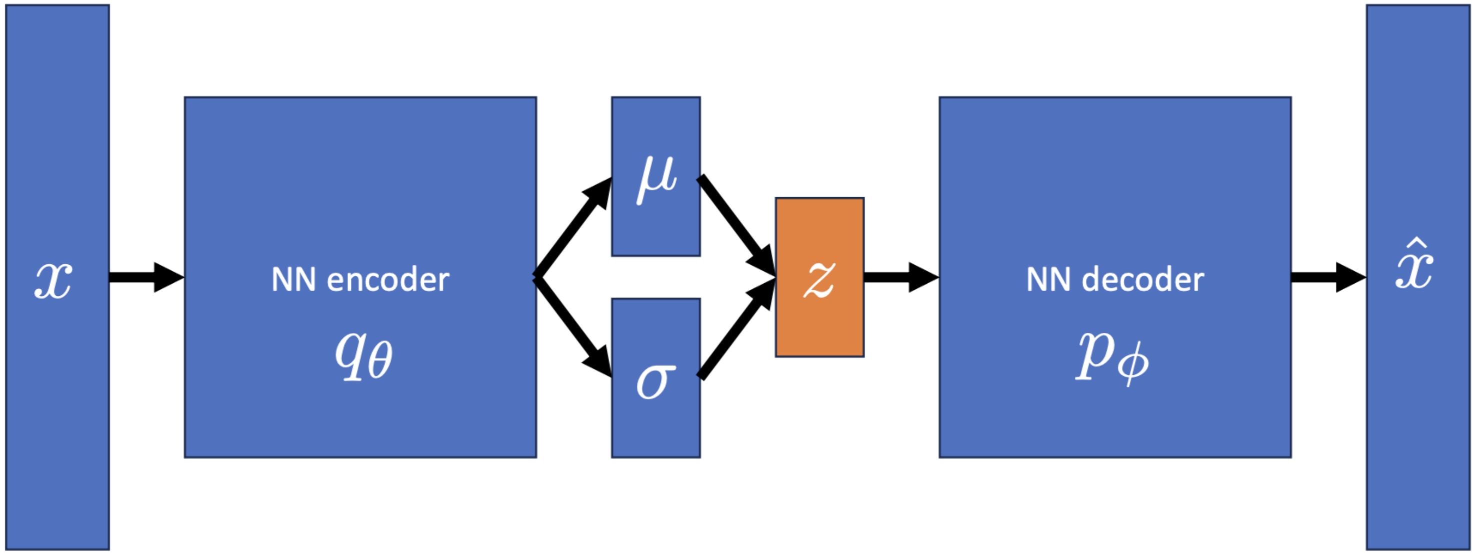
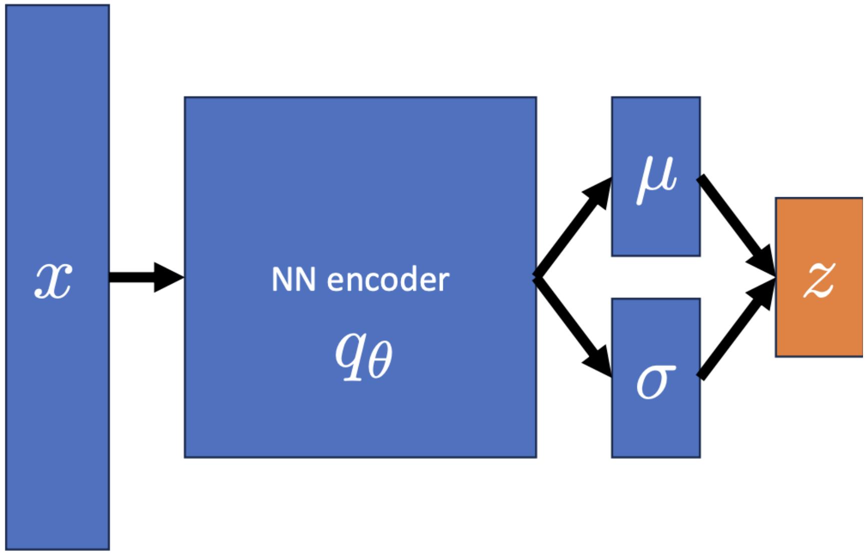


Figure courtesy of Zack Lipton

Variational Autoencoder: Network Perspective



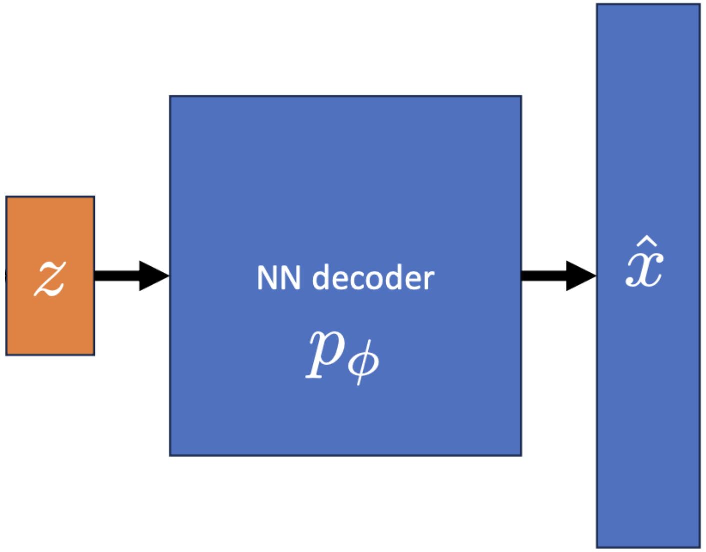
- Encoder learns a mean vector and a (diagonal) covariance matrix for each input

- These are used to *sample* a latent representation e.g.,

$$\boldsymbol{z}^{(i)} \mid \boldsymbol{x}^{(i)} \sim \mathcal{N}\left(\mu_{\boldsymbol{\theta}}(\boldsymbol{x}^{(i)}), \sigma_{\boldsymbol{\theta}}^2(\boldsymbol{x}^{(i)})\right)$$

Figure courtesy of Zack Lipton

Variational Autoencoder: Network Perspective

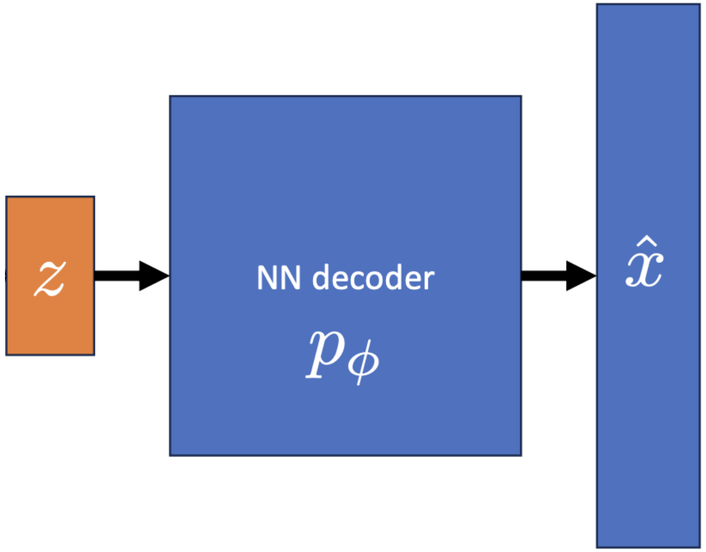


- Decoder tries to minimize the reconstruction error *in expectation* between $\boldsymbol{x}^{(i)}$ and a sample from another learned (conditional) distribution e.g.,

$$\widehat{\boldsymbol{x}}^{(i)} \mid \boldsymbol{z}^{(i)} \sim \mathcal{N}\left(\mu_{\boldsymbol{\phi}}\left(\boldsymbol{z}^{(i)}\right), \sigma_{\boldsymbol{\phi}}^2\left(\boldsymbol{z}^{(i)}\right)\right)$$

Figure courtesy of Zack Lipton

Variational Autoencoder: Network Perspective

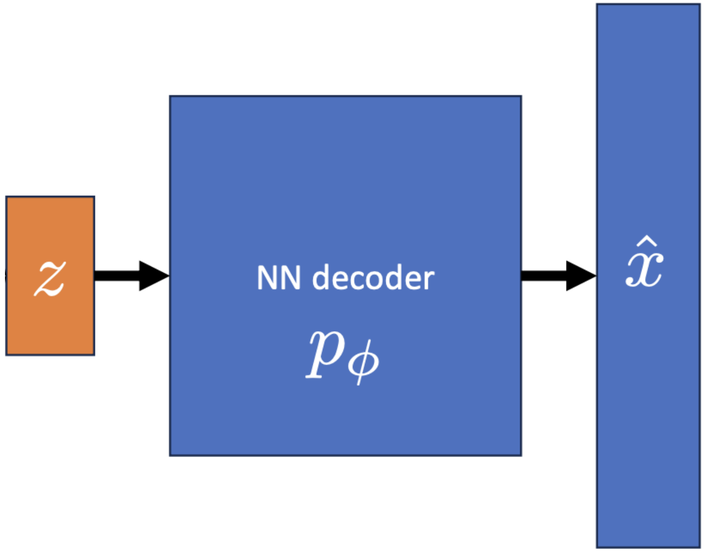


- Decoder tries to maximize the likelihood of the true $\boldsymbol{x}^{(i)}$ under another learned (conditional) distribution e.g.,

$$\widehat{\boldsymbol{x}}^{(i)} \mid \boldsymbol{z}^{(i)} \sim \mathcal{N}\left(\mu_{\boldsymbol{\phi}}\left(\boldsymbol{z}^{(i)}\right), \sigma_{\boldsymbol{\phi}}^{2}\left(\boldsymbol{z}^{(i)}\right)\right)$$

Figure courtesy of Zack Lipton

Variational Autoencoder: Network Perspective



- Decoder tries to minimize the negative log-likelihood of the true $\boldsymbol{x}^{(i)}$ under another learned (conditional) distribution e.g.,

$$\widehat{\boldsymbol{x}}^{(i)} \mid \boldsymbol{z}^{(i)} \sim \mathcal{N}\left(\mu_{\boldsymbol{\phi}}\left(\boldsymbol{z}^{(i)}\right), \sigma_{\boldsymbol{\phi}}^2\left(\boldsymbol{z}^{(i)}\right)\right)$$

Figure courtesy of Zack Lipton

Variational Autoencoder: Network Perspective

- Objective: minimize the negative log-likelihood of the dataset plus a *regularization term* that encourages a dense latent space

$$J(\boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_{i=1}^{N} \ell_i(\boldsymbol{\theta}, \boldsymbol{\phi})$$

$$\ell_i(\boldsymbol{\theta}, \boldsymbol{\phi}) = -\mathbb{E}_{q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\left[\log p_{\boldsymbol{\phi}}\left(\boldsymbol{x}^{(i)}\middle|\boldsymbol{z}\right)\right] + KL\left(q_{\boldsymbol{\theta}}\left(\boldsymbol{z}\middle|\boldsymbol{x}^{(i)}\right) \parallel p(\boldsymbol{z})\right)$$

- Objective: minimize the negative log-likelihood of the dataset plus a *regularization term* that encourages a dense latent space

$$J(\boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_{i=1}^{N} \ell_i(\boldsymbol{\theta}, \boldsymbol{\phi})$$

$$\ell_i(\boldsymbol{\theta}, \boldsymbol{\phi}) = -\mathbb{E}_{q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\left[\log p_{\boldsymbol{\phi}}\left(\boldsymbol{x}^{(i)}\middle|\boldsymbol{z}\right)\right] + KL\left(q_{\boldsymbol{\theta}}\left(\boldsymbol{z}\middle|\boldsymbol{x}^{(i)}\right) \parallel p(\boldsymbol{z})\right)$$

KL Divergence

- For two distributions $q(x)$ and $p(x)$ over $x \in \mathcal{X}$, the **Kullback-Leibler (KL) divergence** is

$$KL(q||p) = \mathbb{E}_q\left[\log \frac{q(x)}{p(x)}\right] = \sum_{x \in \mathcal{X}} q(x) \log \frac{q(x)}{p(x)}$$

KL Divergence

- For two distributions $q(x)$ and $p(x)$ over $x \in \mathcal{X}$, the **Kullback-Leibler (KL) divergence** is

$$KL(q||p) = \mathbb{E}_q\left[\log\frac{q(x)}{p(x)}\right] = \int_{\mathcal{X}} q(x) \log \frac{q(x)}{p(x)} dx$$

$p(x)$

- The KL divergence
  1. measures the **proximity** of two distributions q and p
  2. is minimized when $q(x) = p(x)$ for all $x \in \mathcal{X}$
  3. is **not** symmetric: $KL(q\,||\,p) \neq KL(p\,||\,q)$

KL Divergence: Example

- Keeping all else constant, consider the effect of differences between p and q for certain x' on $KL(q \,||\, p)$

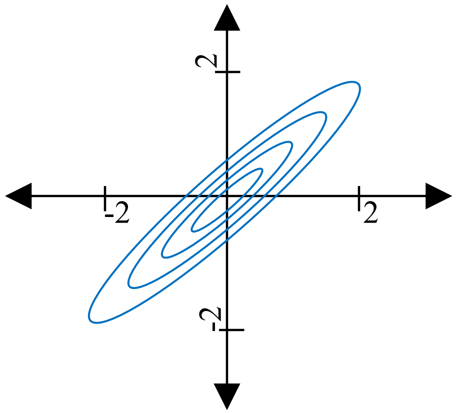
x'	$q(x')$	$p(x')$	$q(x') \log\left(\frac{q(x')}{p(x')}\right)$	effect on $KL(q \,\|\, p)$
1	0.9	0.9	0	no increase
2	0.9	0.1	1.97	big increase
3	0.1	0.9	−0.21	little decrease
4	0.1	0.1	0	little decrease

KL divergence cares about where $q(x)$ is large but cares much less about where $q(x)$ is small

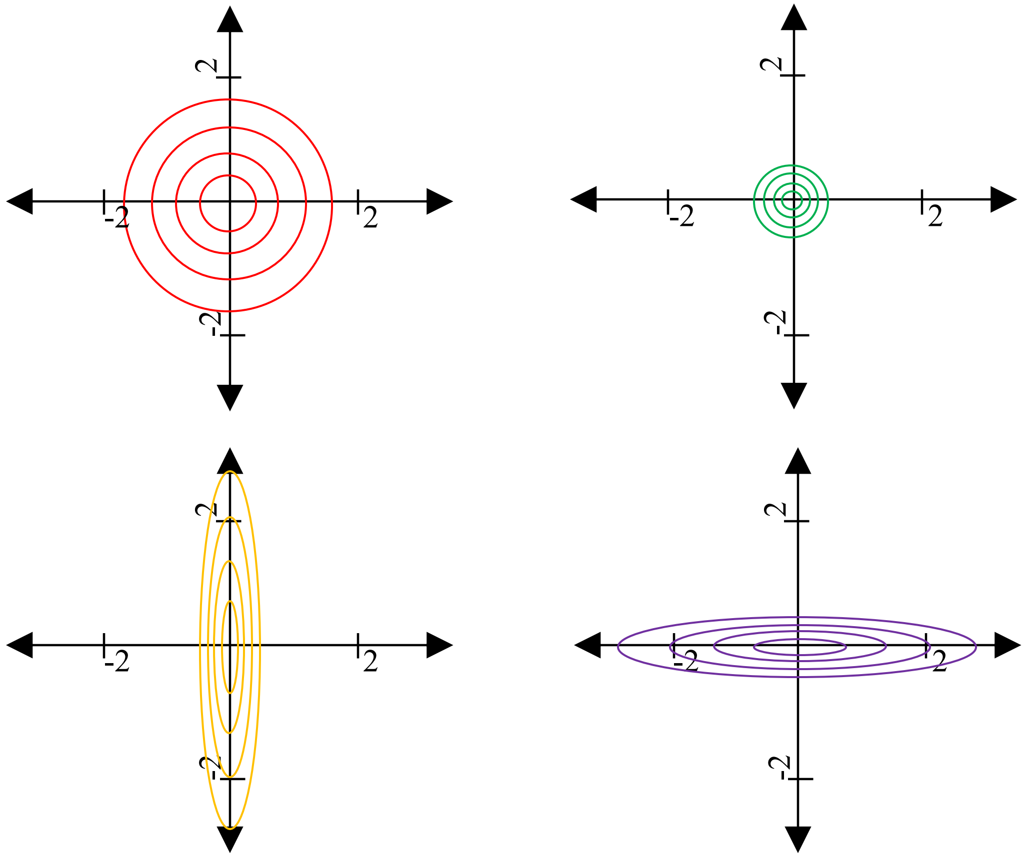
KL Divergence: In-class Exercise

- Which q minimizes $KL(q \,||\, p)$ for the given p?

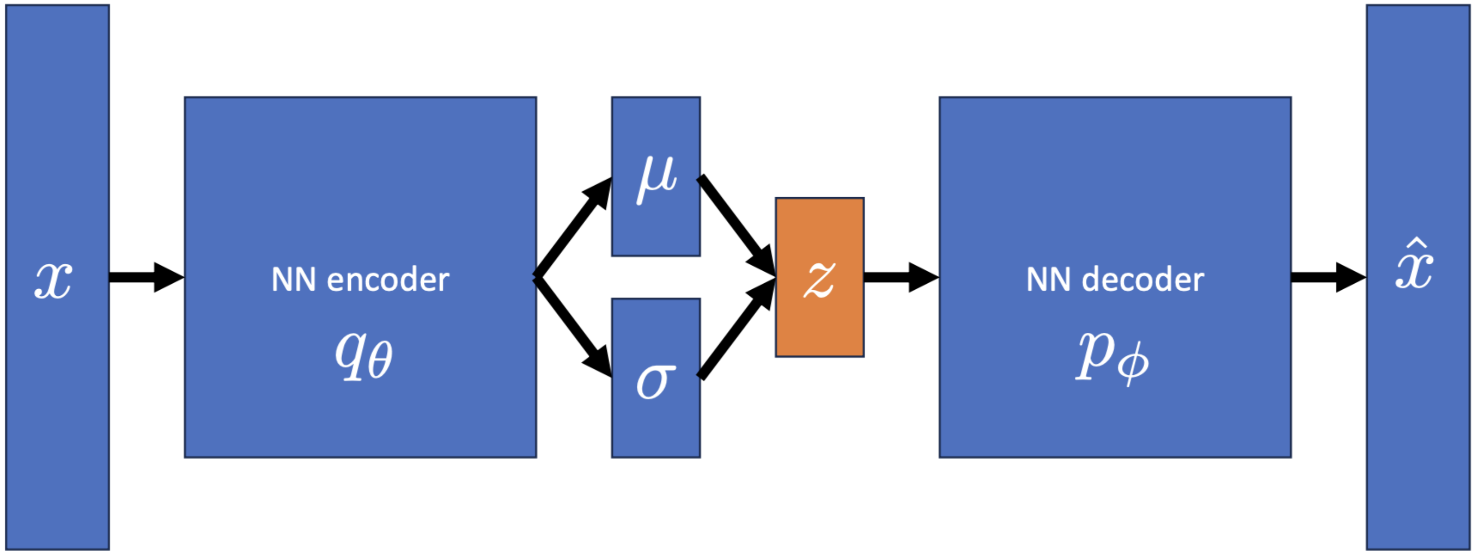
$$p(\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu} = [0,0]^T, \boldsymbol{\Sigma})$$



$$q(x_1, x_2) = \mathcal{N}_1(x_1 \mid \mu_1, \sigma_1^2)\mathcal{N}_2(x_2 \mid \mu_2, \sigma_2^2)$$



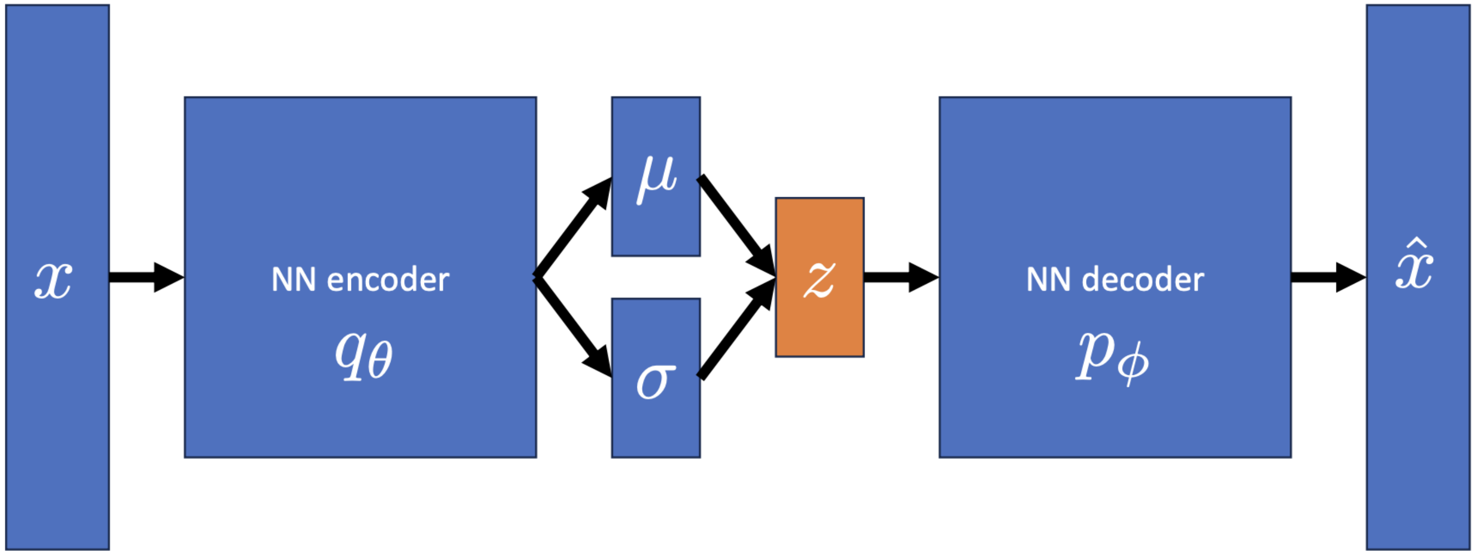
Variational Autoencoder: Network Perspective



- Objective: minimize the negative log-likelihood of the dataset plus a *regularization term* that encourages a dense latent space

$$J(\boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_{i=1}^{N} \ell_i(\boldsymbol{\theta}, \boldsymbol{\phi})$$

$$\ell_i(\boldsymbol{\theta}, \boldsymbol{\phi}) = -\mathbb{E}_{q_{\boldsymbol{\theta}}(\mathbf{z}|\boldsymbol{x}^{(i)})}\left[\log p_{\boldsymbol{\phi}}\left(\boldsymbol{x}^{(i)} \middle| \mathbf{z}\right)\right] + KL\left(q_{\boldsymbol{\theta}}\left(\mathbf{z} \middle| \boldsymbol{x}^{(i)}\right) \parallel p(\mathbf{z})\right)$$



So what should we set p to?

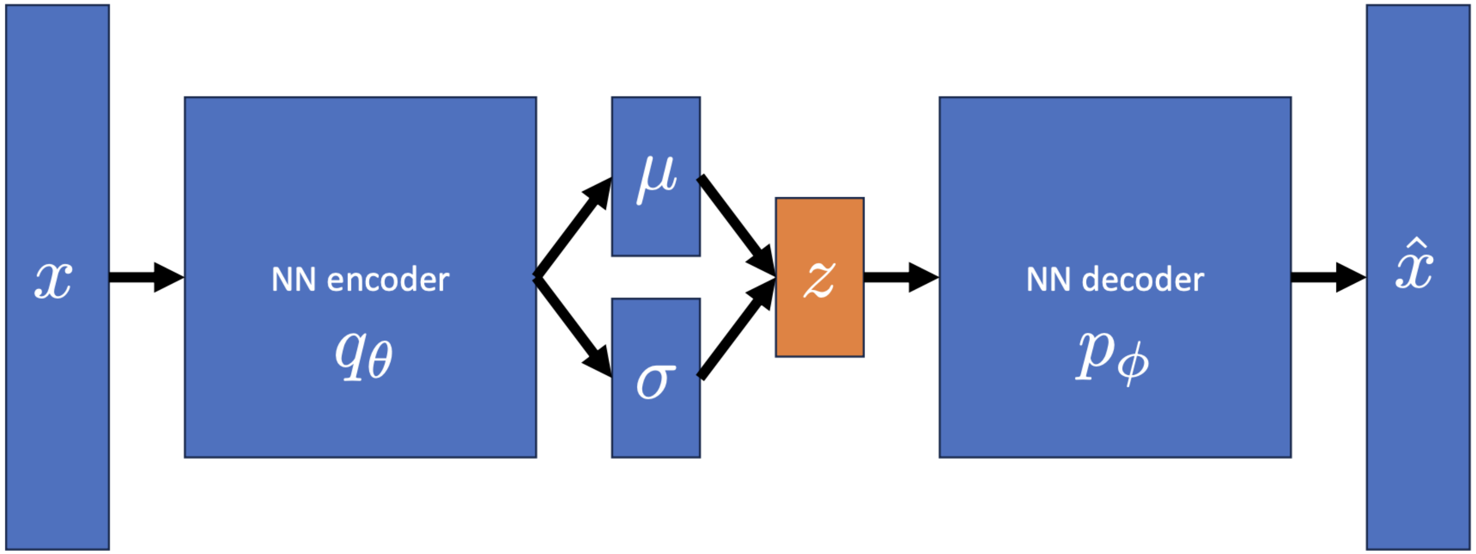
- Objective: minimize the negative log-likelihood of the dataset plus a *regularization term* that encourages a **dense latent space**

$$J(\boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_{i=1}^{N} \ell_i(\boldsymbol{\theta}, \boldsymbol{\phi})$$

$$\ell_i(\boldsymbol{\theta}, \boldsymbol{\phi}) = -\mathbb{E}_{q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\left[\log p_{\boldsymbol{\phi}}\left(\boldsymbol{x}^{(i)} \middle| \boldsymbol{z}\right)\right] + KL\left(q_{\boldsymbol{\theta}}\left(\boldsymbol{z} \middle| \boldsymbol{x}^{(i)}\right) \parallel p(\boldsymbol{z})\right)$$

c.f. $\mathcal{N}(0, I)$

Variational Autoencoder: Network Perspective



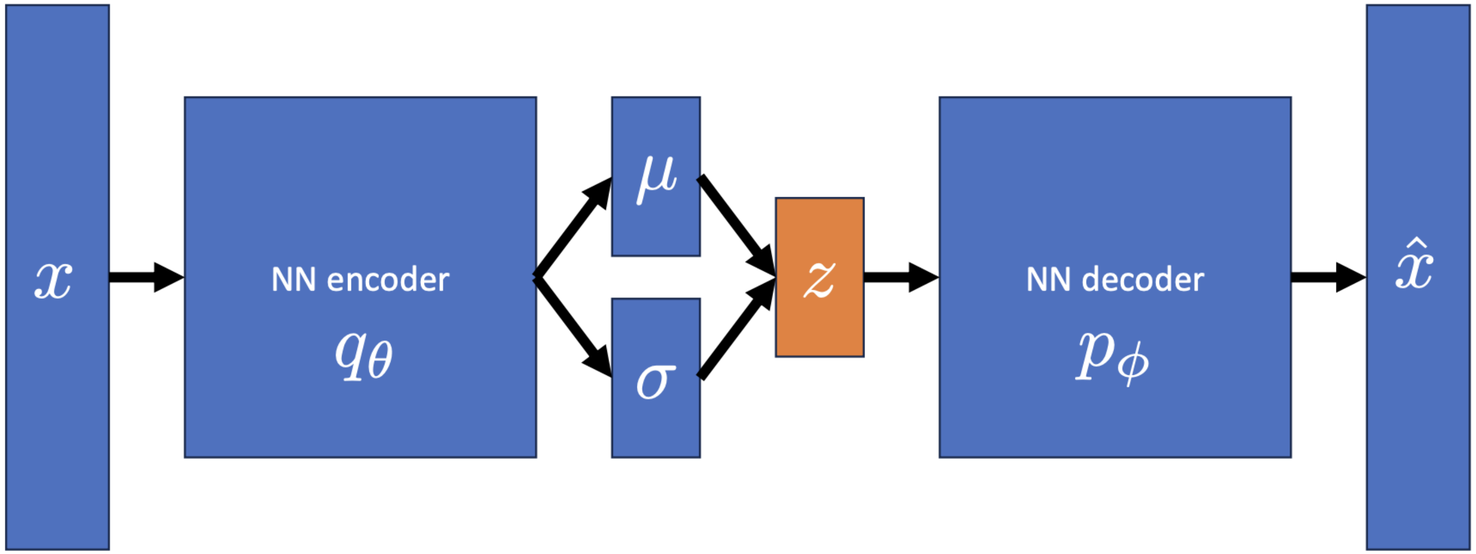
- Objective: minimize the negative log-likelihood of the dataset plus a *regularization term* that encourages a dense latent space

$$J(\boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_{i=1}^{N} \ell_i(\boldsymbol{\theta}, \boldsymbol{\phi})$$

$$\ell_i(\boldsymbol{\theta}, \boldsymbol{\phi}) \approx -\left(\sum_{s=1}^{S} \log p_{\boldsymbol{\phi}}\left(\boldsymbol{x}^{(i)} \middle| \boldsymbol{z}_s\right)\right) + KL\left(q_{\boldsymbol{\theta}}\left(\boldsymbol{z} \middle| \boldsymbol{x}^{(i)}\right) \parallel p(\boldsymbol{z})\right)$$

for samples $\boldsymbol{z}_1, \ldots, \boldsymbol{z}_S \sim q_{\boldsymbol{\theta}}\left(\boldsymbol{z} \mid \boldsymbol{x}^{(i)}\right)$

Can we backpropagate through q_θ given that samples of $\boldsymbol{z}$ are stochastic?

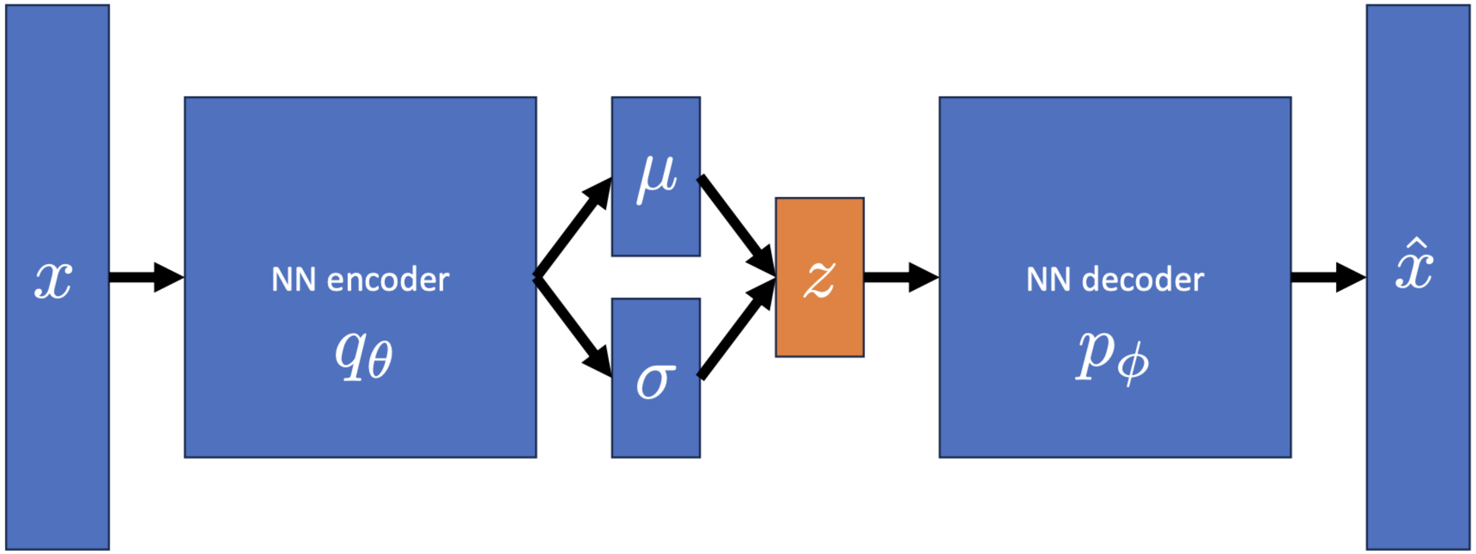


- Objective: minimize the negative log-likelihood of the dataset plus a *regularization term* that encourages a dense latent space

$$J(\boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_{i=1}^{N} \ell_i(\boldsymbol{\theta}, \boldsymbol{\phi})$$

$$\ell_i(\boldsymbol{\theta}, \boldsymbol{\phi}) \approx -\left(\sum_{s=1}^{S} \log p_{\boldsymbol{\phi}}\left(\boldsymbol{x}^{(i)} \middle| \boldsymbol{z}_s\right)\right) + KL\left(q_{\boldsymbol{\theta}}\left(\boldsymbol{z} \middle| \boldsymbol{x}^{(i)}\right) \parallel p(\boldsymbol{z})\right)$$

for samples $\boldsymbol{z}_1, \dots, \boldsymbol{z}_S \sim q_{\boldsymbol{\theta}}\left(\boldsymbol{z} \mid \boldsymbol{x}^{(i)}\right)$



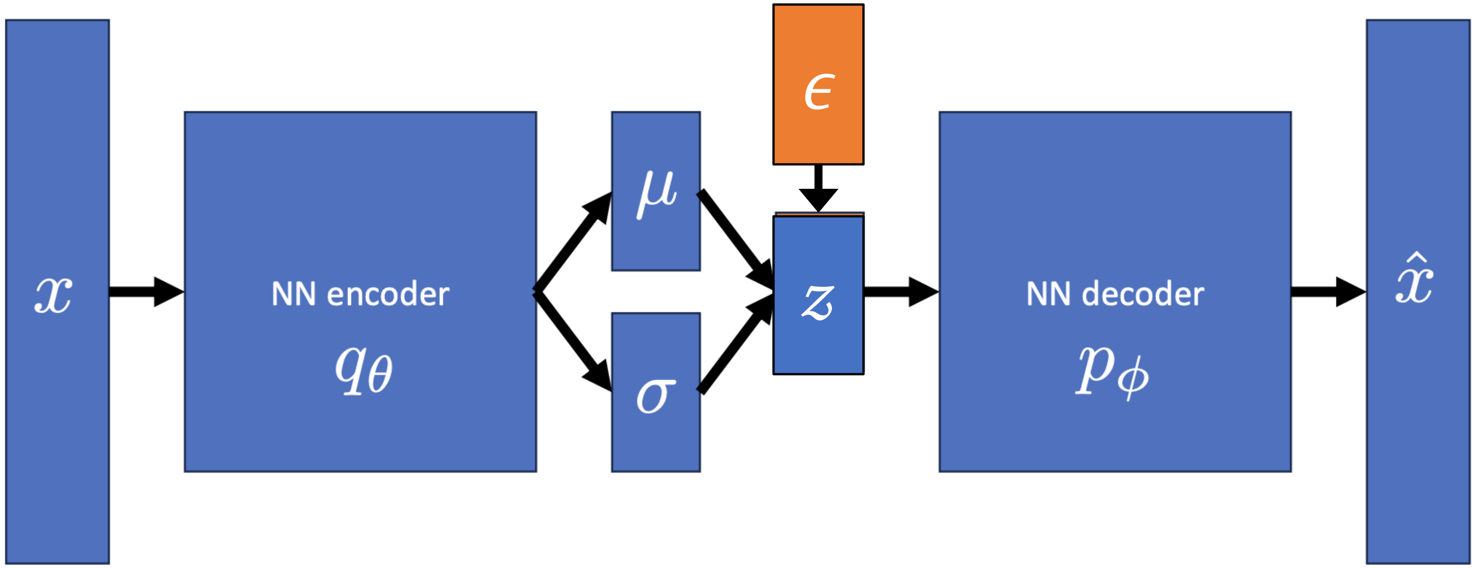
Reparameterization Trick

- Objective: minimize the negative log-likelihood of the dataset plus a *regularization term* that encourages a dense latent space

$$J(\boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_{i=1}^{N} \ell_i(\boldsymbol{\theta}, \boldsymbol{\phi})$$

$$\ell_i(\boldsymbol{\theta}, \boldsymbol{\phi}) \approx -\left(\sum_{s=1}^{S} \log p_{\boldsymbol{\phi}}\left(\boldsymbol{x}^{(i)} \middle| \boldsymbol{z}_s(\boldsymbol{\theta})\right)\right) + KL\left(q_{\boldsymbol{\theta}}\left(\boldsymbol{z} \middle| \boldsymbol{x}^{(i)}\right) \parallel p(\boldsymbol{z})\right)$$

for $\boldsymbol{z}_s(\boldsymbol{\theta}) = \mu_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) + \boldsymbol{\sigma}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) \odot \boldsymbol{\epsilon}_s$ where $\boldsymbol{\epsilon}_s \sim N(\mathbf{0}, I)$



Reparameterization Trick

- Objective: minimize the negative log-likelihood of the dataset plus a *regularization term* that encourages a dense latent space

$$J(\boldsymbol{\theta}, \boldsymbol{\phi}) = \sum_{i=1}^{N} \ell_i(\boldsymbol{\theta}, \boldsymbol{\phi})$$

$$\ell_i(\boldsymbol{\theta}, \boldsymbol{\phi}) \approx -\left(\sum_{s=1}^{S} \log p_{\boldsymbol{\phi}}\left(\boldsymbol{x}^{(i)} \middle| \boldsymbol{z}_s(\boldsymbol{\theta})\right)\right) + KL\left(q_{\boldsymbol{\theta}}\left(\boldsymbol{z} \middle| \boldsymbol{x}^{(i)}\right) \parallel p(\boldsymbol{z})\right)$$

for $\boldsymbol{z}_s(\boldsymbol{\theta}) = \mu_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) + \boldsymbol{\sigma}_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) \odot \boldsymbol{\epsilon}_s$ where $\boldsymbol{\epsilon}_s \sim N(\boldsymbol{0}, I)$

$$\ell_i(\boldsymbol{\theta}, \boldsymbol{\phi}) = -\mathbb{E}_{q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x}^{(i)})}\left[\log p_{\boldsymbol{\phi}}\left(\boldsymbol{x}^{(i)}\middle|\boldsymbol{z}\right)\right] + KL\left(q_{\boldsymbol{\theta}}\left(\boldsymbol{z}\middle|\boldsymbol{x}^{(i)}\right) \parallel p(\boldsymbol{z})\right)$$

$$\approx -\left(\sum_{s=1}^{S} \log p_{\boldsymbol{\phi}}\left(\boldsymbol{x}^{(i)}\middle|\boldsymbol{z}_s(\boldsymbol{\theta})\right)\right) + KL\left(q_{\boldsymbol{\theta}}\left(\boldsymbol{z}\middle|\boldsymbol{x}^{(i)}\right) \parallel p(\boldsymbol{z})\right)$$

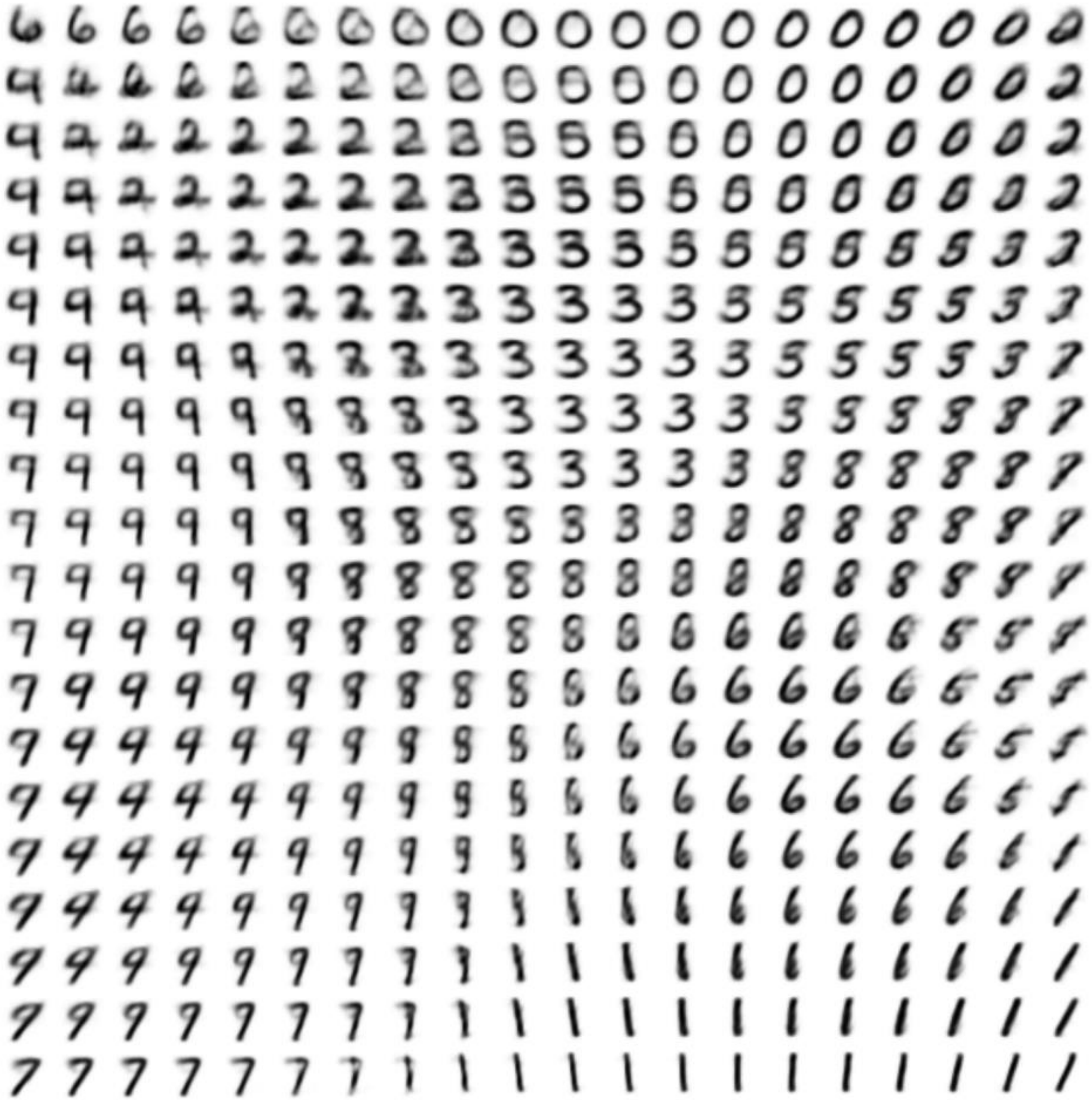
$$= -\left(\sum_{s=1}^{S} \underline{N\left(x^{(i)}; \mu_\phi(z_s(\theta)), \sigma_\phi^2(z_s(\theta))\right)}\right)$$

$$+ \underbrace{KL\left(N\left(\mu_\theta(x^{(i)}), \sigma_\theta^2(x^{(i)})\right) \parallel N(0, I)\right)}$$

where $z_s \sim N\left(\mu_\theta(x^{(i)}), \sigma_\theta^2(x^{(i)})\right)$

Variational Autoencoder: Objective Function

Variational Autoencoder: Latent Space Visualization



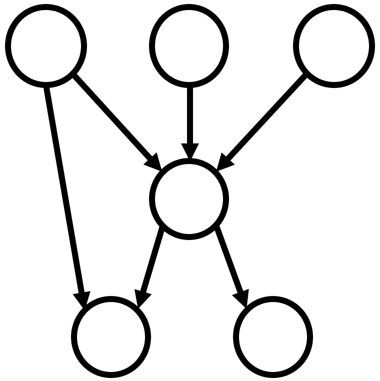
Source: https://arxiv.org/pdf/1312.6114.pdf

Variational Autoencoder: Generated Samples…

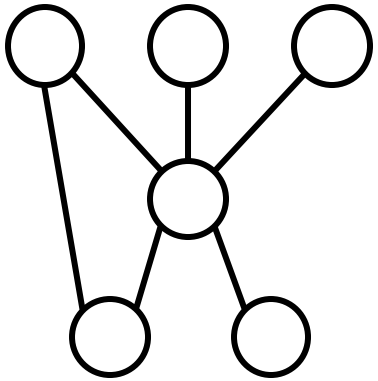
Source: https://arxiv.org/pdf/1312.6114.pdf

Three Types of Graphical Models

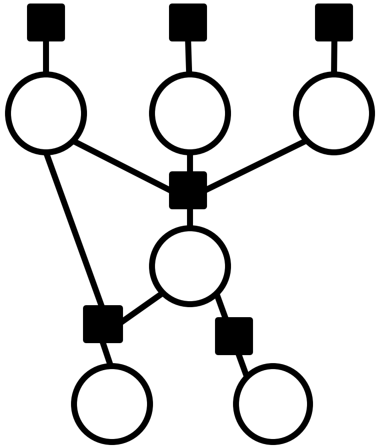
Directed Graphical Model



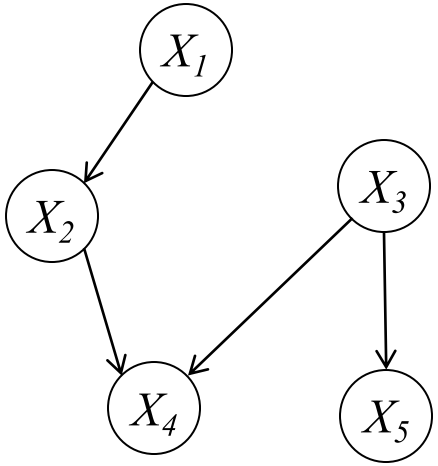
Undirected Graphical Model



Factor Graph

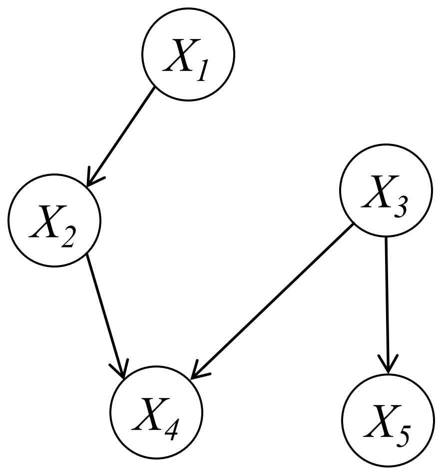


Directed Graphical Models a.k.a. Bayesian Networks



$$
\begin{aligned}
p(x_1, \dots, x_5) &:= p(x_1) \cdot p(x_2 \mid x_1) \\
&\quad \cdot p(x_3) \cdot p(x_4 \mid x_2, x_3) \\
&\quad \cdot p(x_5 \mid x_3)
\end{aligned}
$$

Directed Graphical Models a.k.a. Bayesian Networks



$$P(X_1, \dots, X_D) = \prod_{d=1}^{D} P(X_d \mid \text{parents}(X_d))$$

A Bayesian Network consists of:

- a graph G (the *qualitative specification*), which can be
  - specified using prior knowledge / domain expertise
  - learned from the training data (model selection)
- conditional probabilities (the *quantitative specification*)
  - these will depend on the relative types of the variables