11. Deep Learning Applications II: Generative Models

CS 5242 Neural Networks and Deep Learning

Ai Xin

Agenda

- Introduction
- Autoencoder
- GAN

Introduction

Supervised vs. Unsupervised Learning

- Supervised learning
  - Classification & Regression,
  - Object Detection,
  - Semantic Segmentation
  - Data (x, y)
    - x is data, y is label
  - Goal
    - Learn a function to map x -> y

- Unsupervised learning
  - Clustering,
  - dimensionality reduction
  - Feature learning (Autoencoder)
  - Data (x)
    - Just data, no labels!
  - Goal
    - Learn some underlying hidden structure of the data

Discriminative vs. Generative Model

- Discriminative Model
  - Learn a probability distribution $p(y|x)$
  - Data (x, y)
    - x is data, y is label

- Generative Model
  - Learn a probability distribution $p(x)$
  - Data (x)
    - Just data, no labels!

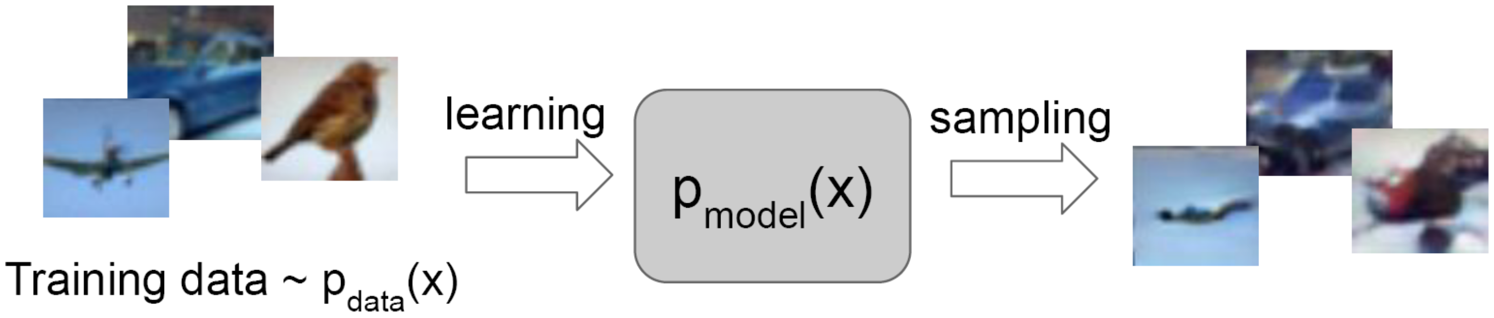
Data: x

Label: y

CAT

Generative Model

- Given training data, generate new samples from same distribution



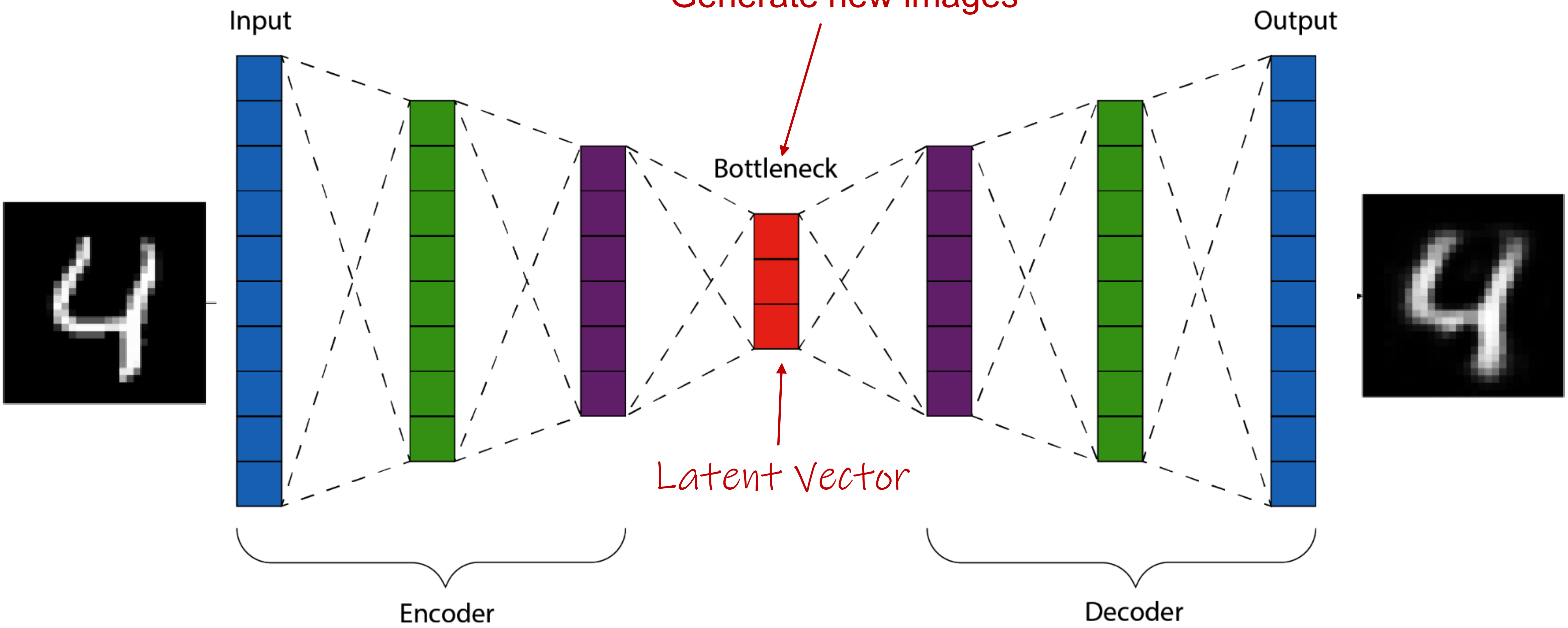
Objectives:

1. Learn $p_{model}(x)$ that approximates $p_{data}(x)$
2. **Sampling new x from $p_{model}(x)$**

Autoencoder

Autoencoder

- A lower dimensional feature
- Reconstruct the original image
- Identify similar images
- Generate new images

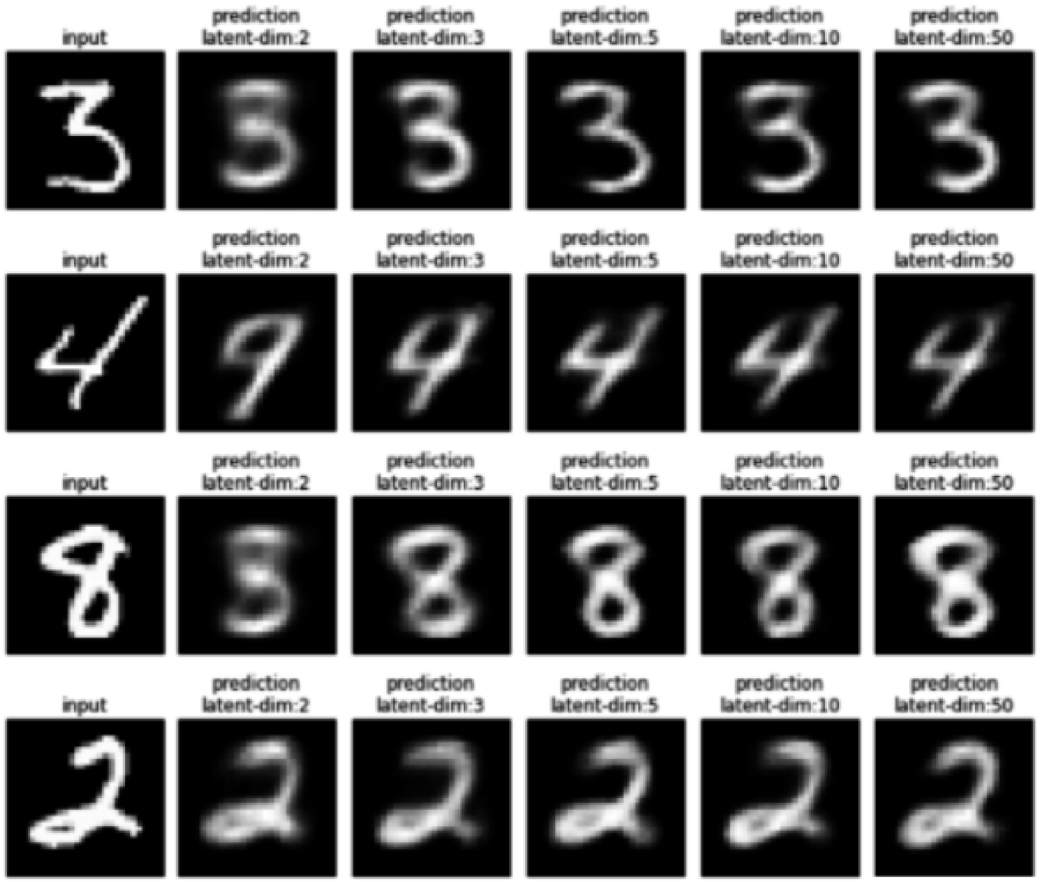


Lab 1: implement a vanilla autoencoder

Bottleneck Layer

Latent_dim = 3

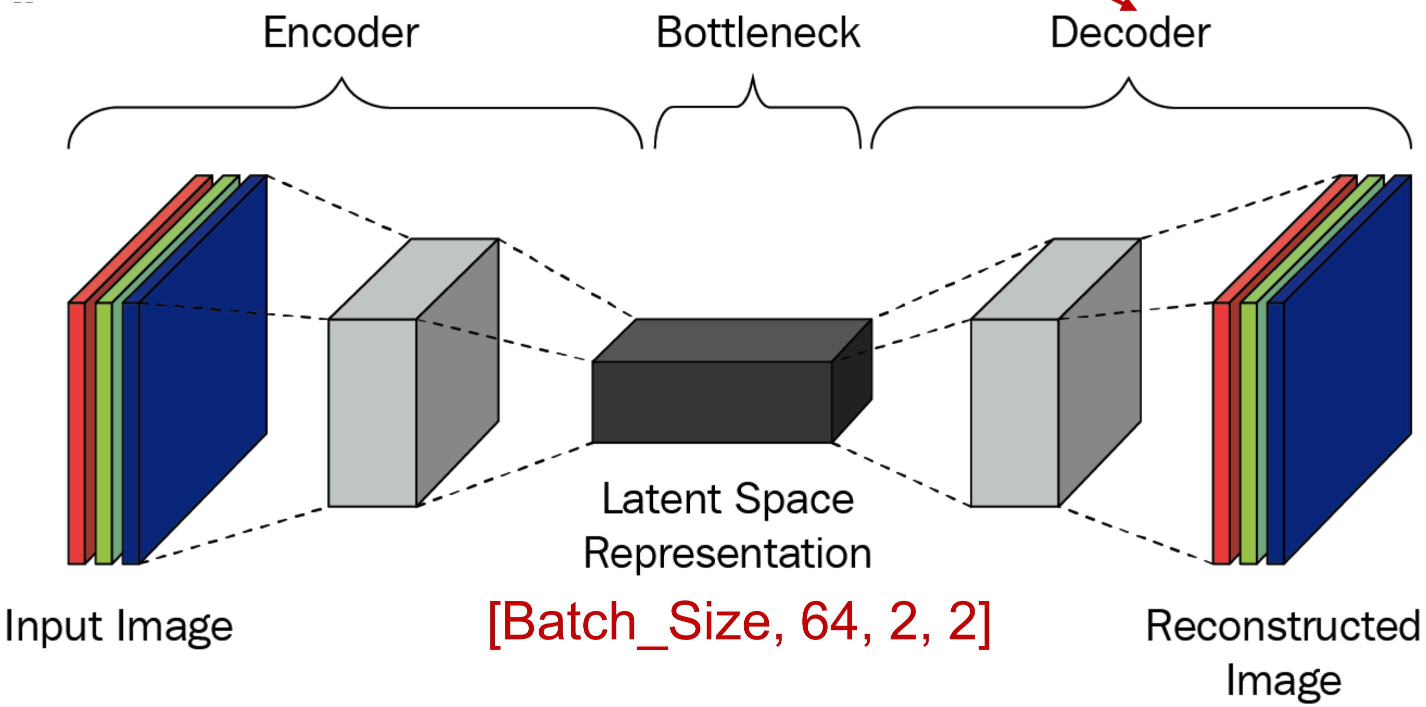
```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Sequential: 1-1                        [-1, 3]                   --
|    └─Linear: 2-1                       [-1, 128]                 100,480
|    └─ReLU: 2-2                         [-1, 128]                 --
|    └─Linear: 2-3                       [-1, 64]                  8,256
|    └─ReLU: 2-4                         [-1, 64]                  --
|    └─Linear: 2-5                       [-1, 3]                   195
├─Sequential: 1-2                        [-1, 784]                 --
|    └─Linear: 2-6                       [-1, 64]                  256
|    └─ReLU: 2-7                         [-1, 64]                  --
|    └─Linear: 2-8                       [-1, 128]                 8,320
|    └─ReLU: 2-9                         [-1, 128]                 --
|    └─Linear: 2-10                      [-1, 784]                 101,136
|    └─Tanh: 2-11                        [-1, 784]                 --
==========================================================================================
Total params: 218,643
Trainable params: 218,643
Non-trainable params: 0
Total mult-adds (M): 0.43
```



Higher Latent_dim ➔
Improved Clarity of reconstructed image

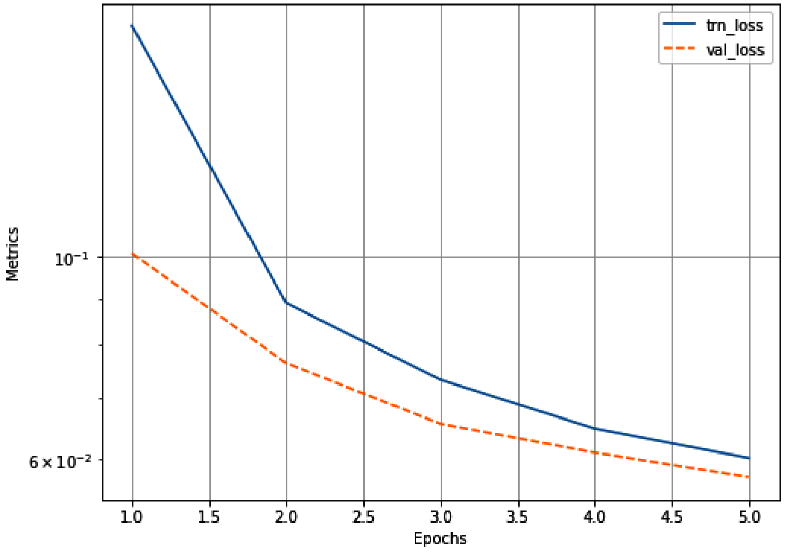
Lab 2: implement a convolutional autoencoder

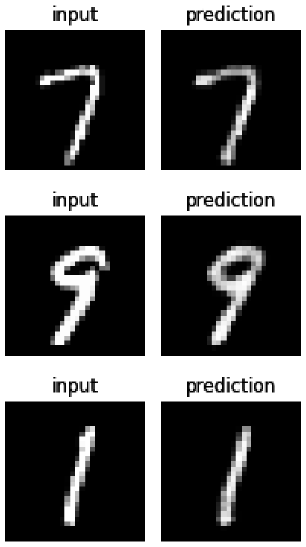
```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
├─Sequential: 1-1                        [-1, 64, 2, 2]            --
|    └─Conv2d: 2-1                       [-1, 32, 10, 10]          320
|    └─ReLU: 2-2                         [-1, 32, 10, 10]          --
|    └─MaxPool2d: 2-3                    [-1, 32, 5, 5]            --
|    └─Conv2d: 2-4                       [-1, 64, 3, 3]            18,496
|    └─ReLU: 2-5                         [-1, 64, 3, 3]            --
|    └─MaxPool2d: 2-6                    [-1, 64, 2, 2]            --
├─Sequential: 1-2                        [-1, 1, 28, 28]           --
|    └─ConvTranspose2d: 2-7              [-1, 32, 5, 5]            18,464
|    └─ReLU: 2-8                         [-1, 32, 5, 5]            --
|    └─ConvTranspose2d: 2-9              [-1, 16, 15, 15]          12,816
|    └─ReLU: 2-10                        [-1, 16, 15, 15]          --
|    └─ConvTranspose2d: 2-11             [-1, 1, 28, 28]           65
|    └─Tanh: 2-12                        [-1, 1, 28, 28]           --
==================================================================
Total params: 50,161
Trainable params: 50,161
Non-trainable params: 0
Total mult-adds (M): 3.64
```

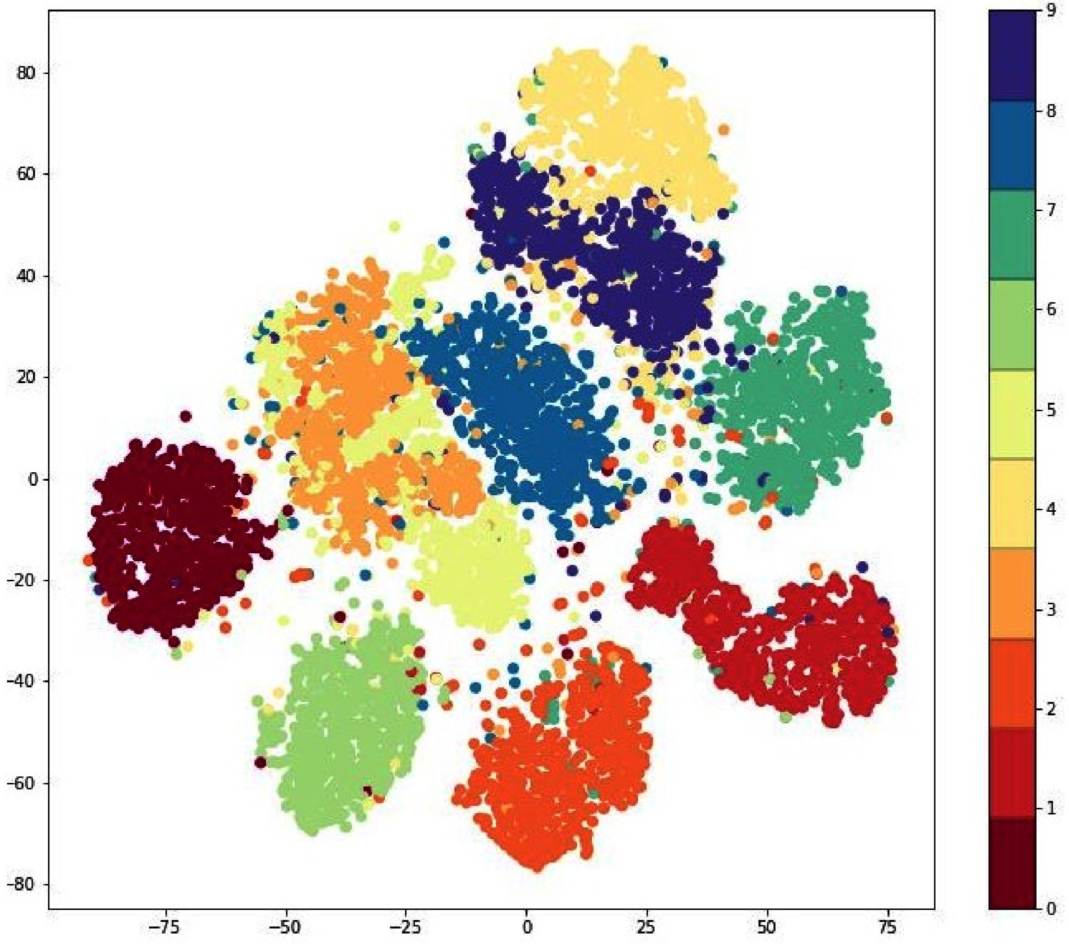


Lab 2: implement a convolutional autoencoder

Model Performance







Grouping similar images using t-SNE

(64-dimensional vector ➔ 2 dimensional vector)

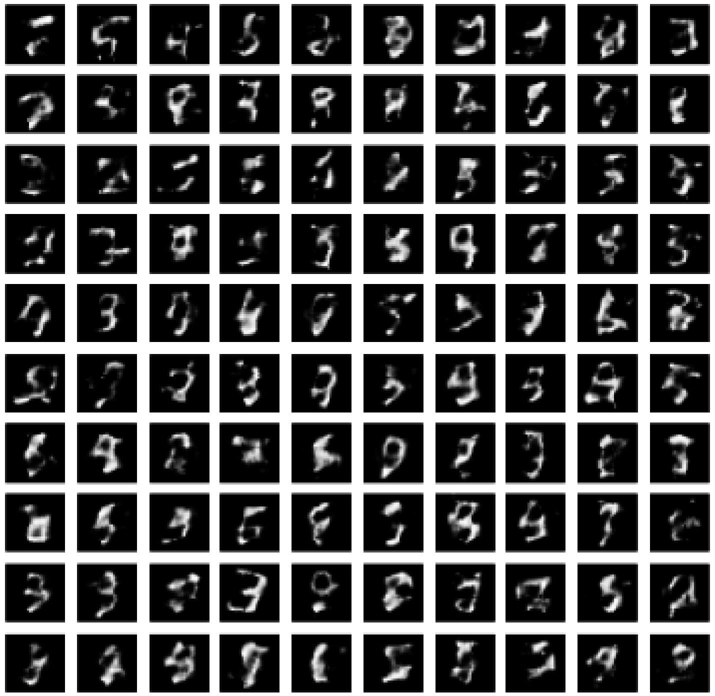
Latent Vectors

Lab 2: implement a convolutional autoencoder

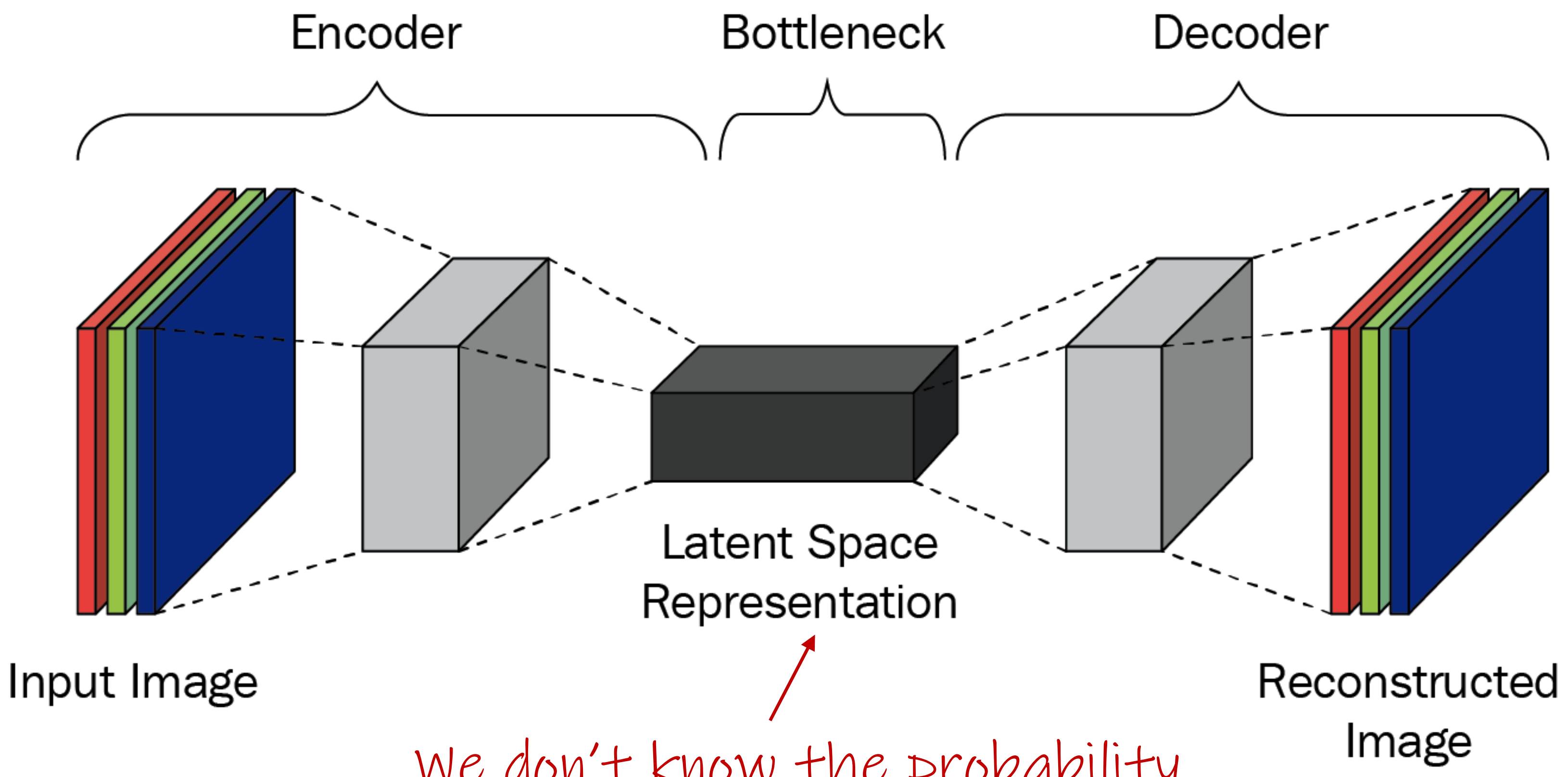
- Create random vectors to generate new images
  - rand_vectors are generated to using mean & variance of latent_vectors

```
rand_vectors = []
for col in latent_vectors.transpose(1,0):
    mu, sigma = col.mean(), col.std()
    rand_vectors.append(sigma*torch.randn(1,100) + mu)
```

- Newly generated images are less clear than before
- Simply using rand_vectors cannot generate realistic images



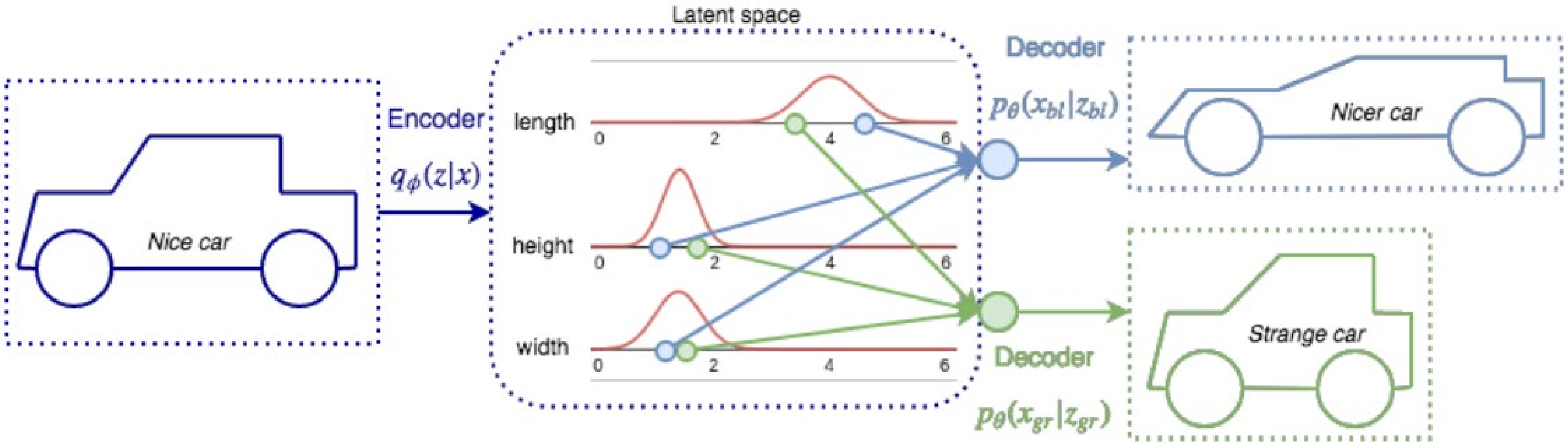
Why autoencoder cannot generate realistic images?



we don't know the probability distribution of latent space z

Variational Autoencoder (VAE)

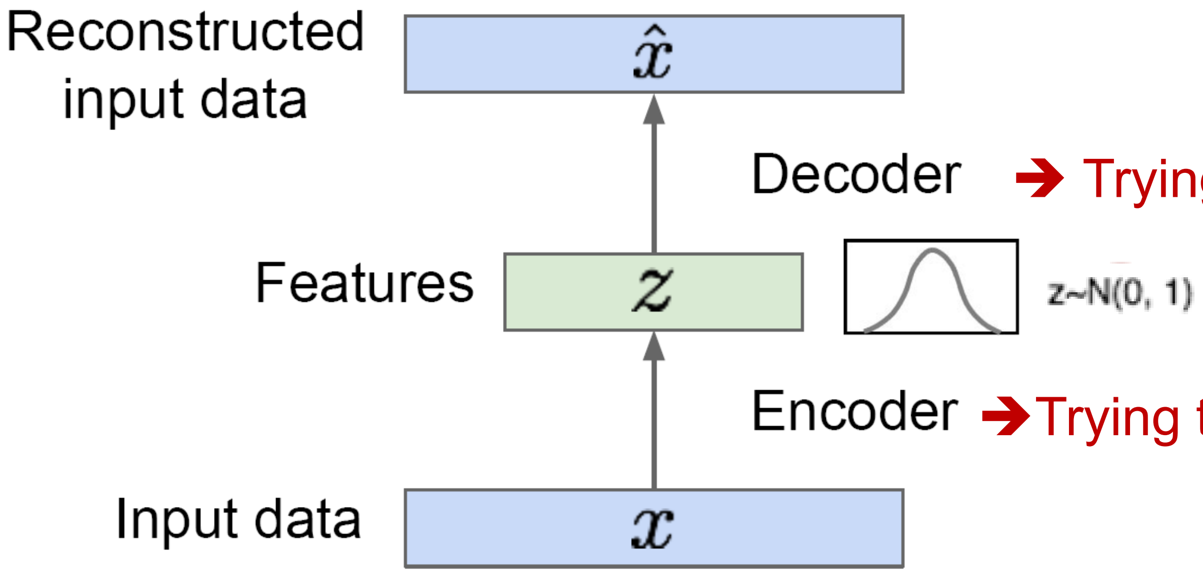
- VAE can describe a latent representation in probabilistic terms
  - a probability distribution for each latent attribute
  - makes it easier for random sampling and interpolation
- Example Illustration



Variational Autoencoder (VAE)

- Objective: Learn model parameters to maximize likelihood of training data

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



Decoder ➔ Trying to learn p(x|z)

z~N(0, 1) ➔ Assume simple normal distribution for p(z)

Encoder ➔ Trying to learn p(z|x)

decoder weights

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

approximate

$q_\varphi(\mathbf{z}|\mathbf{x})$

encoder weights

VAE Loss Function

KL divergence loss

- $p(z) \sim N(0, 1)$
- $q(z|x) \sim N(\mu, \sigma)$

$$\boxed{\sum_{i=1}^{n} \sigma_i^2 + \mu_i^2 - log(\sigma_i) - 1}$$

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

$$\Downarrow$$

Minimize $L(\theta, \varphi; \mathbf{x}) = \underbrace{-D_{KL}(q_\varphi(\mathbf{z}|\mathbf{x}) || p_\theta(\mathbf{z}))} + \underbrace{E_{q_\varphi(\mathbf{z}|\mathbf{x})}[\log(p_\theta(\mathbf{x}|\mathbf{z}))]}$

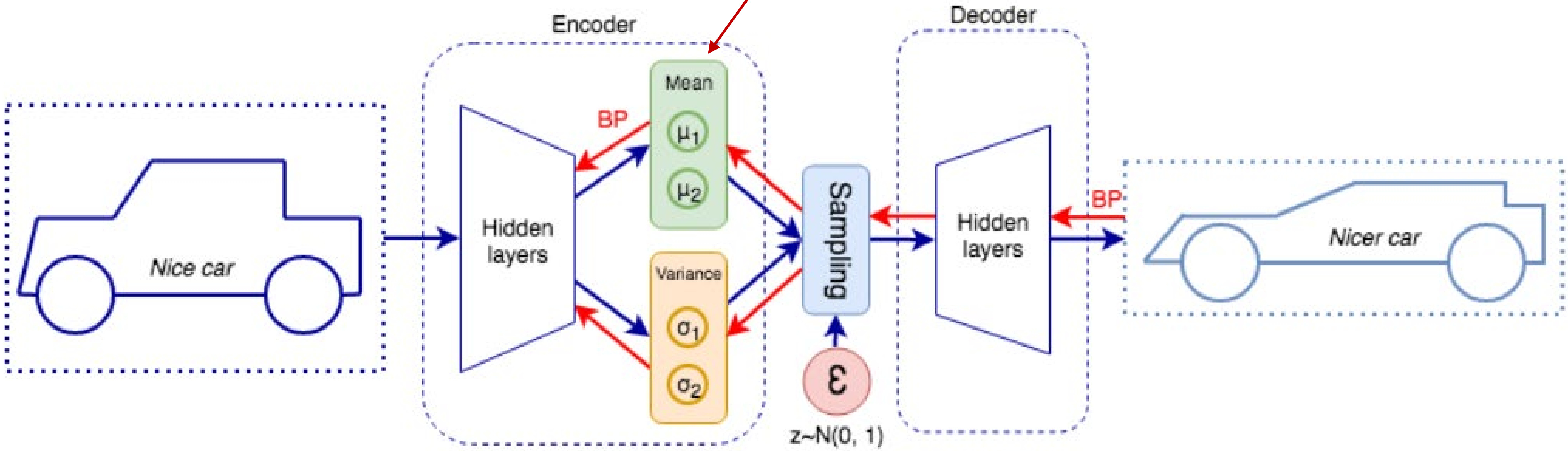
KL Divergence: measures how close the two distribution are.
➔ We assume p(z) follow N(0, 1), thus by minimizing this loss we try to get q(z|x) (i.e. the generated feature) as close as possible to a normal distribution of N(0, 1)

Reconstruction Loss: measures the difference between original input and its reconstruction

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Implement VAE

Bottleneck layer won't directly output latent features but output two vectors, which describe the mean & variance of latent features



Through training, the generated latent features converges to a Normal distribution of N(0, 1)

Latent sample z is generated using mean & variance vector

$$\mathbf{z} = \mu + \sigma \odot \varepsilon$$

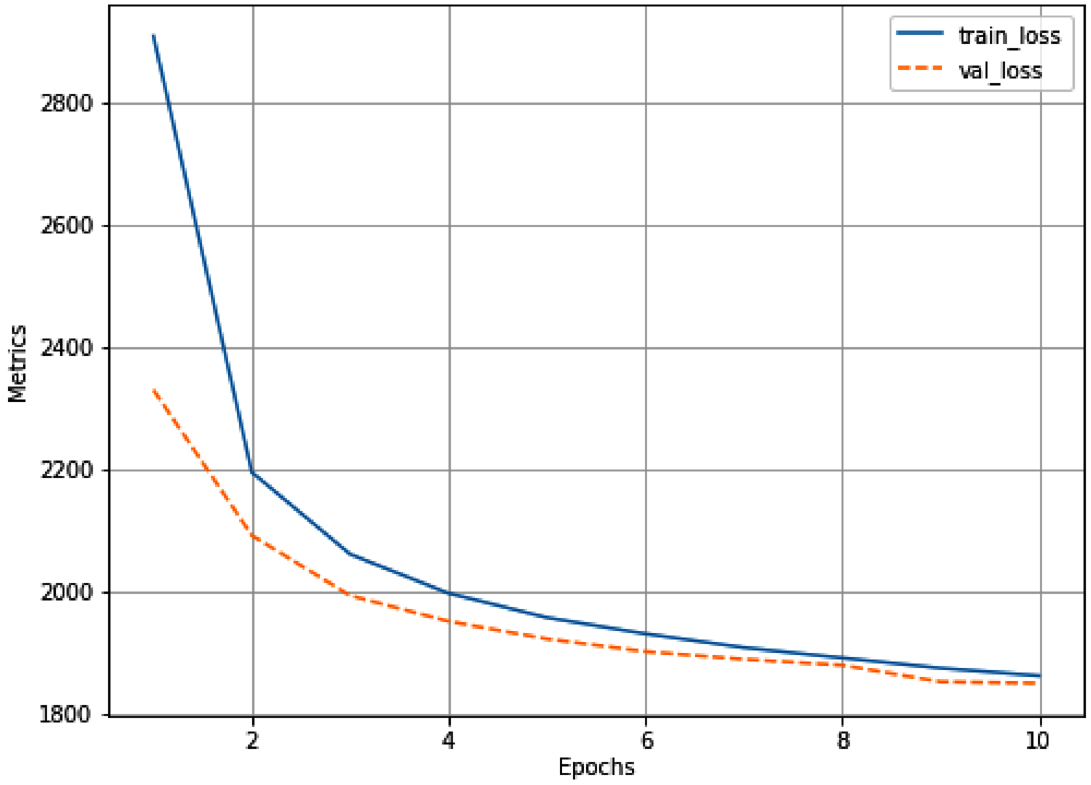
latent mean

latent variance

$\epsilon \sim \mathcal{N}(0, I)$

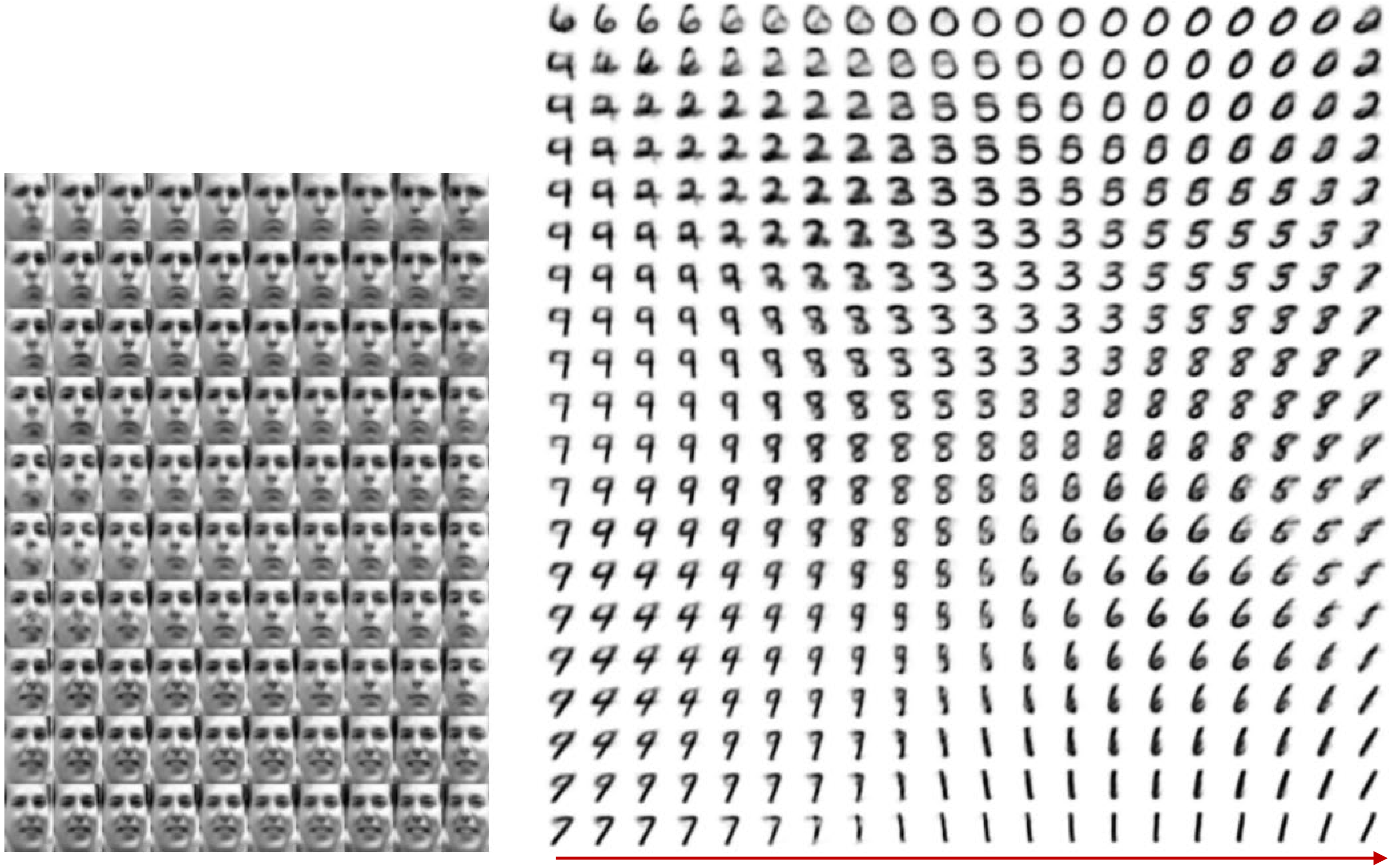
a random noise N(0, 1) can generate a realistic image through decoder

Lab 3: Implement a VAE



VAE Performance

Change a specific variable in latent vector z



change z2

change z1

(a) Learned Frey Face manifold

(b) Learned MNIST manifold

Age Progression/Regression by Conditional Adversarial Autoencoder (CAAE)

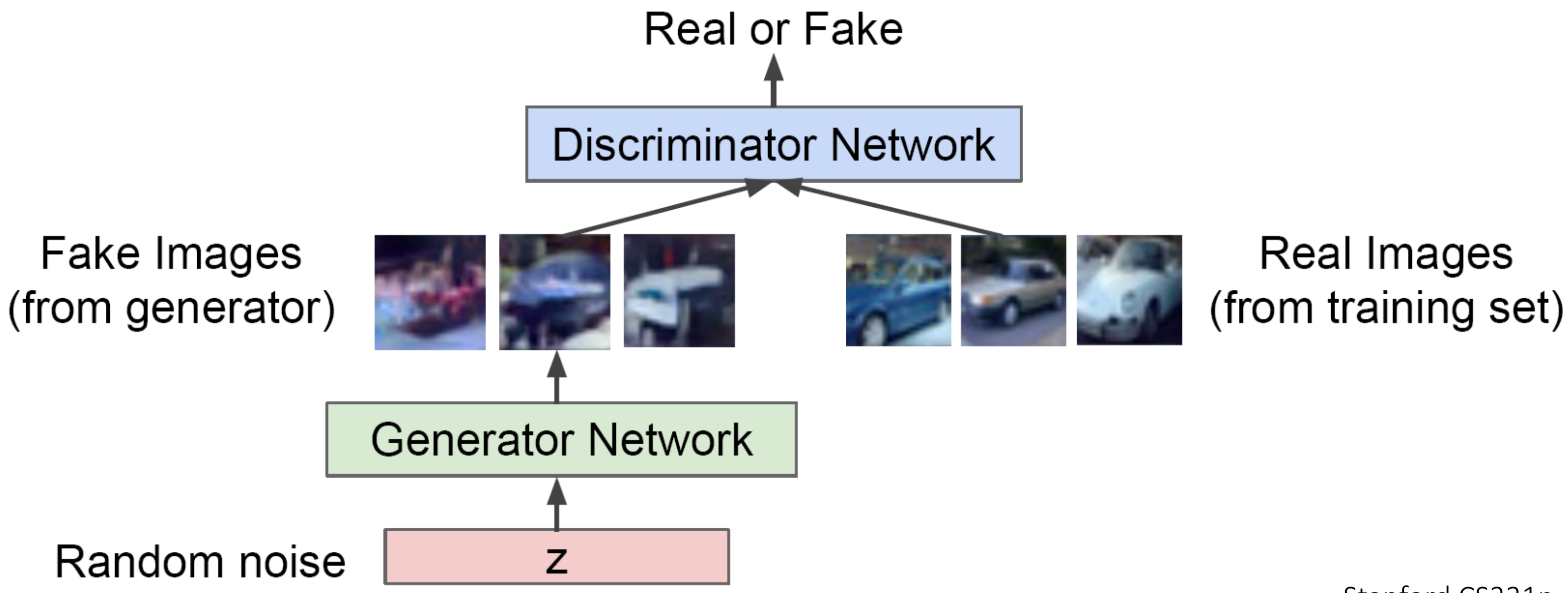
Reconstructed images

Testing samples used to generate the age ascending images

Generative Adversarial Network (GAN)

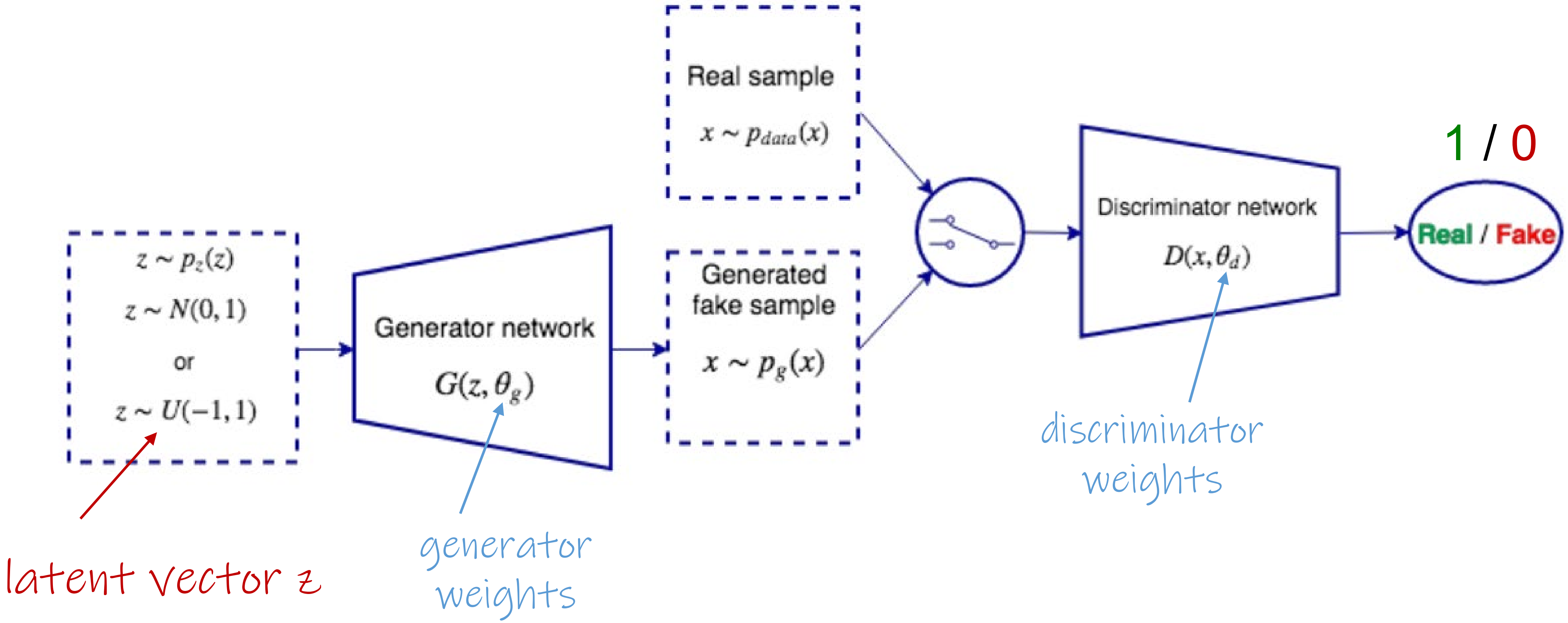
Introduction

- Give up on probability modelling
- Just want ability to sample
  - Sample from a random noise to generate an image
  - Use a discriminator network to tell whether the generated image is within data distribution ("real") or not ("fake")



Training GAN

- Discriminator tries to differentiate Real & Fake images (Binary Classification)
- Generator tries to deceive discriminator by generating close to real image



Discriminator

- A binary classification model: using binary cross-entropy as loss function

$$H(p, q) = -(p(\mathbf{x}) \log q(\mathbf{x}) + (1 - p(\mathbf{x})) \log(1 - q(\mathbf{x})))$$

target probability
{0, 1}

model predicted
probability (0<q<1)

- For a mini batch of m samples:

$$H(p, q) = -\frac{1}{m} \sum_{j=1}^{m} (p(\mathbf{x}_j) \log(q(\mathbf{x}_j)) + (1 - p(\mathbf{x}_j)) \log(1 - q(\mathbf{x}_j)))$$

Discriminator

- Discriminator Loss

Cumulative class probability: half real image and half fake image

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})}\log(D(\mathbf{x})) - \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}\log(1 - D(G(\mathbf{z})))$$

Real Image
p = 1

Fake Image
p = 0

- Goal: Minimize $J^{(D)}$ or Maximize negative $J^{(D)}$

Generator

- Generator Loss: Only has Fake Image, i.e. p = 0

$$J^{(G)} = \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})} \log(1 - D(G(\mathbf{z})))$$

- Goal: Minimize $J^{(G)}$ or Maximize $\boxed{\text{negative } J^{(G)}}$

$$\boxed{-\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})} \log(1 - D(G(\mathbf{z})))}$$

Classification Loss for Fake Images (p =0)

Discriminator tries to minimize the classification loss, but generator tries to maximize the classification loss

Putting it all together

- Minmax Objective

Negative of Classification Loss

$$\min_G \max_D V(G, D) = \boxed{\frac{1}{2}\mathbb{E}_{\mathbf{x}\sim p_{data}(\mathbf{x})} \log(D(\mathbf{x})) + \frac{1}{2}\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})} \log(1 - D(G(\mathbf{z})))}$$

- Iterative Training
  1. Freeze Generator, Train the Discriminator to maximize V(G, D)
  2. Freeze Discriminator, Train the Generator to minimize V(G, D)

➔ Converge to Nash Equilibrium, i.e. Generator is so good that Discriminator is no longer able to distinguish between real and fake images, i.e. will always output 0.5.

GAN Training Algorithm

for number of training iterations **do**

for k steps **do**

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

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D_{\theta_d}(x^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(z^{(i)}))) \right]$$

end for

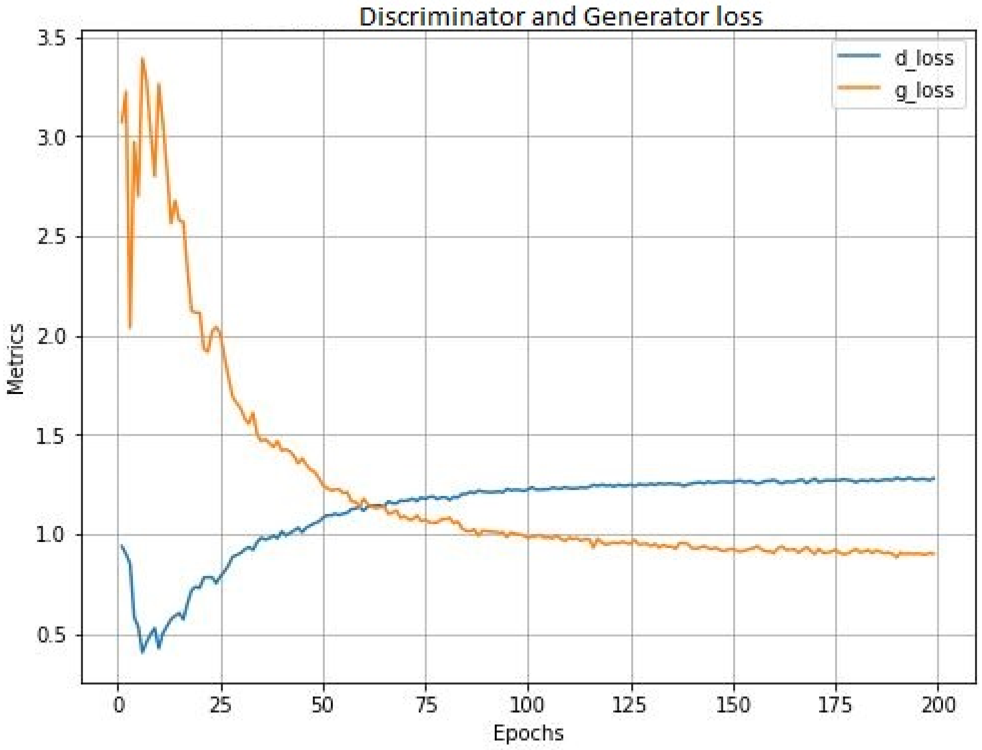
- Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \dots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log(D_{\theta_d}(G_{\theta_g}(z^{(i)})))$$

end for

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Lab 4: Using GANs to generate handwritten digits



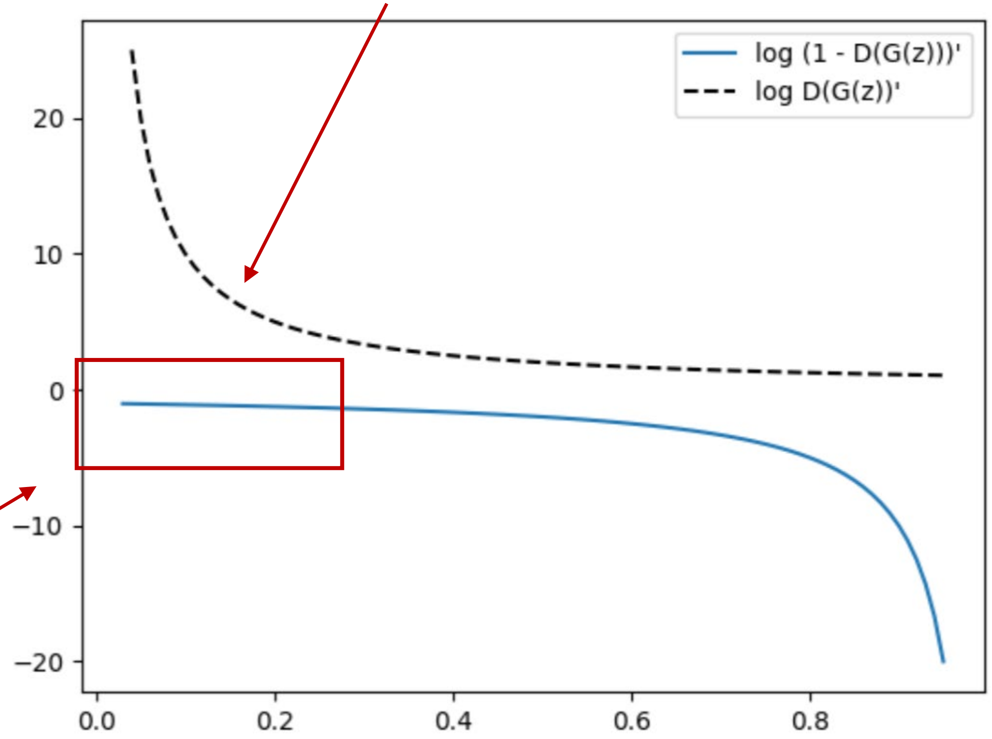
Problems with training GAN

- Generator Loss (minimize)

$$J^{(G)} = \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})} \log(1 - D(G(\mathbf{z})))$$

$$J^{(G)} = -\mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})} \log(D(G(\mathbf{z})))$$

Solution: use a different loss function



Initially discriminator can easily distinguish between real & fake images ➔ D(G(z)) close 0 ➔ the gradient is also close to zero➔ difficult to train Generator weights

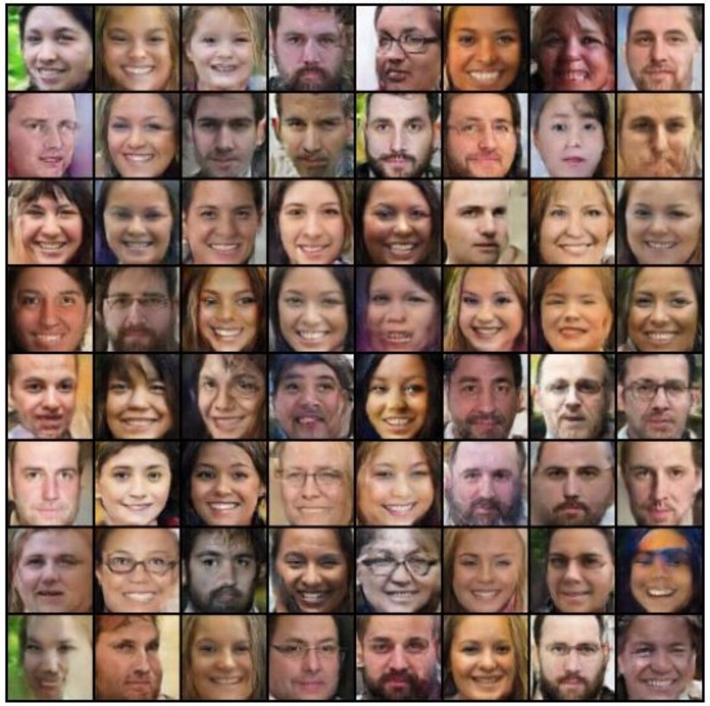
Problems with training GAN

- Gradient descent algorithm is designed to find minimum point of loss function, instead of Nash Equilibrium ➔ training may fail to converge but oscillate
- The discriminator cannot be too good or too bad for the training to success
  - Too good ➔ zero error gradient ➔ prevent generator learning anything
  - Too bad ➔ backpropagate the wrong information to generator
- Mode Collapse
  - The generator generates very similar samples
  - Low diversity

Improving GAN

- Normalize the inputs
  - Normalize the images between -1 and 1
  - Tanh as the last layer of the generator output
- Sample z from a gaussian distribution instead of uniform distribution
- Use Dropouts in G in both train and test phase
- Use Soft and Noisy Labels
  - Real Image label: [0.7, 0.9]; Fake Image label: [0, 0.3]
  - Occasionally flip the labels when training the discriminator

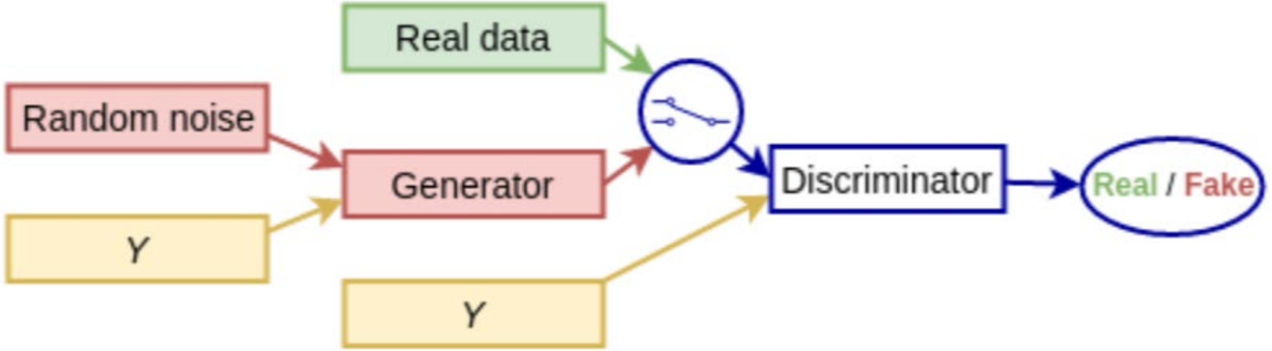
Types of GAN



- DCGAN (Deep Convolutional GAN)
  - Use batch norm for most layers of D and G
    - except last layer of G and first layer of D
  - For D, use strided convolutions instead of pooling layers
  - For G, use transpose convolutions to upsample the latent vector to the generated image
  - No fully-connected layers with the exception of the last layer of D
  - LeakyRelu activations for all the layers (except the output layer) of D and G
    - Output of G: Tanh
    - Output of D: Logistic
  - Use Adam

Slide credit: Wang Wei

Types of GAN

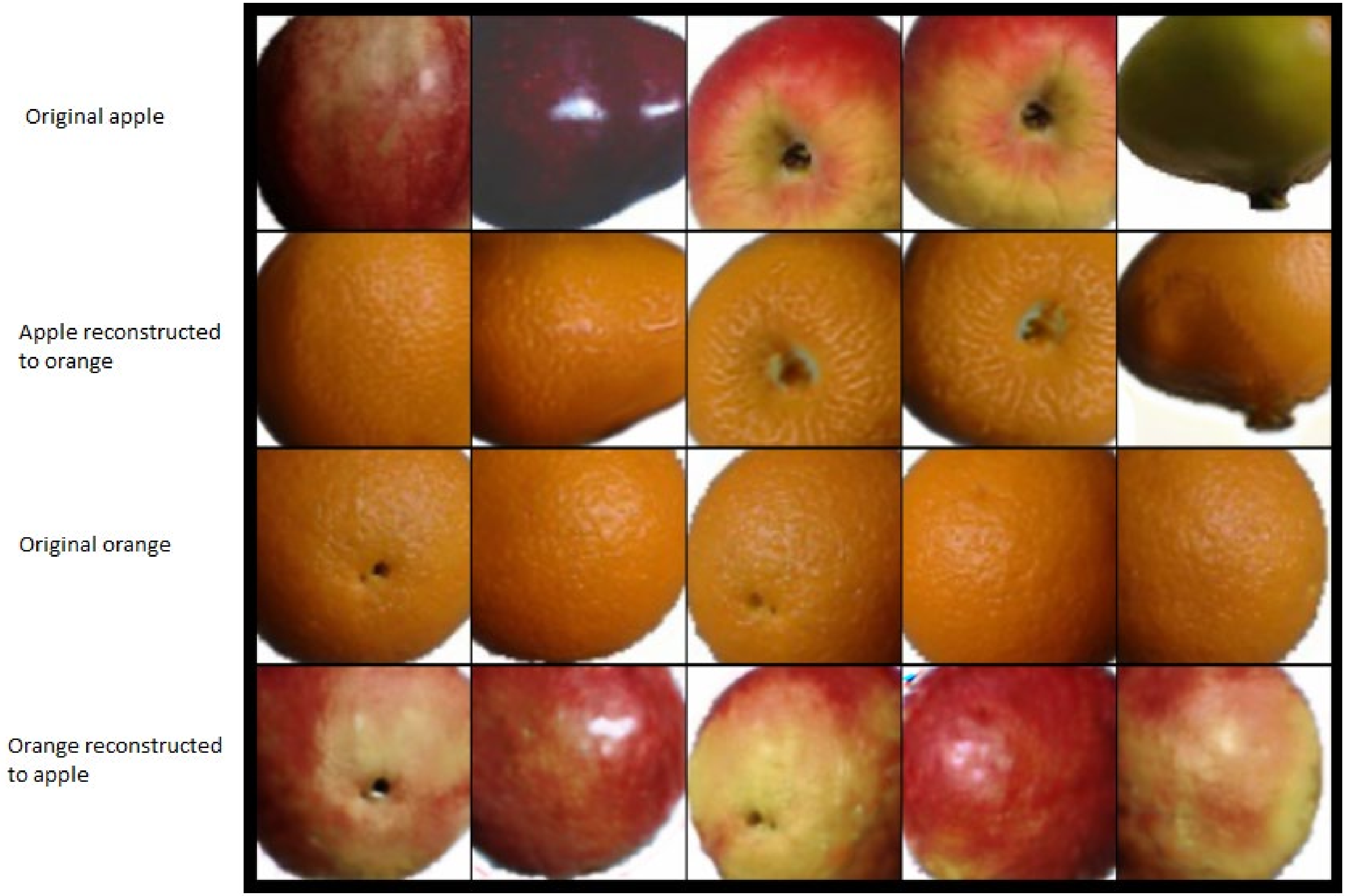


- Conditional CGAN

CGAN for conditional labels 3, 8, and 9

Types of GAN

- CycleGAN
- image translation



Types of GAN

- StyleGAN
- Generate high quality images

Picture: *These people are not real – they were produced by our generator that allows control over different aspects of the image.*

Reference

- VAE Paper https://arxiv.org/abs/1312.6114
- GAN Paper https://arxiv.org/abs/1406.2661
- Ivan Vasilev "Advanced Deep Learning with Python"
- Some slides are taken from Stanford Course CS231n