VAEs and GANs

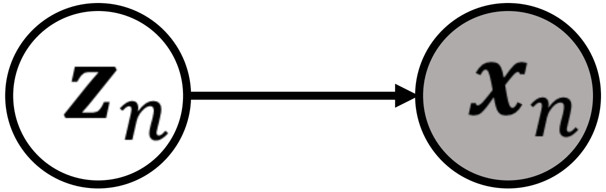
CS771: Introduction to Machine Learning

Nisheeth

Generative Models with Latent Variables

- We have already looked at latent variable models in this class
- Used for
  - Clustering
  - Dimensionality reduction

Latent variable z_n usually encodes some latent properties of the observation $\boldsymbol{x}_n$



- Broadly, latent variable models approximate the distribution on X

$$p(X) \approx \sum_z p(X|z;\theta)p(z)$$

- Can apply this approximation in a variety of applications
  - Such as generation of new examples

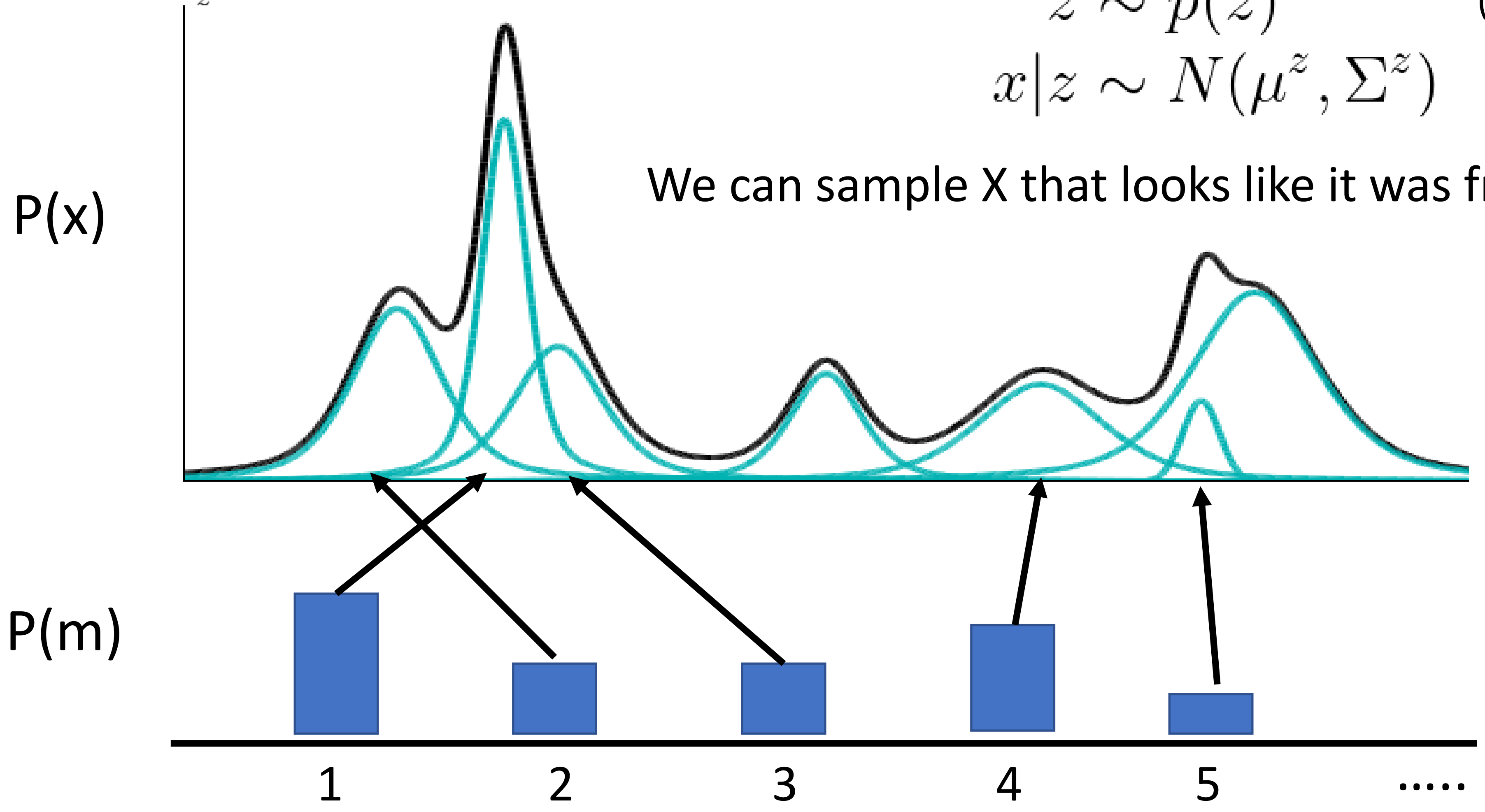
Example: Gaussian Mixture Model

$$p(X) \approx \sum_z p(X|z;\theta)p(z)$$

$$z \sim p(z) \qquad \text{(categorical)}$$

$$x|z \sim N(\mu^z, \Sigma^z)$$

We can sample X that looks like it was from p(X)



A general principle of generation

- Data is encoded into a different representation
- New data is generated by sampling from the new representation
- GMMs are just one type of encoding-decoding scheme

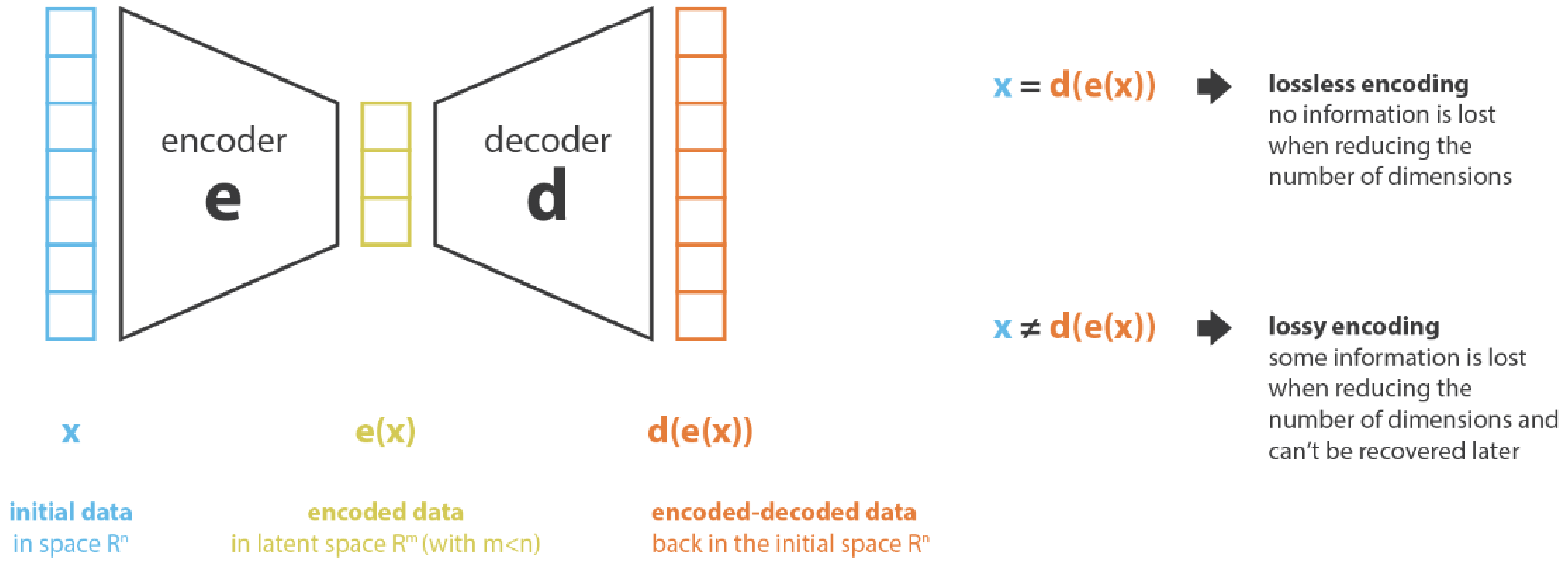
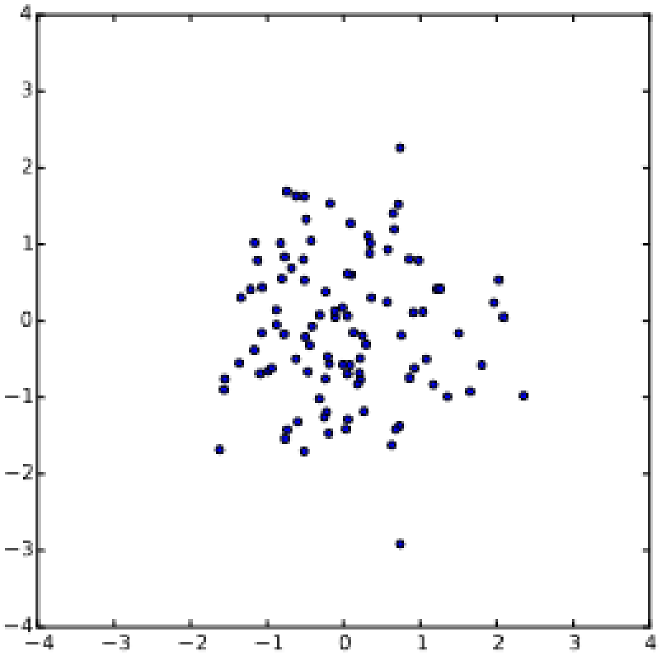


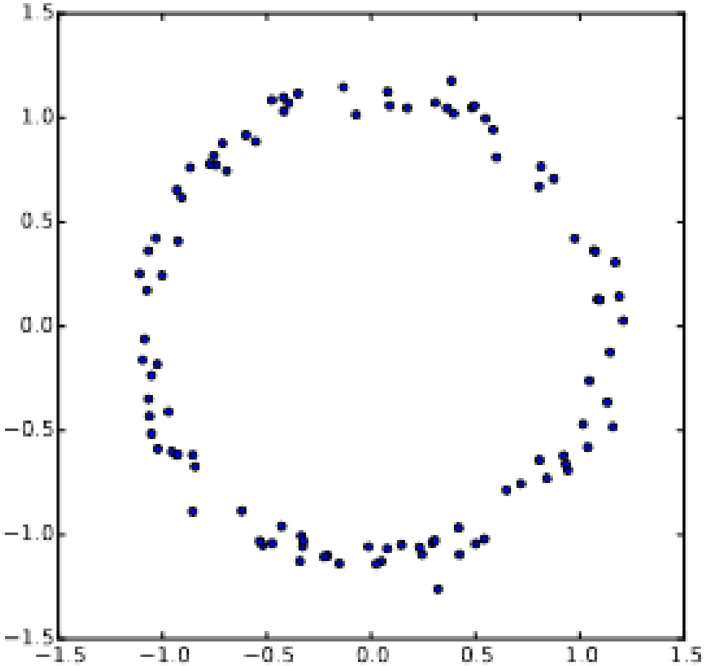
Image credit (link)

Creating flexible encoders



Actually,

$$y = g(x) = \frac{x}{10} + \frac{x}{\|x\|}$$



Easy to **encode** this data distribution of a random variable X with a bivariate Gaussian

What about the data distribution of this random variable Y?

Variational auto-encoders: the basic premise

- Any distribution in d dimensions can be generated by taking a set of d normally distributed random variables and mapping them through a sufficiently complex function
- We use a neural network encoder to learn this function from data

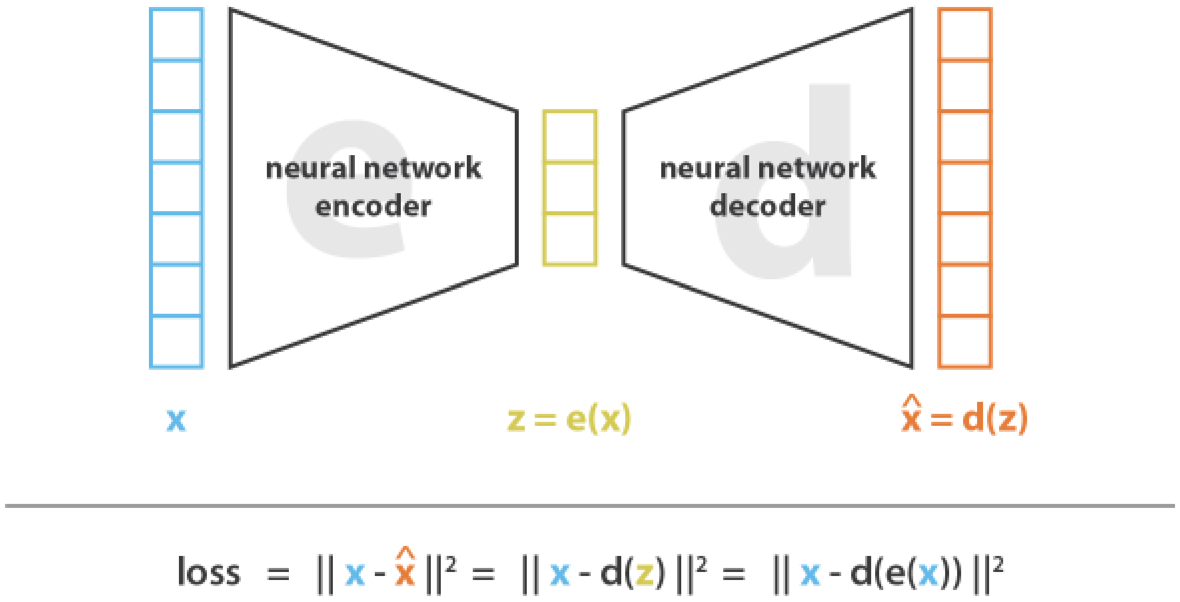


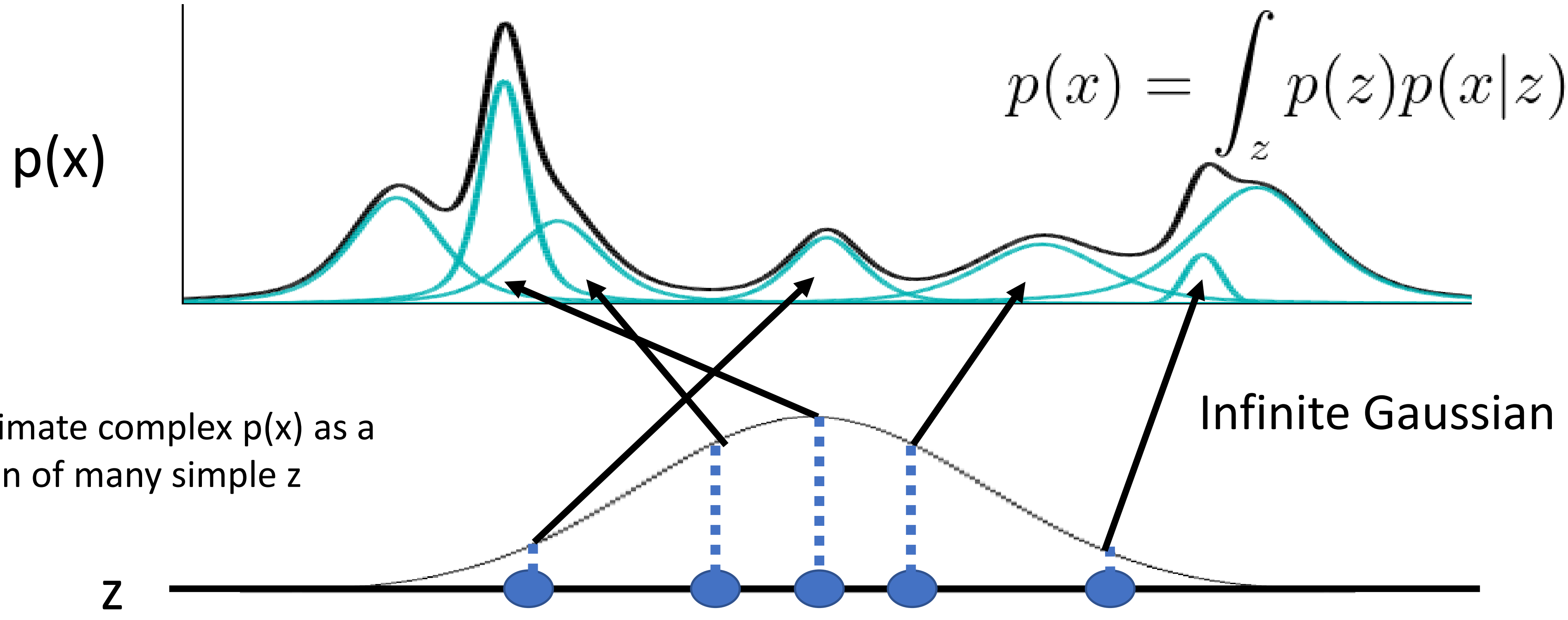
Image credit (link)

VAEs concept

Each dimension of z represents a data attribute

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

$$p(x) = \int_z p(z)p(x|z)dz$$



We approximate complex p(x) as a composition of many simple z

VAEs in practice

- Brute force approximation of P(X)
  - Sample a large number of z values
  - Compute $P(X) \approx \frac{1}{n} \sum_i P(X|z_i)$
- Problem, when z is high dimensional, you'd need a very large n to sample properly
- VAEs try to sample p(X|z) efficiently
  - Key idea: the X → z mapping is sparse in a large enough neural network
  - Corollary: most p(X|z) will be zero
- Rather than directly sample P(X|z), we try and estimate Q(z|x) that gives us the z that are most strongly connected with any given x
  - VAEs assume Q are Gaussian

The VAE objective function

- We want to minimize

$$\mathcal{D}\left[Q(z)\|P(z|X)\right] = E_{z\sim Q}\left[\log Q(z) - \log P(z|X)\right].$$

- Which is equivalent to maximizing

$$\log P(X) - \mathcal{D}\left[Q(z)\|P(z|X)\right] = E_{z\sim Q}\left[\log P(X|z)\right] - \mathcal{D}\left[Q(z)\|P(z)\right].$$

- VAE assumes that we can define some Q(z|X) that maximizes

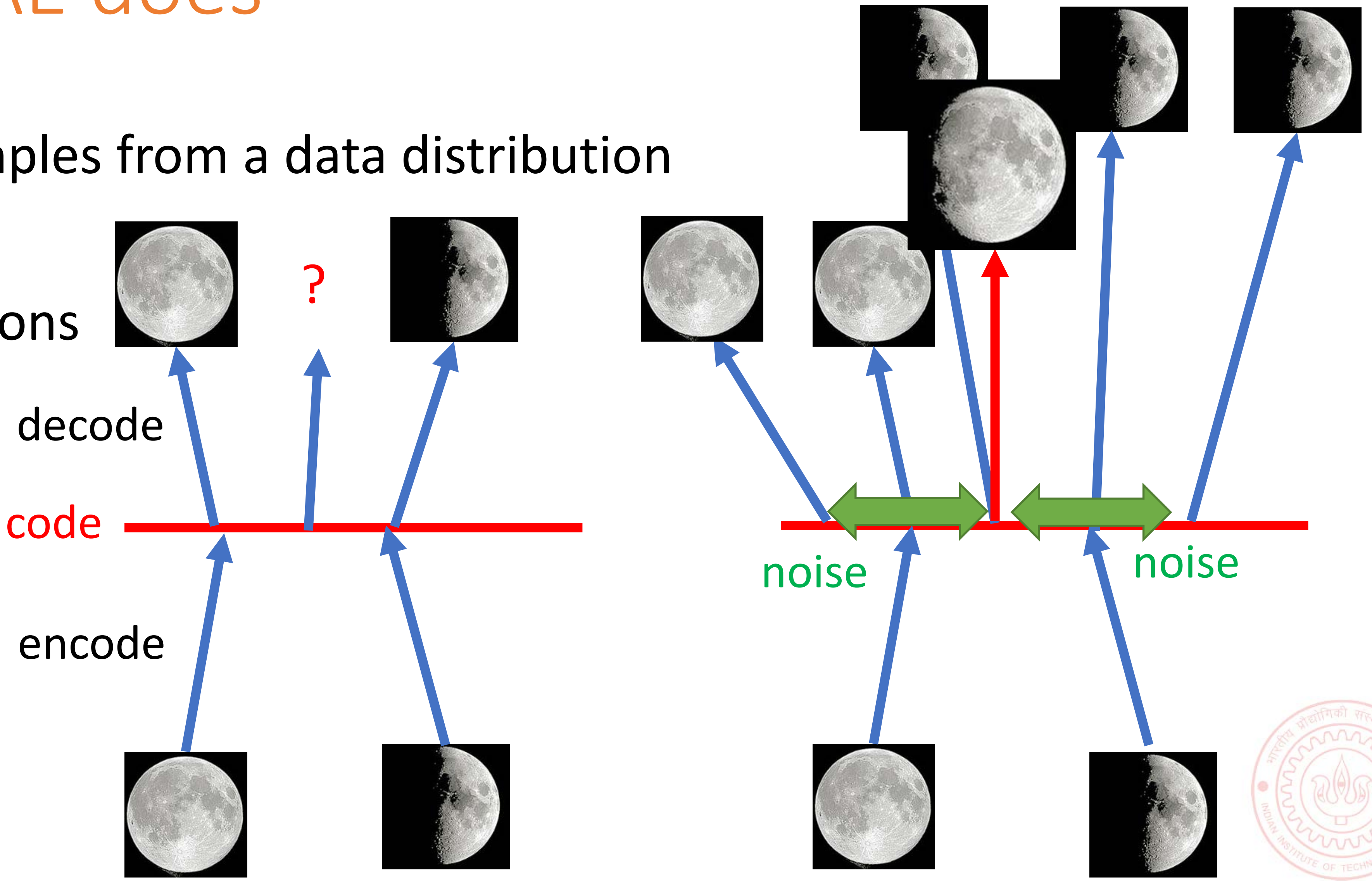
$$\log P(X) - \mathcal{D}\left[Q(z|X)\|P(z|X)\right] = E_{z\sim Q}\left[\log P(X|z)\right] - \mathcal{D}\left[Q(z|X)\|P(z)\right],$$

- The RHS is maximized using stochastic gradient descent, sampling a single value of X and z from Q(z|X) and then calculating the gradient of $\log P(X|z) - \mathcal{D}\left[Q(z|X)\|P(z)\right].$

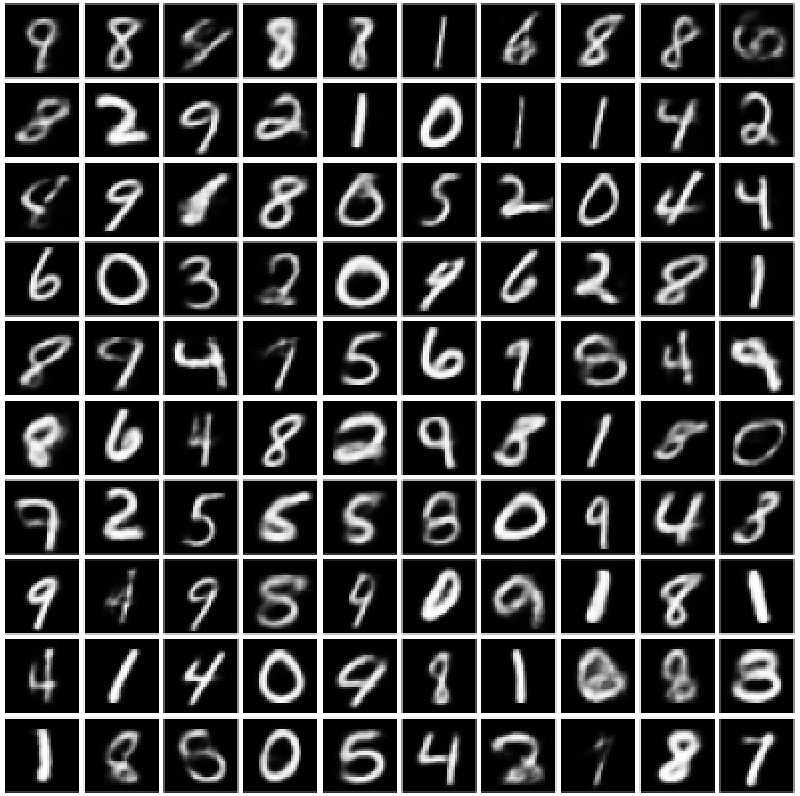
See here for derivations and a more detailed explanation

What a VAE does

- Generate samples from a data distribution
- For any data
- Cool applications



VAE outputs

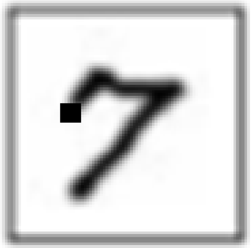


Samples from a VAE trained on MNIST

Samples from a VAE trained on a faces dataset

VAE limitations

- People have mostly moved on from VAEs to use GANs for generating synthetic high-dimensional data
- VAEs are theoretically complex
- Don’t generalize very well
- Are pragmatically under-constrained
  - Reconstruction error need not be exactly correlated with realism



Realistic

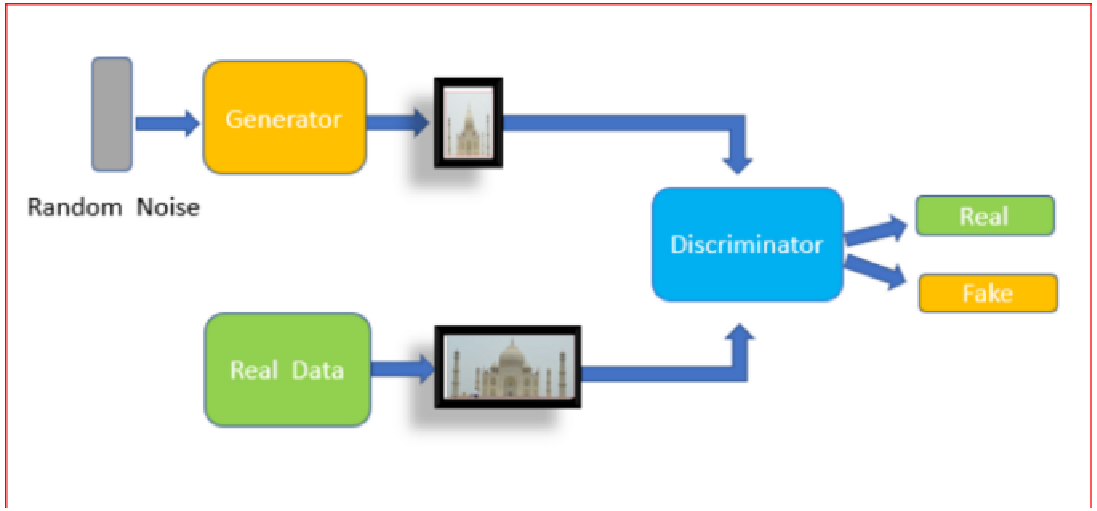
Fake

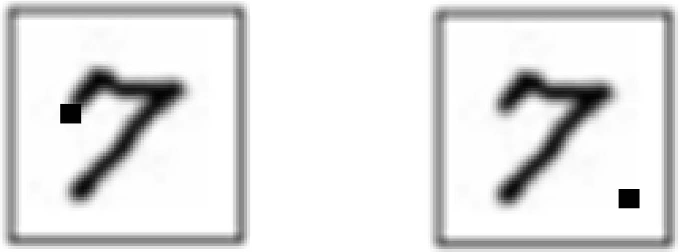
Generative adversarial networks (GANs)

- VAEs approximate P(X) using latent variables z, with the mapping between X and z pushed through a NN function approximation that ensures that the transformed data can be well represented by a mixture of Gaussians
- But approximating P(X) directly is complicated, and approximating it well in the space of an arbitrarily defined reconstruction error does not generalize well in practice
- GANs go about approximating P(X) using an indirect approach

Adversarial training

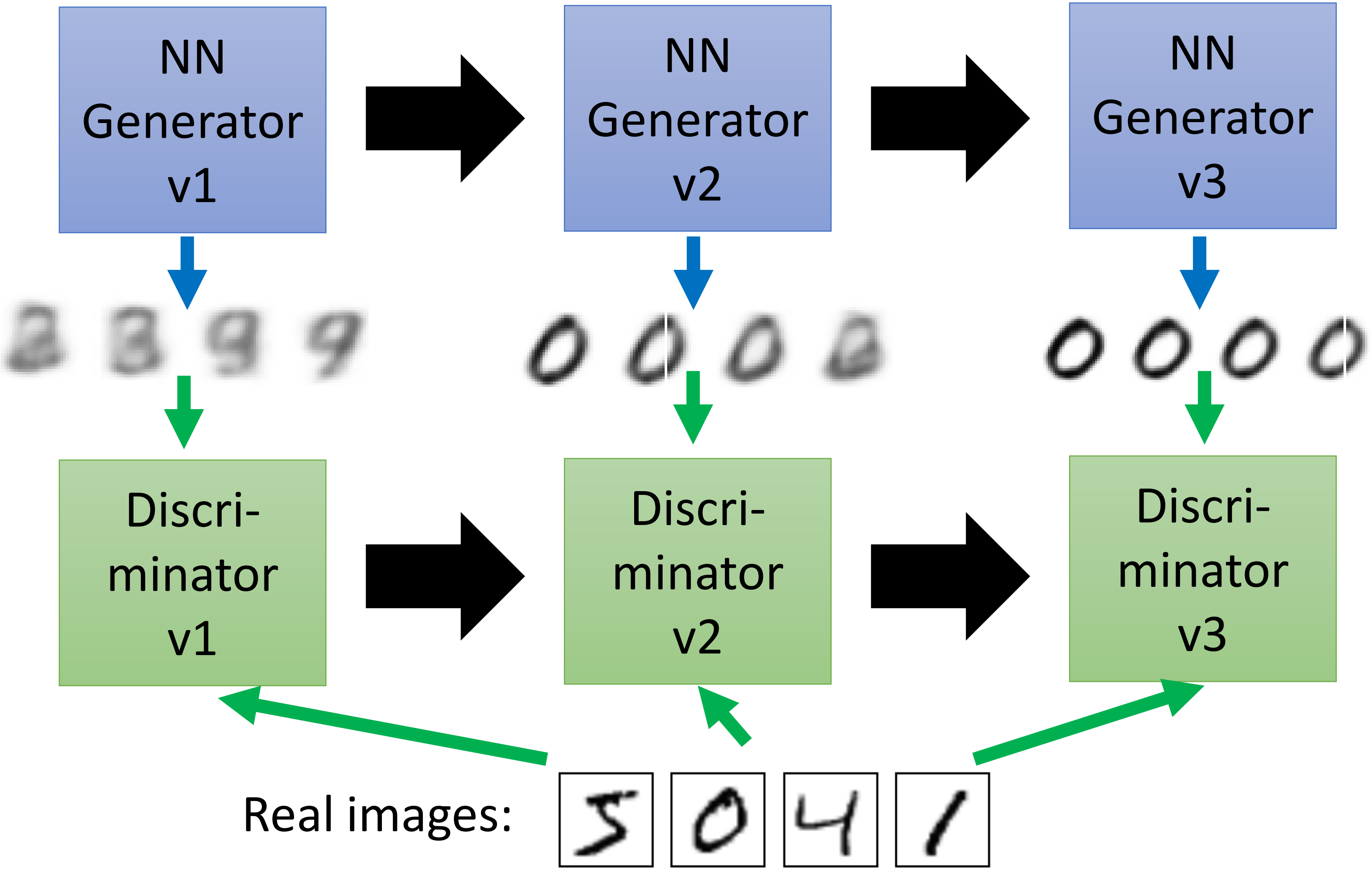
- Two models are trained – a generator and a discriminator
- The goal of the discriminator is to correctly judge whether the data it is seeing is real, or synthetic
    - Objective function is to maximize classification error
- The goal of the generator is to fool the discriminator
    - It does this by creating samples as close to real data as possible
    - Objectively tries to minimize classification error
- No longer reliant on reconstruction error for quality assessment



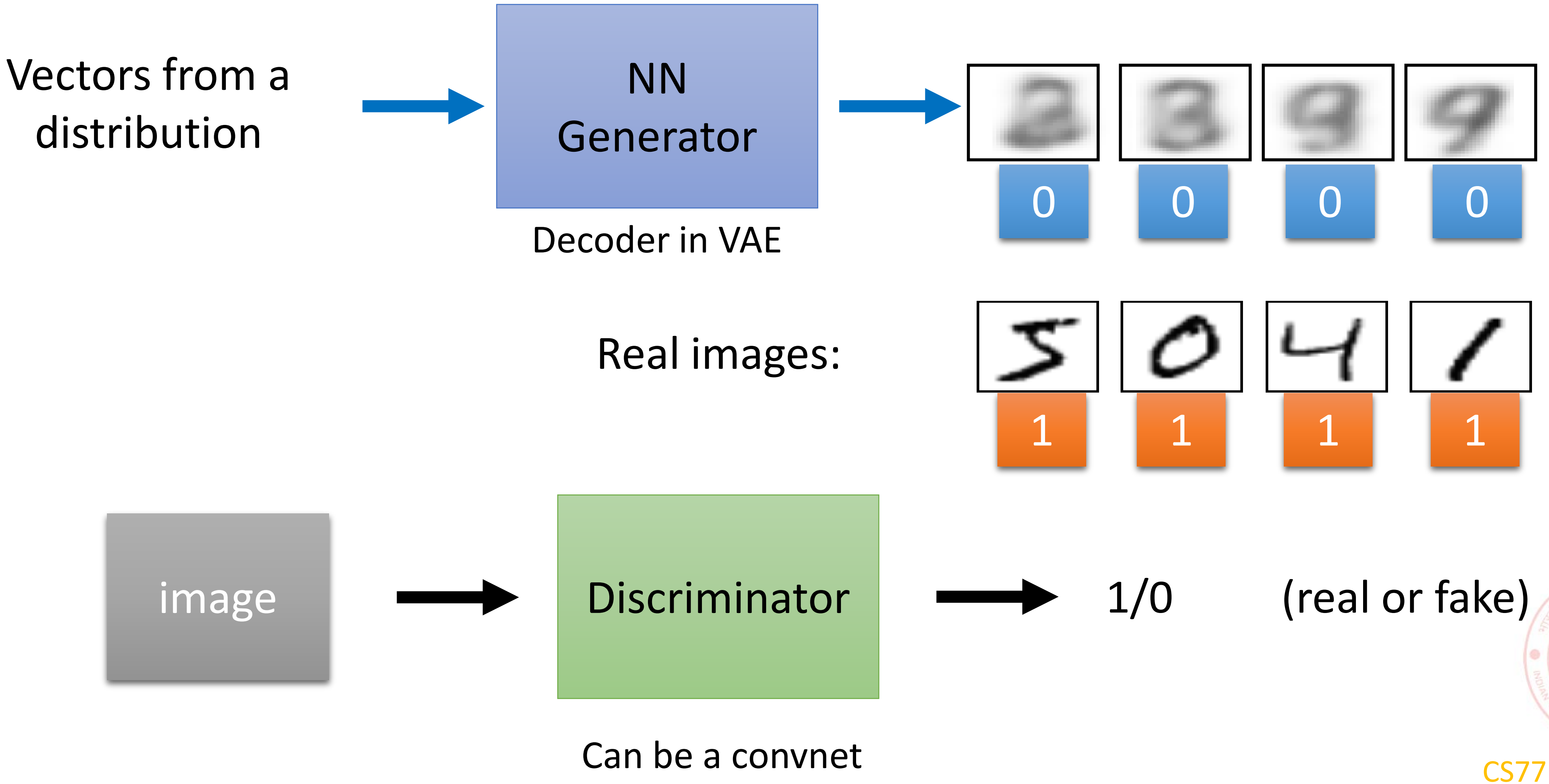


Realistic Fake

GANs


NN
Generator
v1
NN
Generator
v2
NN
Generator
v3
Discri-
minator
v1
Discri-
minator
v2
Discri-
minator
v3
Real images:

GAN - Discriminator



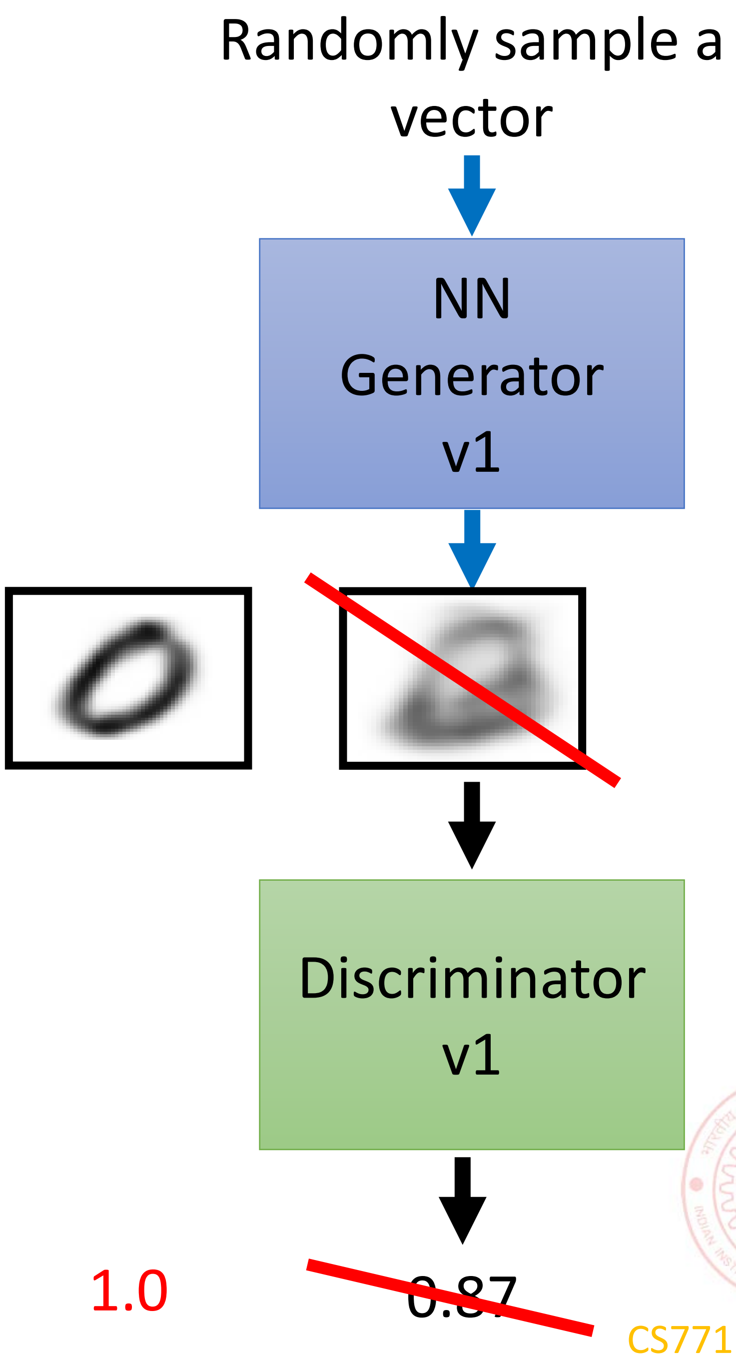
GAN - Generator

“Tuning” the parameters of generator

The output be classified as “real” (as close to 1 as possible)

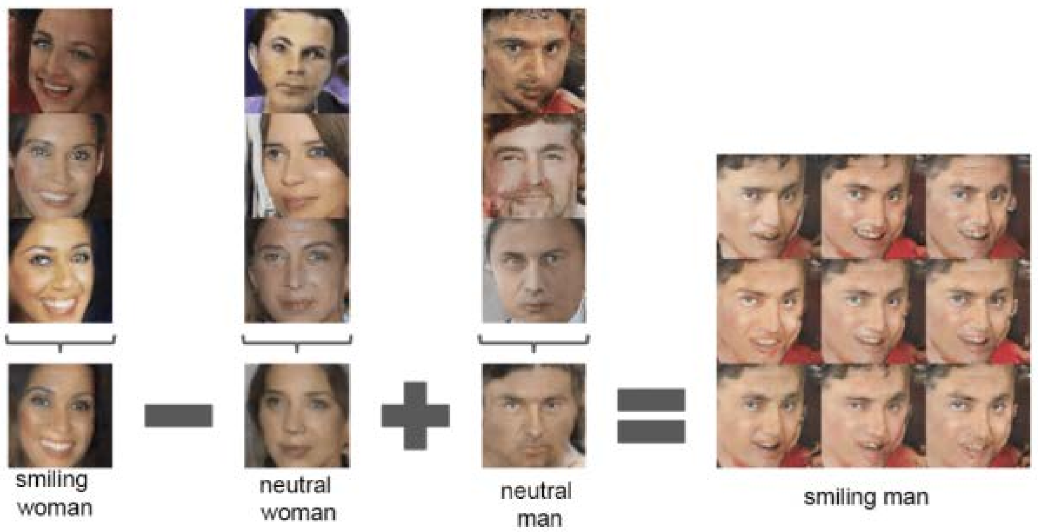
Generator + Discriminator = a network

Use gradient descent to find the parameters of generator



GAN outputs

- The latent space learned in GANs is very interesting
- People have showed that vector additions and subtractions are meaningful in this space
- Can control novel item compositions almost at will
- A big ‘deepfakes’ industry is growing up around this



Example of Vector Arithmetic on Points in the Latent Space for Generating Faces With a GAN.
Taken from Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks.

For more details, see here

Summary

- In CS771, you have learned the basic elements of ML
- Representing data as multidimensional numerical representations
- Defining model classes based on different mathematical perspectives on data
- Estimating model parameters in a variety of ways
- Defining learning objectives mathematically, and optimizing them
- Evaluating outcomes, to some degree
- What will you do with this knowledge?