

# Unsupervised Deep Learning

**LEADINGINDIA.AI**  
NATIONWIDE AI SKILLING & RESEARCH INITIATIVE

# Session Coverage

---

Unsupervised Learning

---

Sparse Coding

---

Restricted Boltzmann Machines

---

Deep Belief Network

---

Autoencoder

---

Variants of Autoencoders

---

Deep Autoencoder

---

Discriminative vs Generative Learning

---

Variational Autoencoder

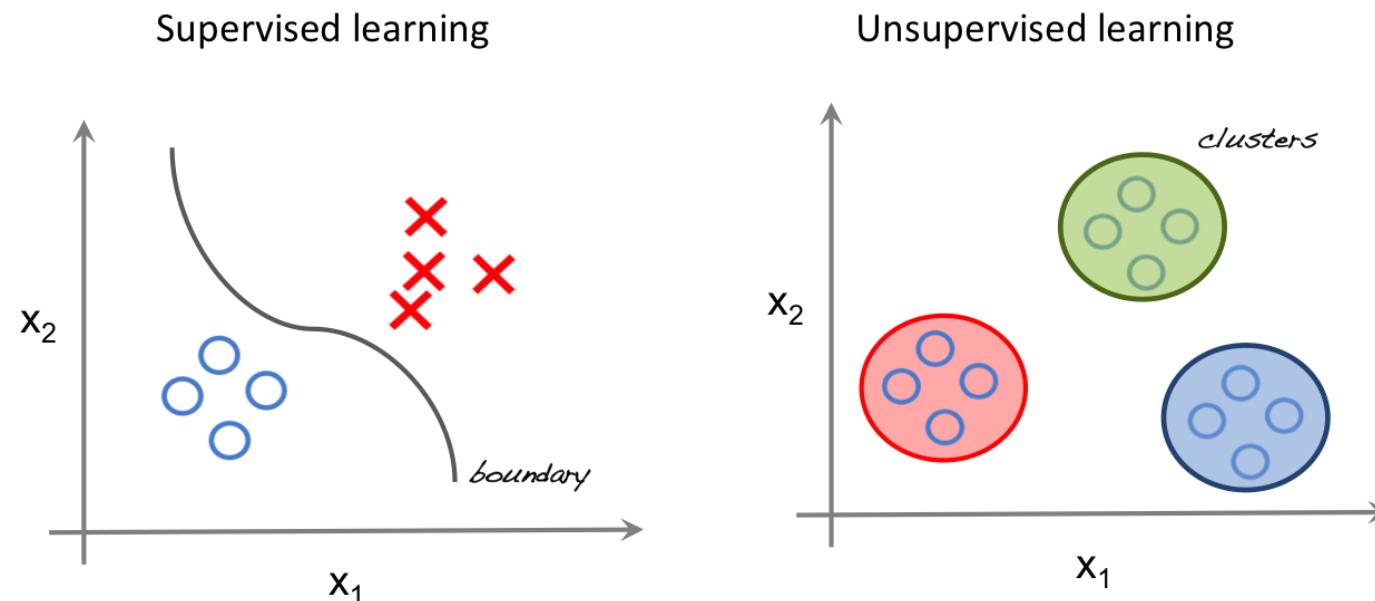
---

Generative Adversarial Networks

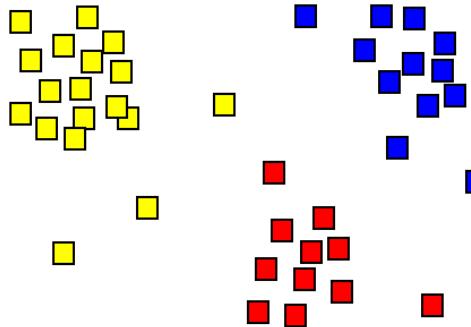
---

# Unsupervised Learning

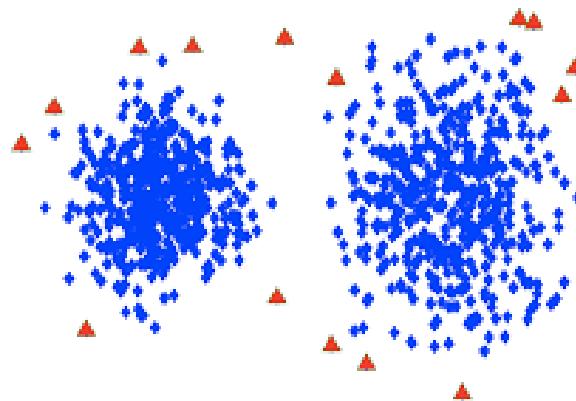
- Used when there is **input data (X)** and **no** corresponding **output label (Y)**
- Objective is to explore the data and find some pattern within
- Very useful when there is **lot of data** but **few labels**
- Example tasks such as association rule, clustering, compression, dimensionality reduction, data generation



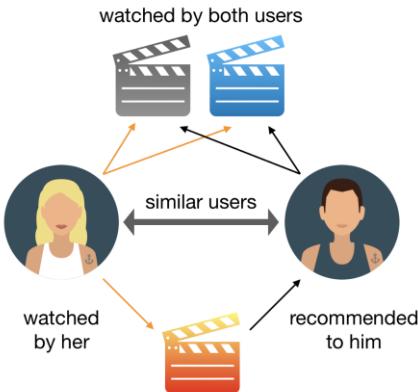
# Applications of Unsupervised Learning



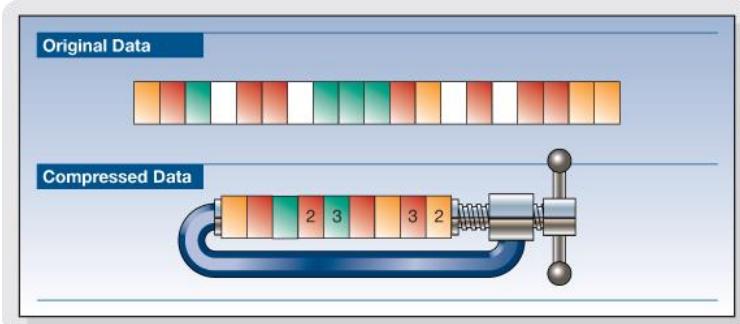
Clustering



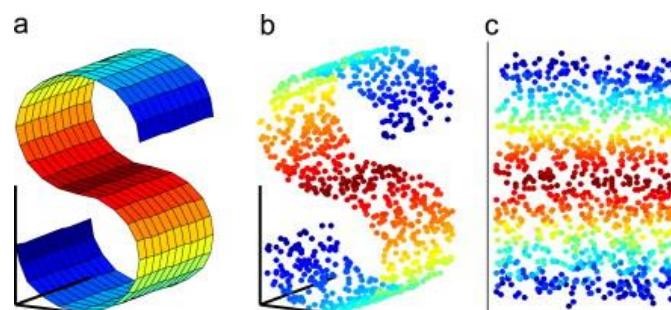
Anomaly Detection



Recommender System



Data Compression

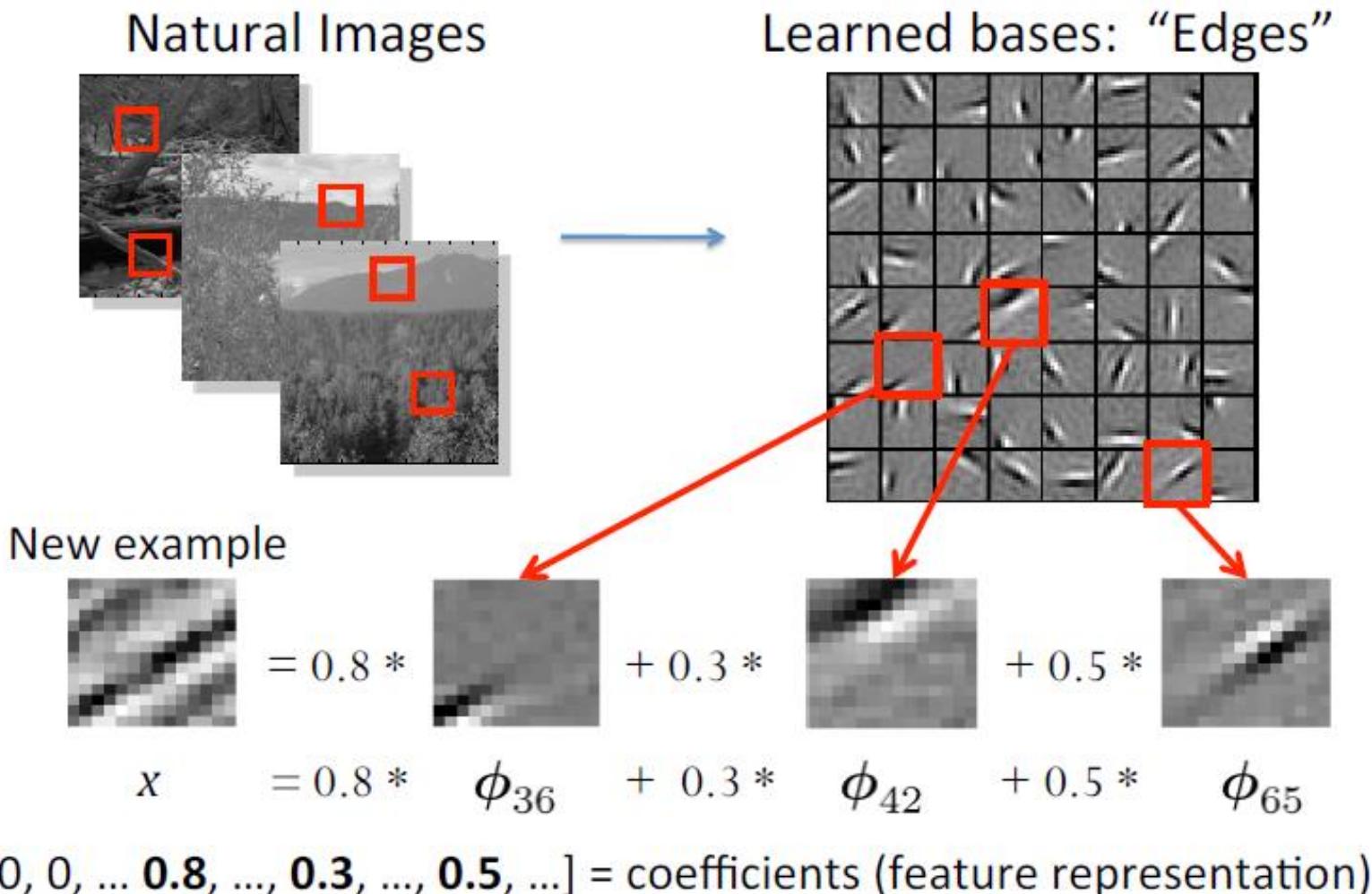


Dimensionality Reduction



Data Generation

# Sparse Coding



# Sparse Coding

- Sparse coding (Olshausen & Field, 1996). Originally developed to explain early visual processing in the brain (edge detection).
- **Objective:** Given a set of input data vectors  $\{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N\}$ , learn a dictionary of bases  $\{\phi_1, \phi_2, \dots, \phi_K\}$ , such that:

$$\mathbf{x}_n = \sum_{k=1}^K a_{nk} \phi_k,$$

Sparse: mostly zeros



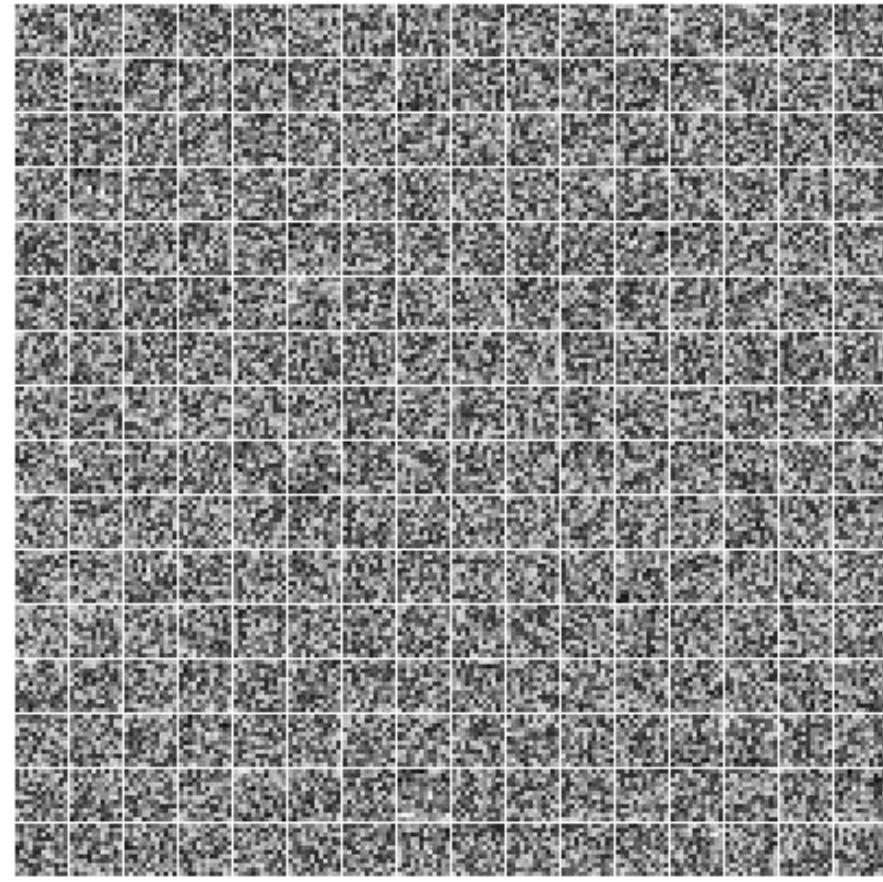
- Each data vector is represented as a sparse linear combination of bases.

# Sparse Coding Training

- Input image patches:  $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N \in \mathbb{R}^D$
- Learn dictionary of bases:  $\phi_1, \phi_2, \dots, \phi_K \in \mathbb{R}^D$

$$\min_{\mathbf{a}, \phi} \sum_{n=1}^N \left\| \mathbf{x}_n - \sum_{k=1}^K a_{nk} \phi_k \right\|_2^2 + \lambda \sum_{n=1}^N \sum_{k=1}^K |a_{nk}|$$

Reconstruction error Sparsity penalty



- Alternating Optimization:
  1. Fix dictionary of bases  $\phi_1, \phi_2, \dots, \phi_K$  and solve for activations  $\mathbf{a}$  (a standard Lasso problem).
  2. Fix activations  $\mathbf{a}$ , optimize the dictionary of bases (convex QP problem).

# Sparse Coding Testing

- Input: a new image patch  $x^*$ , and K learned bases  $\phi_1, \phi_2, \dots, \phi_K$
- Output: sparse representation  $\mathbf{a}$  of an image patch  $x^*$ .

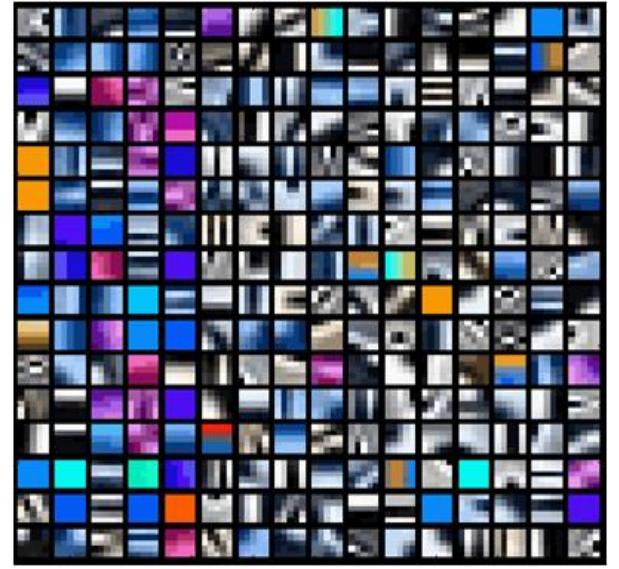
$$\min_{\mathbf{a}} \left\| \mathbf{x}^* - \sum_{k=1}^K a_k \boldsymbol{\phi}_k \right\|_2^2 + \lambda \sum_{k=1}^K |a_k|$$

$$\begin{array}{ccccccccc} \text{Image Patch } x^* & = & 0.8 * & \text{Image } \phi_{36} & + & 0.3 * & \text{Image } \phi_{42} & + & 0.5 * & \text{Image } \phi_{65} \\ x^* & = & 0.8 * & \phi_{36} & + & 0.3 * & \phi_{42} & + & 0.5 * & \phi_{65} \end{array}$$

[0, 0, ... **0.8**, ..., **0.3**, ..., **0.5**, ...] = coefficients (feature representation)

# SC for Image Quality Enhancement

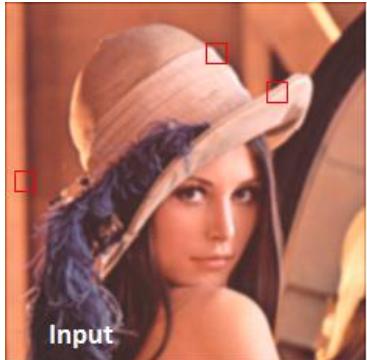
- **Input:** Corrupted image patch
- **Output:** HQ Reconstructed image patch



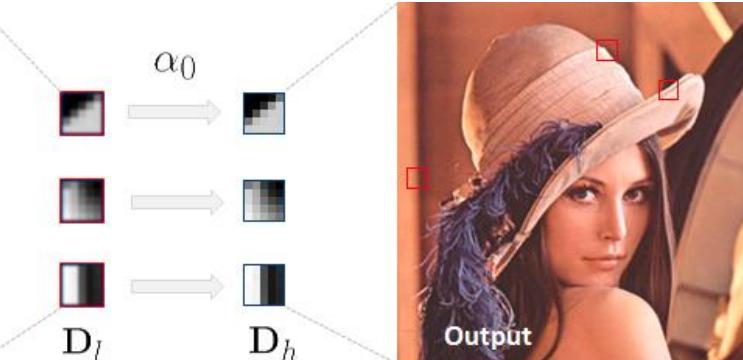
Dictionary Learned



Inpainting



Input



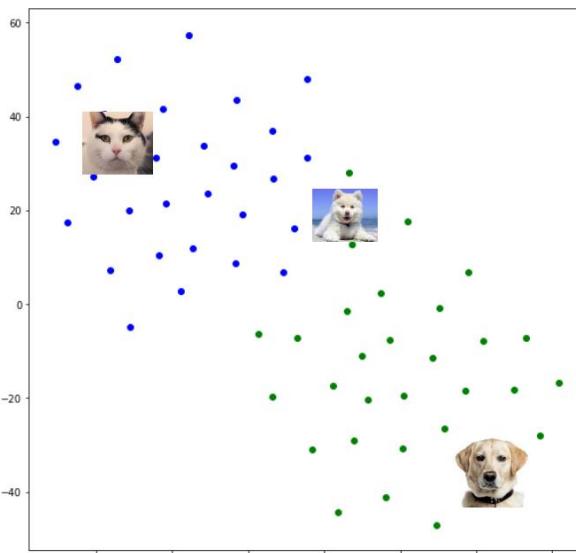
Super-Resolution



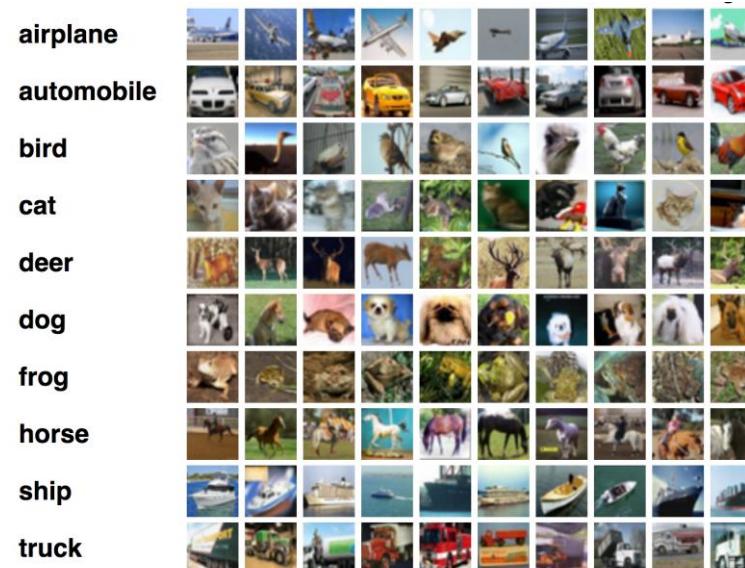
Deblurring

# SC as Feature Descriptor

- Images can be represented in terms of **sparse coefficients**
- Any DL/ML model can be trained based on those sparse features as features



Clustering



Object Recognition

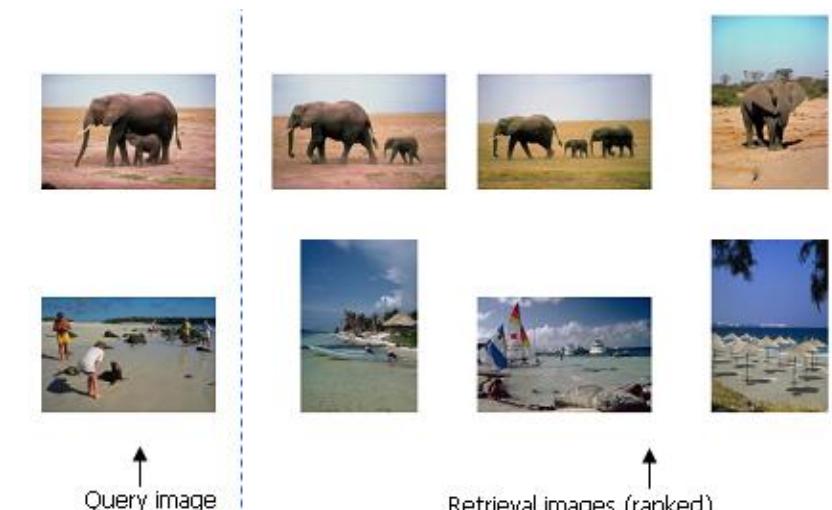
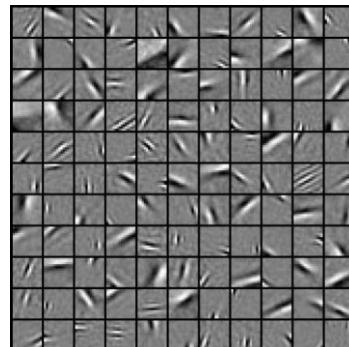


Image Retrieval

# Analysis of Sparse Coding

- **Pros**
  - Faster and works well for **small image patches**
  - Similar as how visual cortex works i.e. biologically plausible
- **Cons**
  - Does not scale well for **bigger patches/images**
  - Sparse codes are only single level of image representation, **not hierarchical** as neural networks



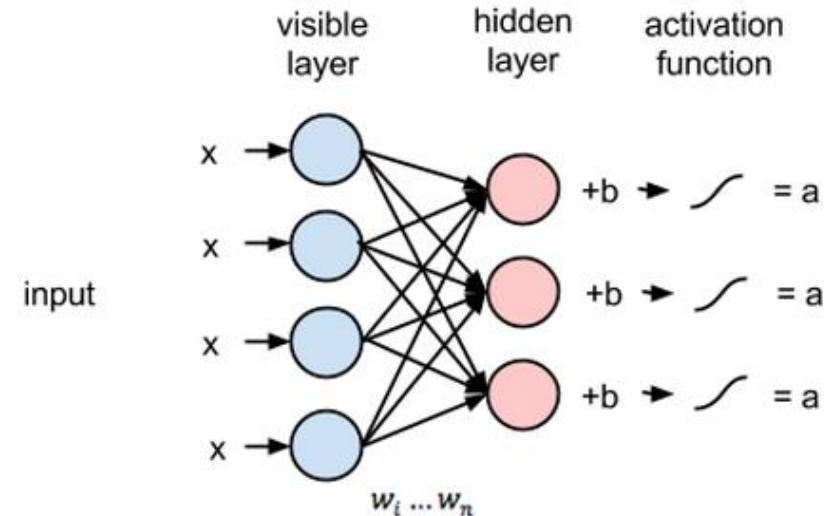
VS



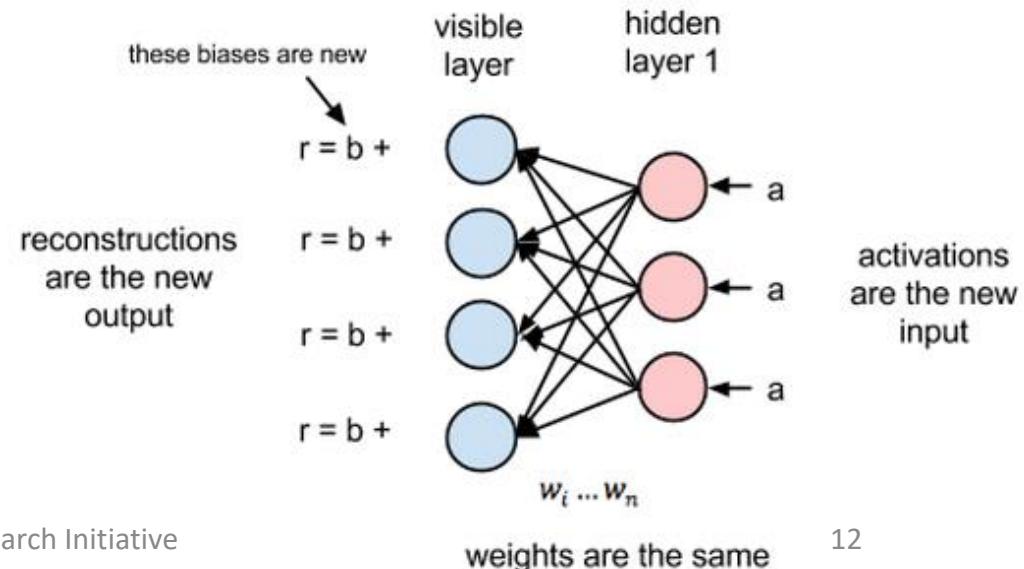
# Restricted Boltzmann Machine

- **RBM** is a symmetrical, bipartite and bidirectional graph
- **Bipartite:** No connection b/w nodes of same layer
- **Symmetrical:** It uses shared weights for both activation and reconstruction
- **Bidirectional:** Connection is common for both activation and reconstruction
- RBM gets an input, generate some activation based weights. Then the activation is given as input from the hidden layer to reconstruct the input again using same weights.

Stage 1: Activation



Stage 2: Reconstruction



# RBM Training

- Energy function of RBM

$$E(\mathbf{v}, \mathbf{h}) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i h_j w_{ij}$$

- Joint Distribution for  $\mathbf{v}$  and  $\mathbf{h}$

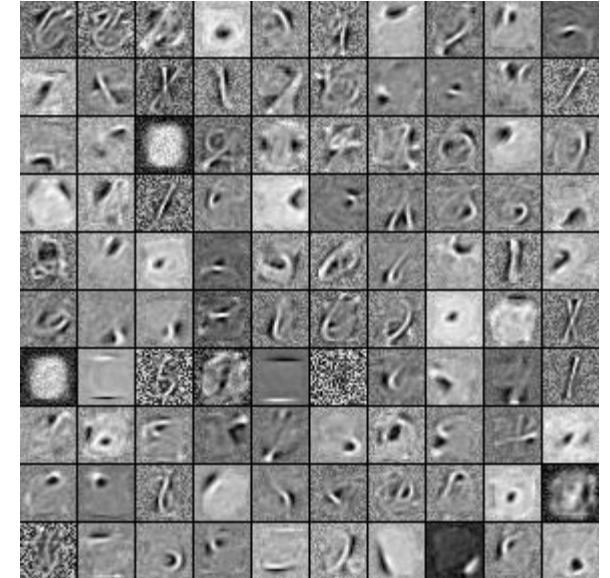
$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v}, \mathbf{h})}$$

$$Z = \sum_{\mathbf{v}, \mathbf{h}} e^{-E(\mathbf{v}, \mathbf{h})}$$

- Conditional probabilities for  $\mathbf{h}$  and  $\mathbf{v}$

$$p(h_j = 1 | \mathbf{v}) = \frac{1}{1 + e^{-(b_j + W_j v_i)}} = \sigma(b_j + \sum_i v_i w_{ij})$$

$$p(v_i = 1 | \mathbf{h}) = \frac{1}{1 + e^{-(a_i + W_i h_j)}} = \sigma(a_i + \sum_j h_j w_{ij})$$



Learned RBM filters for MNIST

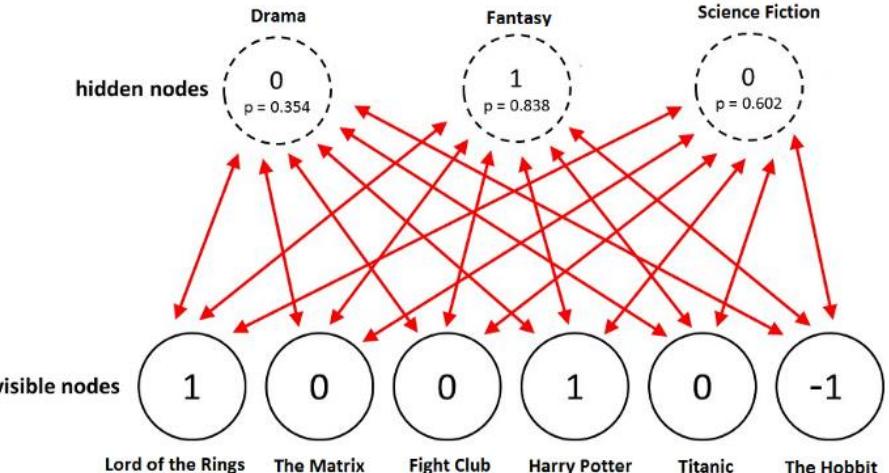


Samples from RBM

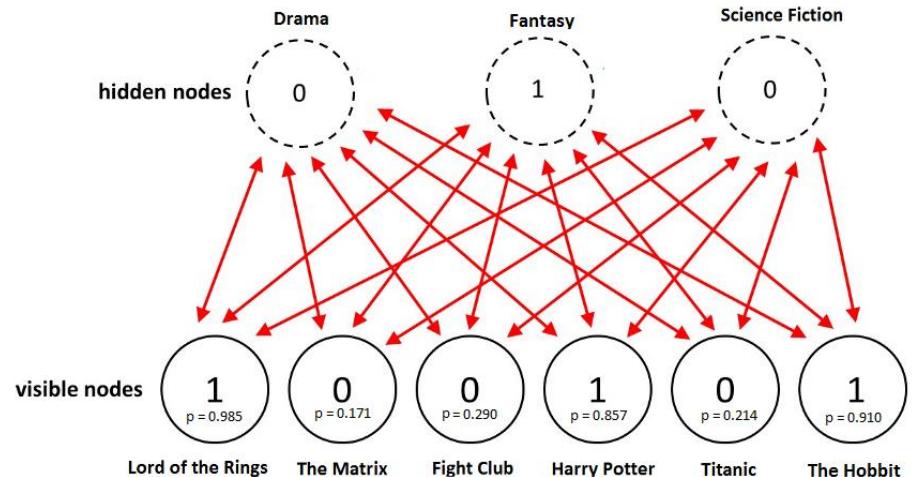
# RBM for Collaborative Filtering

## Steps in RBM

1. Train the network on the data of all users
2. During inference time take the training data of a specific user
3. Use this data to obtain the activations of hidden neurons
4. Use the hidden neuron values to get the activations of input neurons
5. The new values of input neurons show the rating the user would give yet unseen movies



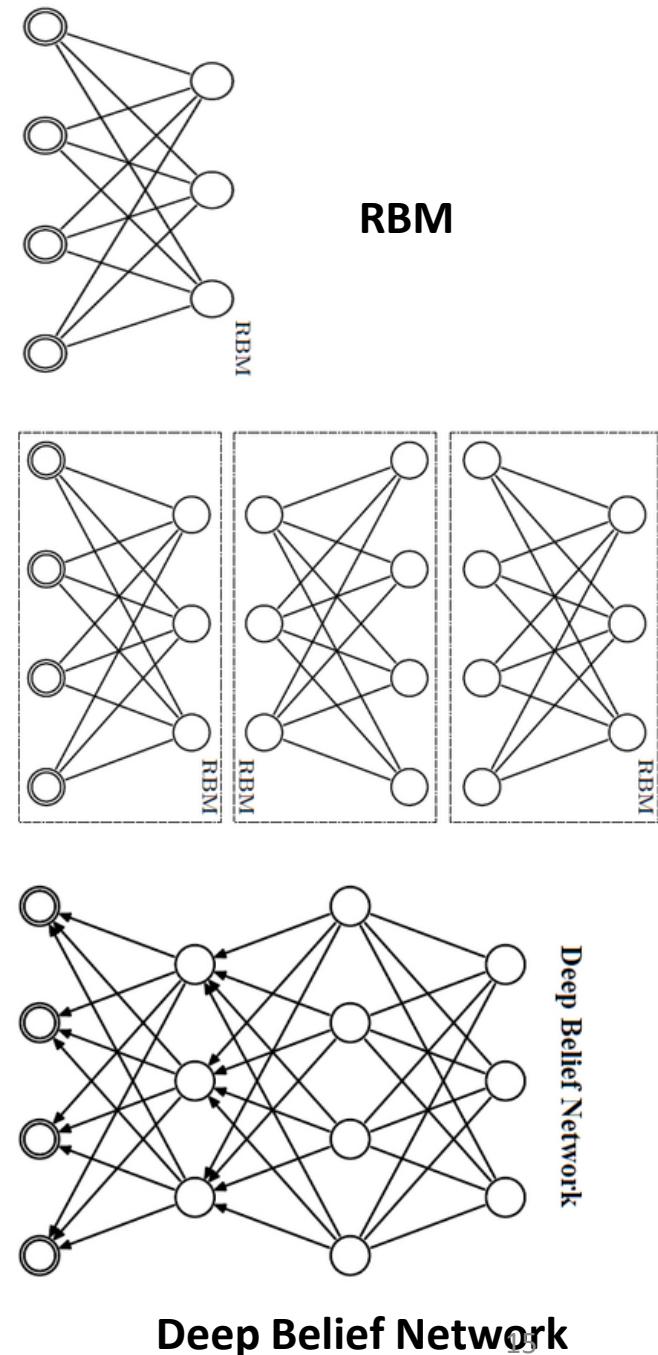
New user input to find latent factor



Reconstruction to predict rating

# Deep-Belief Network

- Deep-Belief network can be defined as a stack of restricted Boltzmann machines
- Each RBM layer communicates with both the previous and subsequent layers
- Stack of RBMs might end with a Softmax layer to create a classifier, or it may simply help cluster unlabeled data in an unsupervised learning scenario
- Often used for **unsupervised pretraining**
- **Applications:** Deep-belief networks are used to recognize, cluster and generate images, video sequences and motion-capture data
- **Note:** Rarely used these days... **but why?**



# Analysis of RBM and DBN

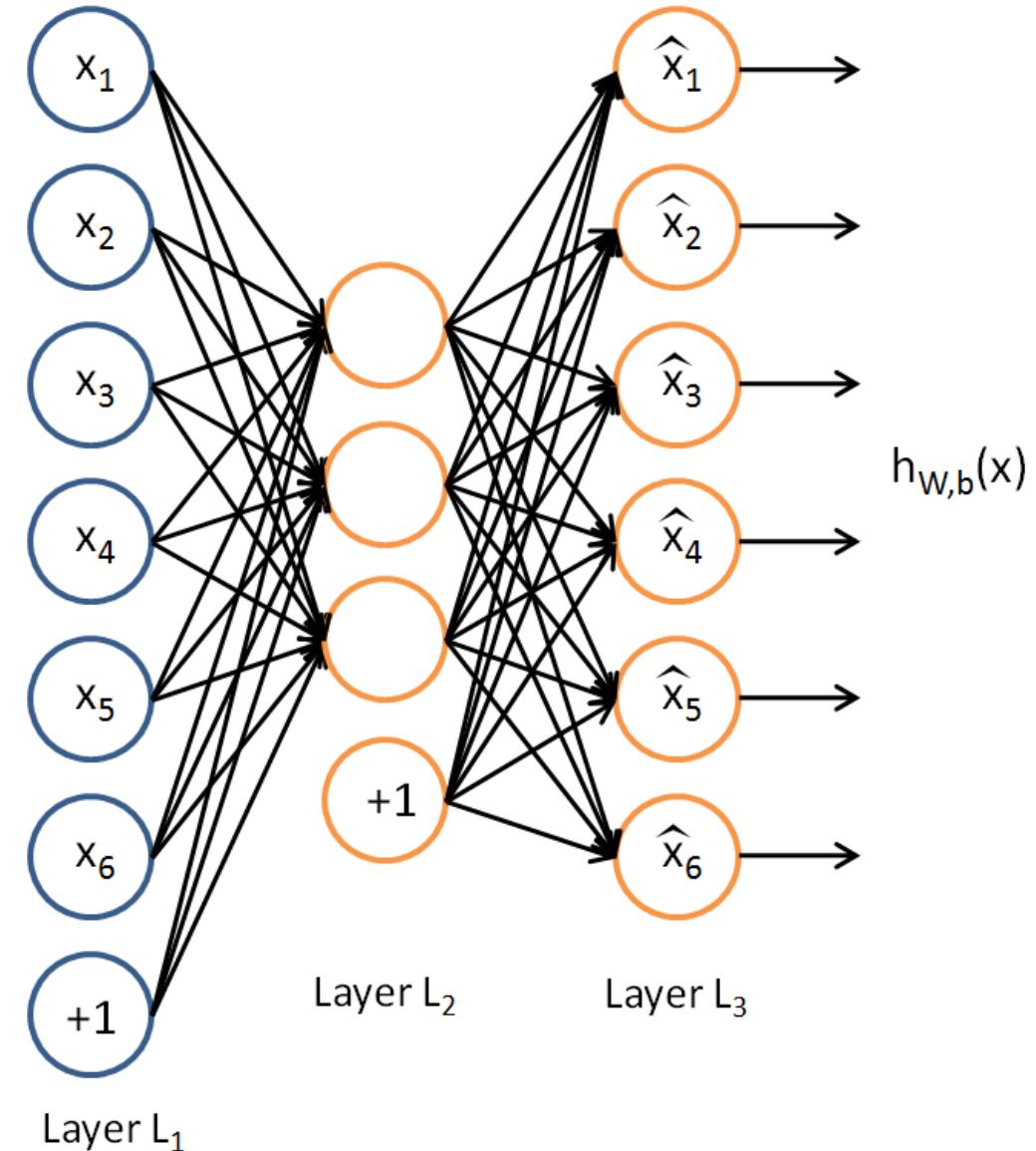
- **Pros**
  - Models the joint probability of data X and labels Y
  - DBN exploits large amount of unlabelled data to train a supervised model i.e. unsupervised pretraining
- **Cons**
  - Training involves complex procedures such as Gibbs Sampling and Contrastive divergence
  - Large amount of labelled data enabled end-to-end supervised training
  - Learning patterns using non-probabilistic models are performing practically better in many task
  - **Take away:** i.e. exact joint and marginal probabilities is not much need in most cases

# Autoencoder

An Autoencoder is a feedforward neural network that learns to predict the input itself in the output.

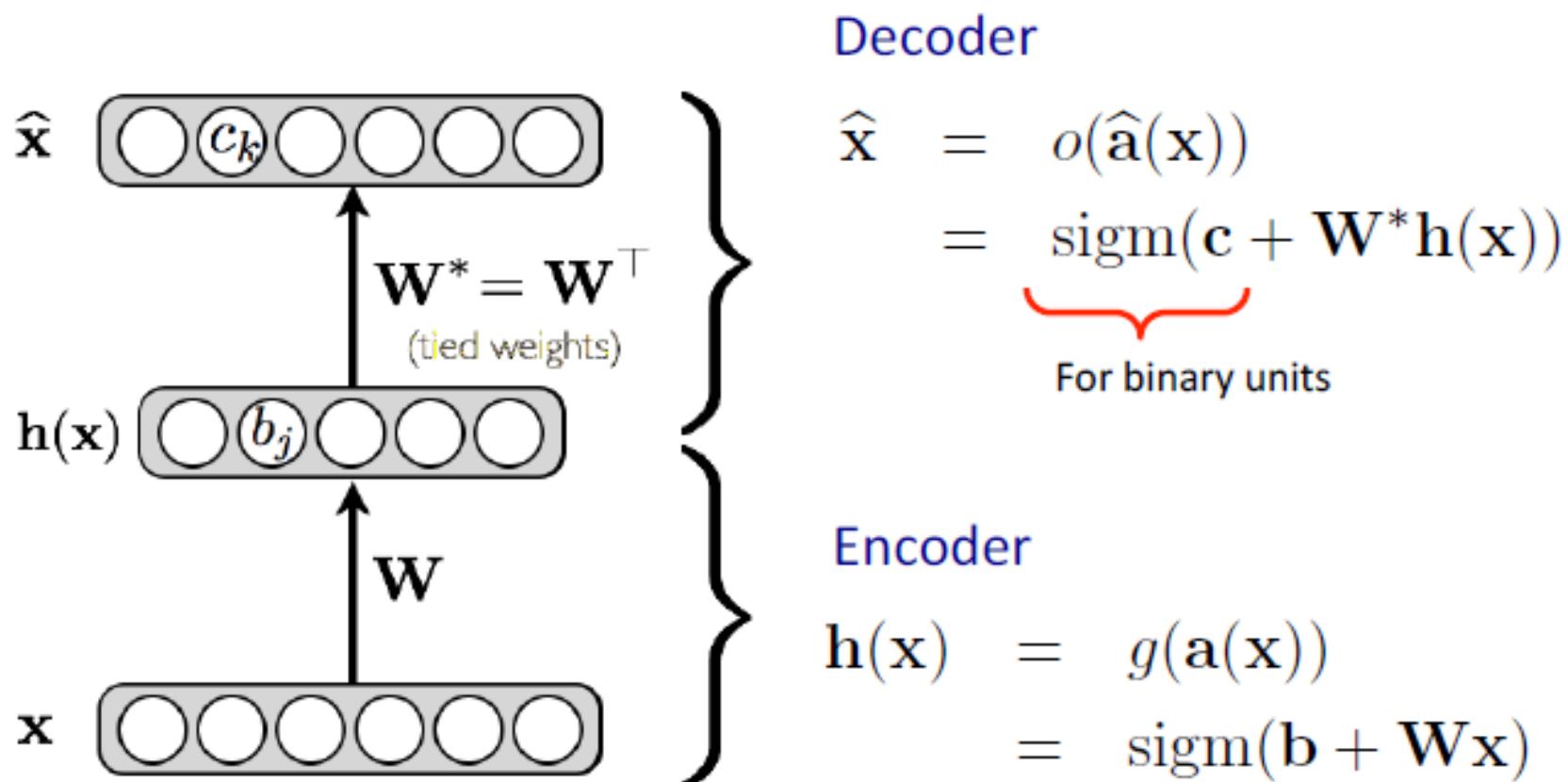
$$y^{(i)} = x^{(i)}$$

- The input-to-hidden part corresponds to an **encoder**
- The hidden-to-output part corresponds to a **decoder**
- Different from RBM which is not a feedforward network but a probabilistic model
- Many cool applications are based on encoder-decoder frameworks



# Autoencoder

- Feed-forward neural network trained to reproduce its input at the output layer



# Loss Function

- Loss function for binary inputs

$$l(f(\mathbf{x})) = - \sum_k (x_k \log(\hat{x}_k) + (1 - x_k) \log(1 - \hat{x}_k))$$

- Cross-entropy error function (reconstruction loss)  $f(\mathbf{x}) \equiv \hat{\mathbf{x}}$

- Loss function for real-valued inputs

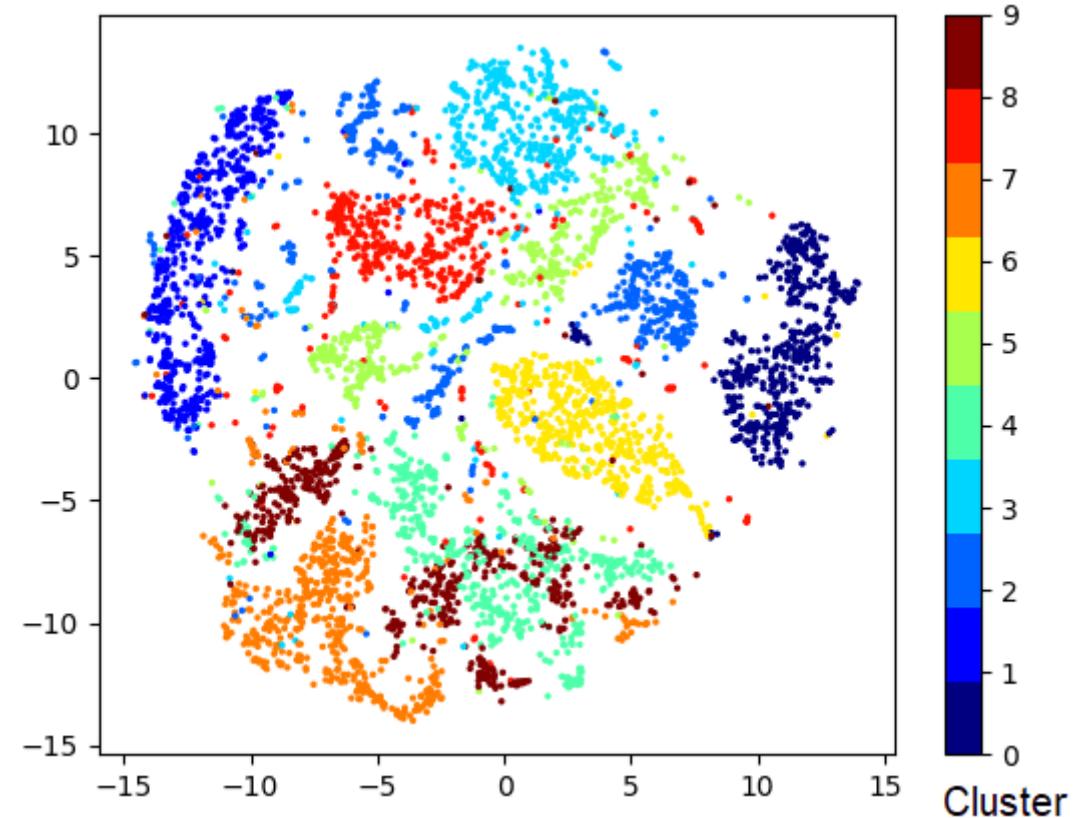
$$l(f(\mathbf{x})) = \frac{1}{2} \sum_k (\hat{x}_k - x_k)^2$$

- sum of squared differences (reconstruction loss)
- we use a linear activation function at the output

# AE for Dimensionality Reduction

- **Visualizing high-dimensional data** is challenging.
- **t-SNE** is the most commonly used method but struggles with large number of dimensions (typically above 32).
- Autoencoders are used as a pre-processing step to reduce the dimensionality, and this compressed representation is used by t-SNE to visualize the data in 2D space.

t-Distributed Stochastic Neighbour Embedding

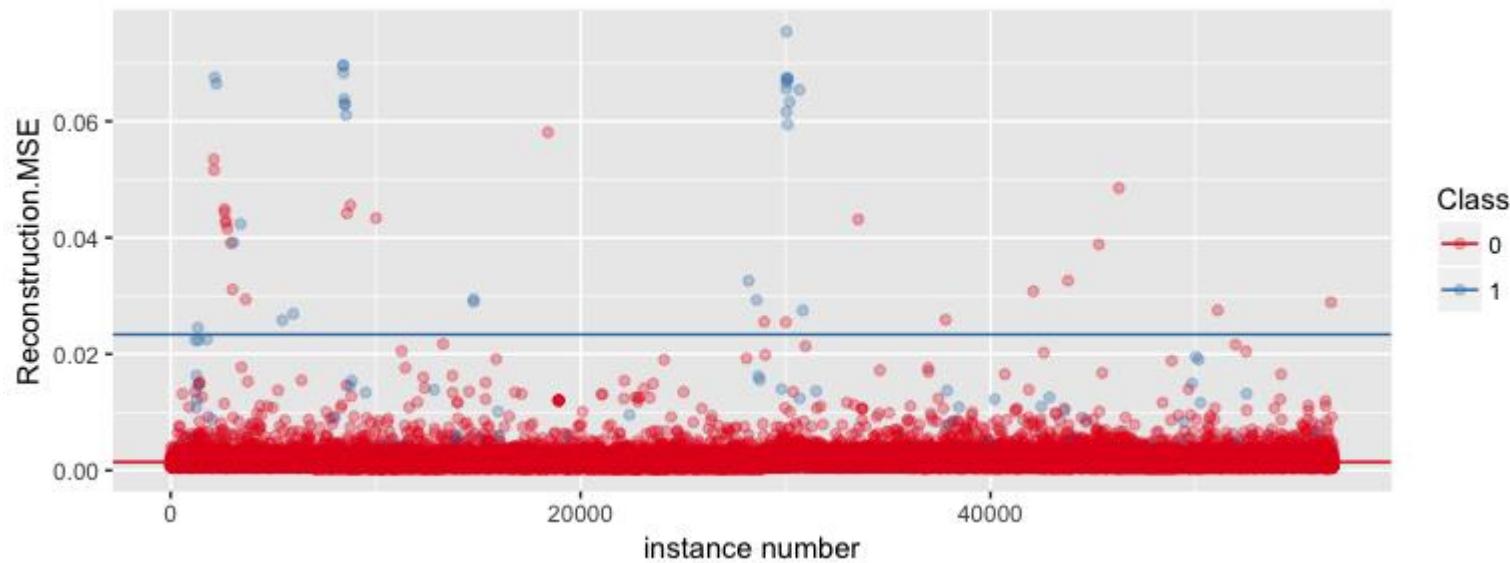


t-SNE visualizations for clustering on MNIST dataset

# AE for Anomaly Detection

- AE can be used for anomaly detection such as Credit Card Frauds
- **Features:** such as time, amount, etc can be learned using an AE
- **Training:** AE will aim to minimize the reconstruction error of training data
- **Testing:** Transactions with higher reconstruction error will be labelled as Credit Card Frauds based on some threshold

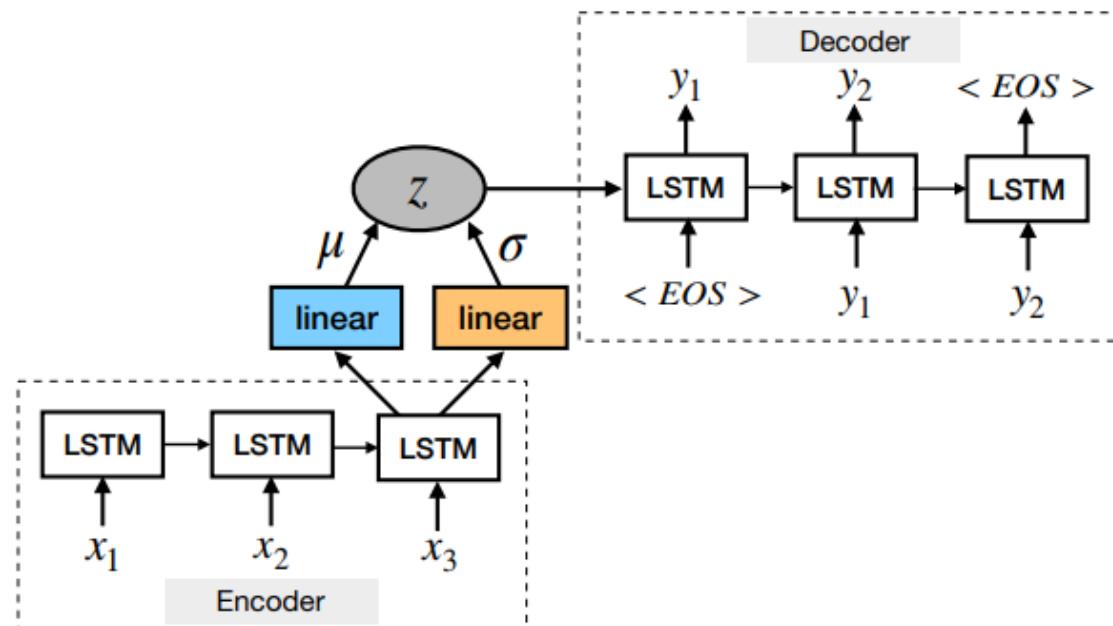
**Note:** there is no perfect classification into fraud and non-fraud cases but the mean MSE is definitely higher for fraudulent transactions than for regular ones



Reconstruction error rate is high for anomalous transactions

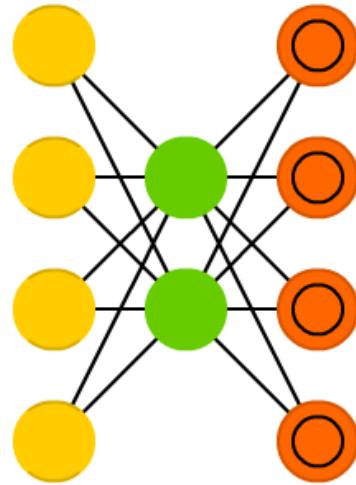
# Recurrent AE for Sentence Generation

- **RNN Encoder** will receive the sentence as input
- **RNN Decoder** will reconstruct input sentence as output
- New sentence can be generated using the latent vector



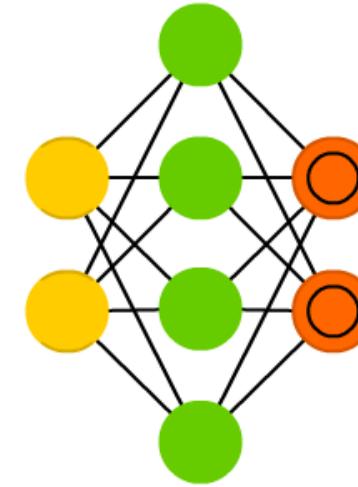
RNN-based VAE for sentence generation

# Autoencoder: Undercomplete vs Overcomplete



**Undercomplete Autoencoder**

- Used for compressing data into low dimension
- Objective is to minimize the reconstruction error



**Overcomplete Autoencoder**

- Used for representing data with more features via high dimension (useful for supervised learning tasks)
- **Caution:** Sometimes memorizes data since reconstruction error is low... **Solution: Add noise**

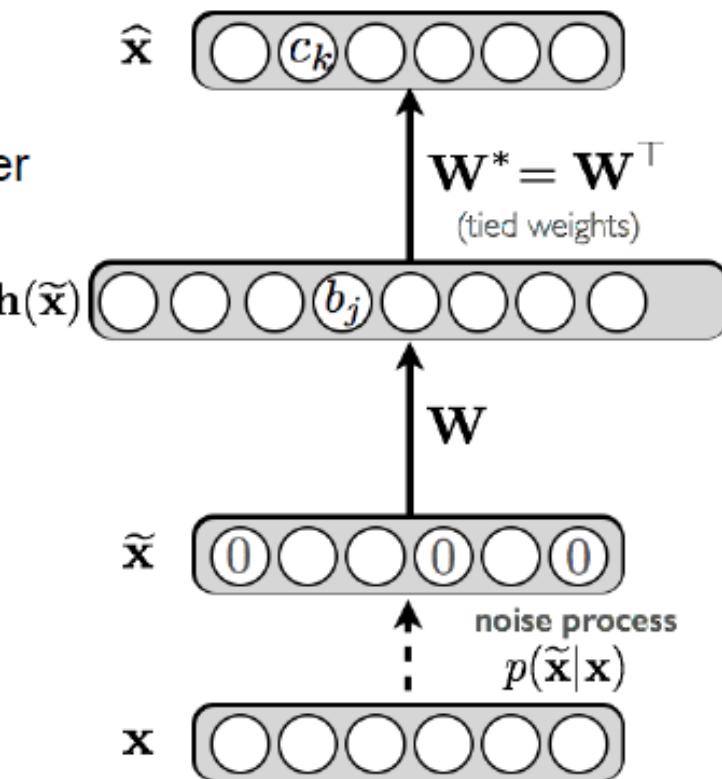
# Denoising Autoencoder

- Idea: representation should be robust to introduction of noise:

- random assignment of subset of inputs to 0, with probability  $\nu$
- Similar to dropouts on the input layer
- Gaussian additive noise

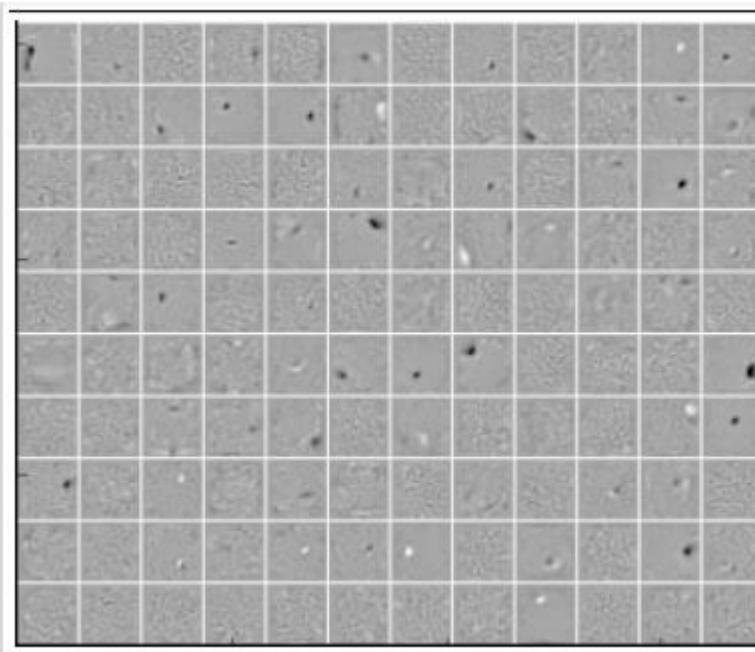
- Reconstruction  $\hat{\mathbf{x}}$  computed from the corrupted input  $\tilde{\mathbf{x}}$

- Loss function compares  $\hat{\mathbf{x}}$  reconstruction with the noiseless input  $\mathbf{x}$

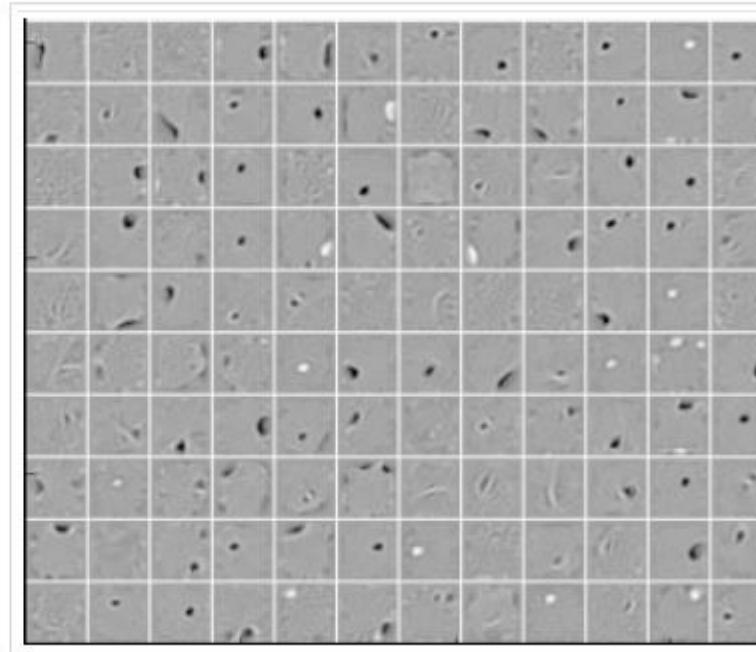


# Corruption vs Learned Filters

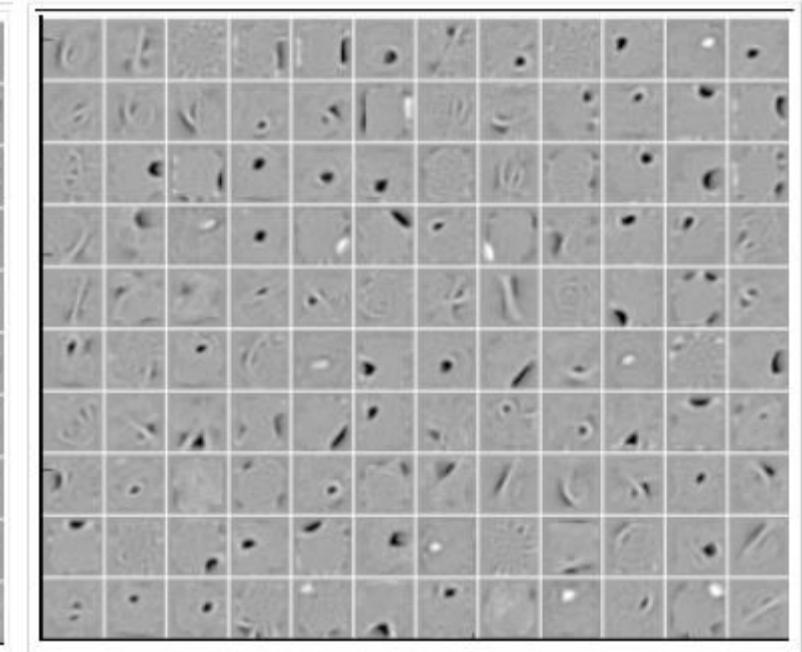
Non-corrupted



25% corrupted input



50% corrupted input

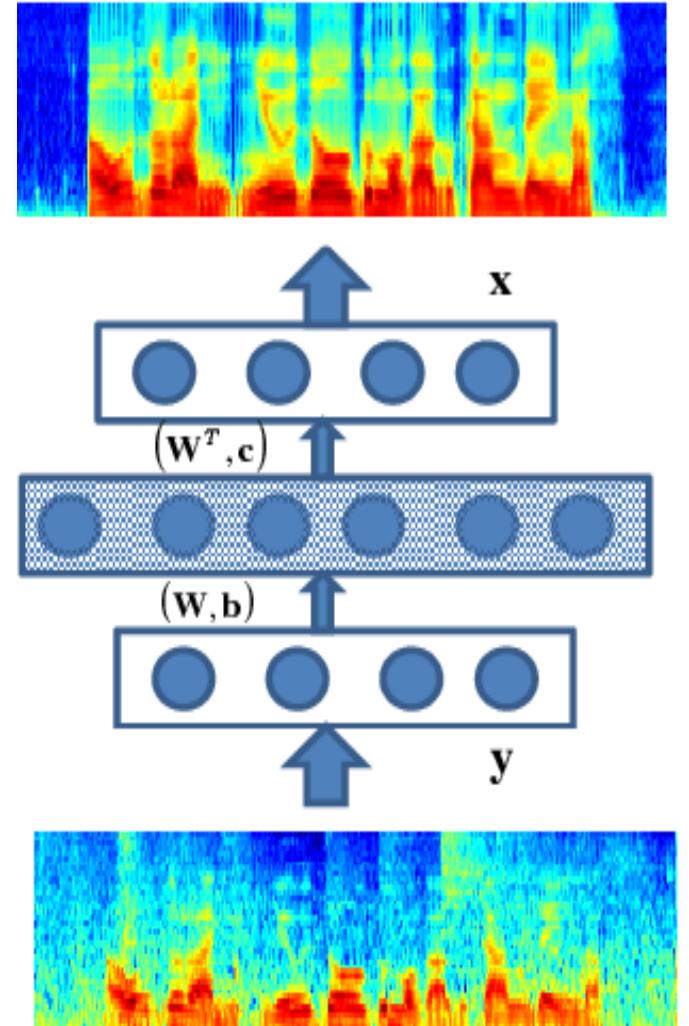


**More the corruption, filters are learned well**

**Caution: Too much corruption can lead to lower reconstruction**

# DAE for Speech Enhancement

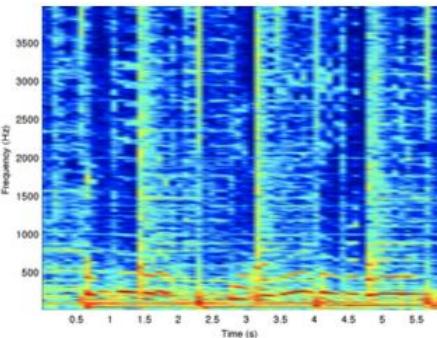
- **Features:** Raw audio or frequency features such as STFT, MFCC, etc.
- **Training:** DAE will get noisy audio as input and aim to construct noiseless audio as output
- **Testing:** Noisy audio input will be processed where DAE will loose the noises in the latent representation and clean audio will come as output in decoder.
- **Data:** Noisy and noiseless versions of same audio
- What if you have only noiseless audio?
  - Solution: Add some noise through augmentation and generate
- What if you have only noisy audio?
  - Solution: Manual/semi automated cleaning of audio



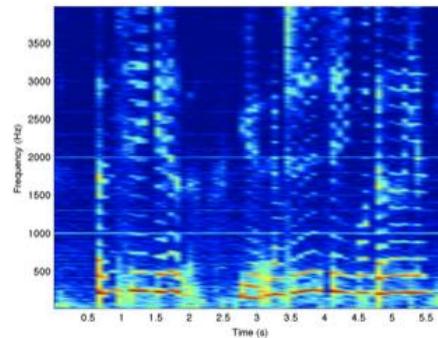
DAE based Speech Enhancer

# DAE for Music Voice Separation

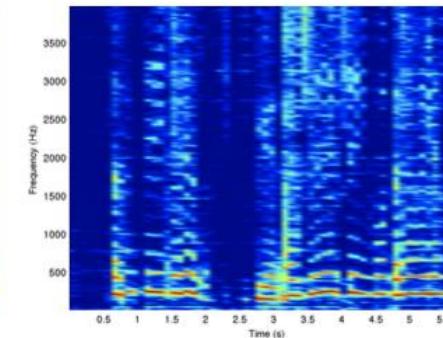
- Music can be model as noise where voice can be modelled as target
- Or voice can be modelled as noise where music can be predicted as target
- Recent Example: MaD TwinNet for music voice separation



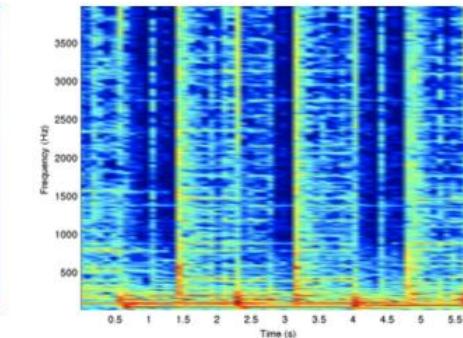
(a) Mixture



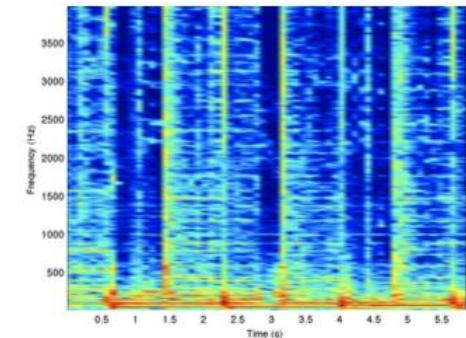
(b) Clean vocal



(c) Recovered vocal



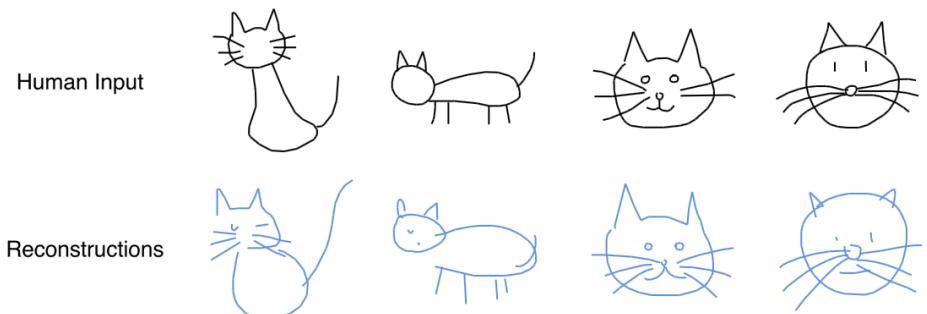
(d) Clean music



(e) Recovered music

Output of DAE that separates voice from music

# Sequence-to-sequence (seq2seq) Autoencoder



$$\text{cat} + (\text{pig} - \text{pig}) = \text{cat}$$

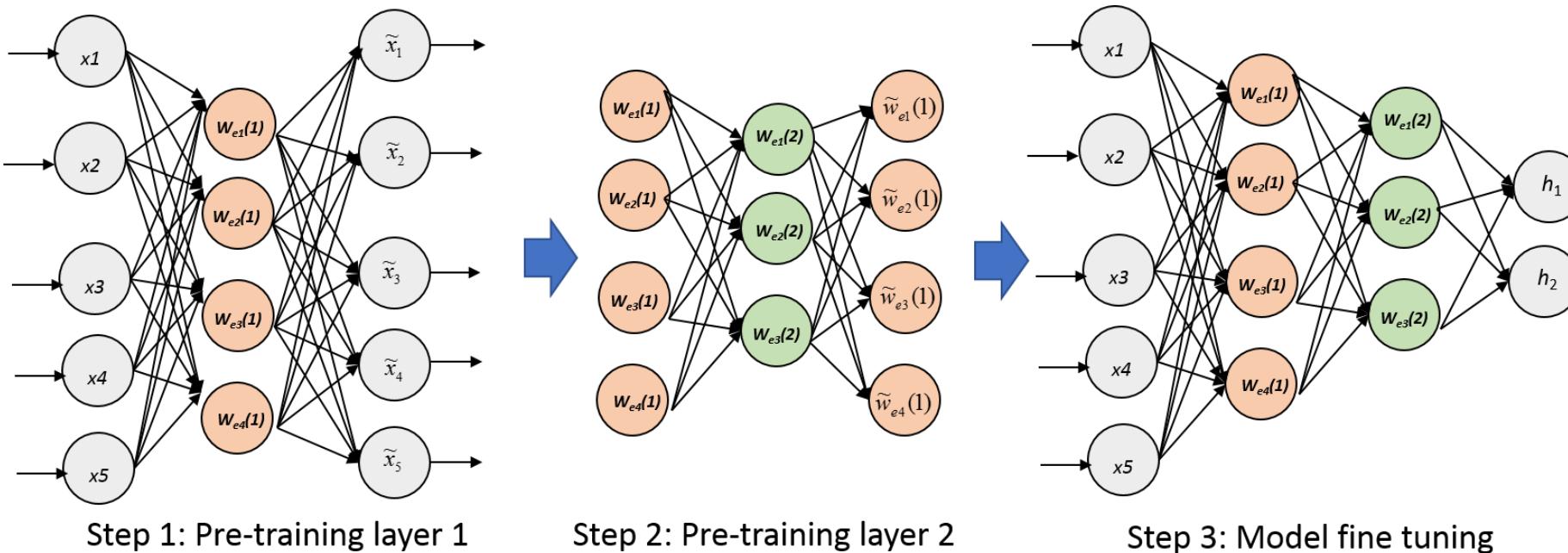
$$\text{pig} + (\text{cat} - \text{cat}) = \text{pig}$$

Latent Space Arithmetic

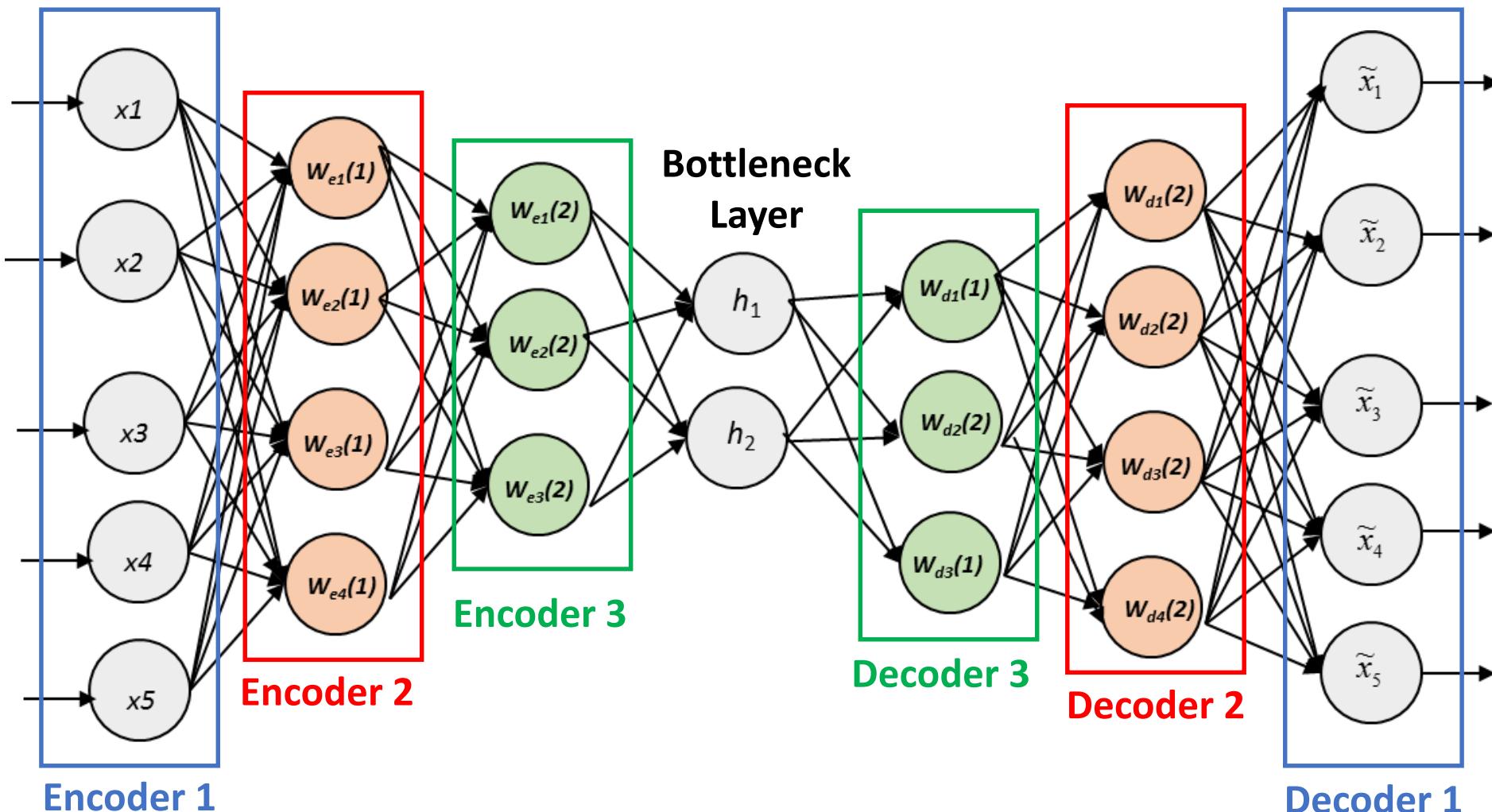
Magenta Project <https://magenta.tensorflow.org/>

# Stacked Autoencoders

- Autoencoders can be trained layer-wise and further concatenated to create stack of autoencoders
- Each hidden layer will be treated as input layer to further compress it

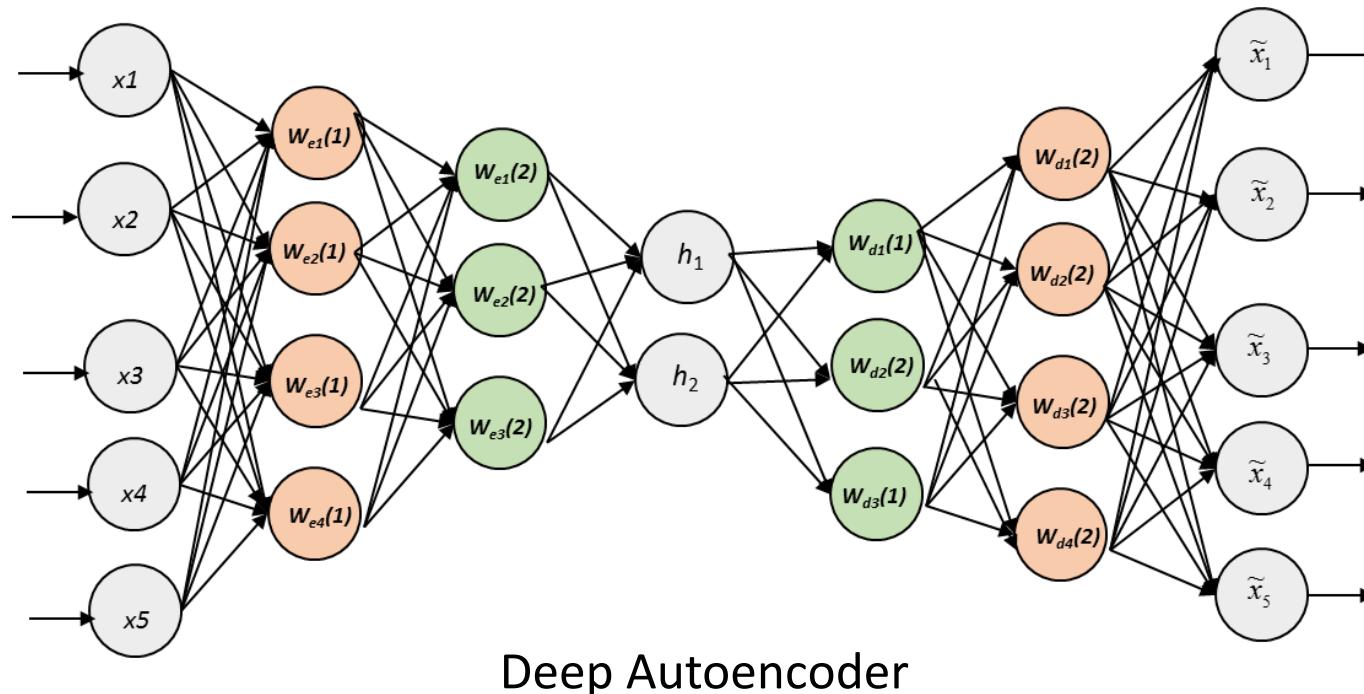


# Stacked Autoencoders



# Deep Autoencoder

- Deep Neural Network with multilayer encoder and decoder can be trained
- Any number of layers and their nodes can be chosen but usually symmetry is preferred
- Output activation depends upon input values X

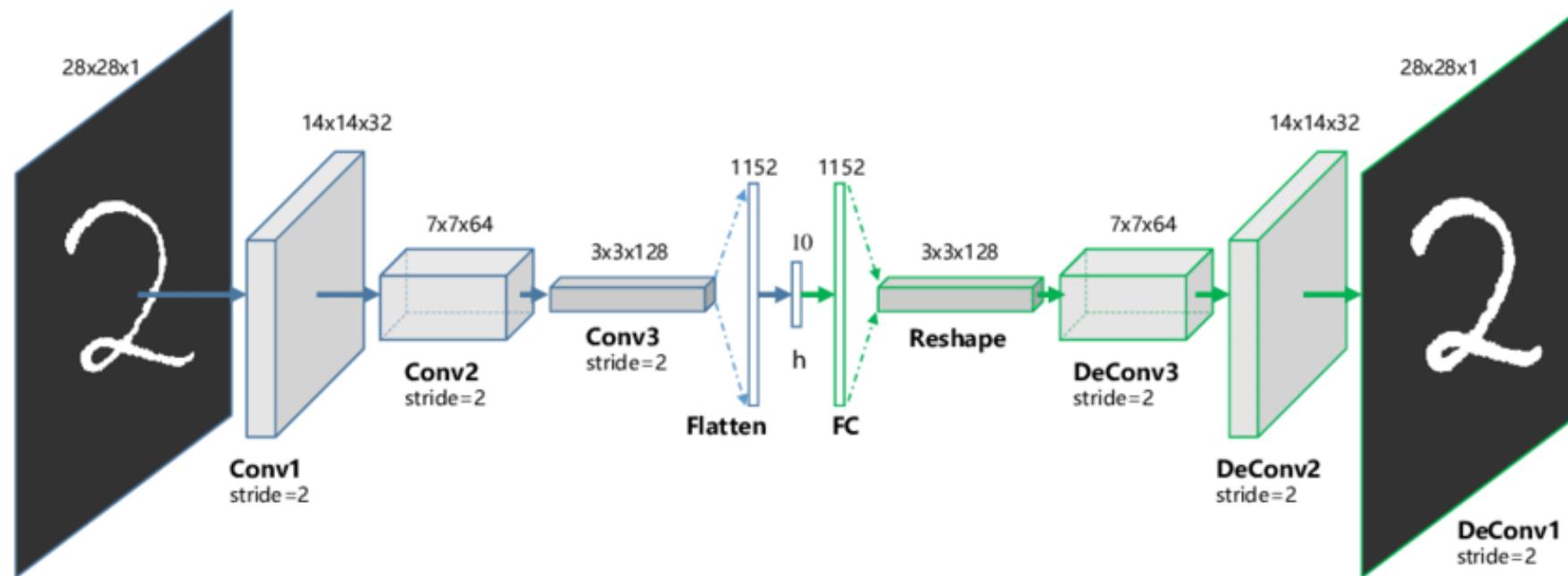


# Stacked vs Deep Autoencoder

- Stacked autoencoders
  - We can continuously monitor how each layer is learning
  - Error during training of each layer is added further and become overall error of model
- Deep autoencoders
  - Training is easy where error reaches minimum as there is no intervention from us
- Stacked autoencoders are preferred less as layer-wise pretraining is hectic.
- End-to-end training of deep autoencoder is preferred in most cases
- **Naming convention:** people often prefer the term **autoencoder** to mean any autoencoder in general, irrespective of whether they are denoising, stacked, deep, convolutional, recurrent etc.

# Convolutional Autoencoder

- ConvAE comprises encoder with convolution pooling and decoder with deconvolution and up sampling.
- Referred as Hourglass Model 



Convolutional Autoencoder

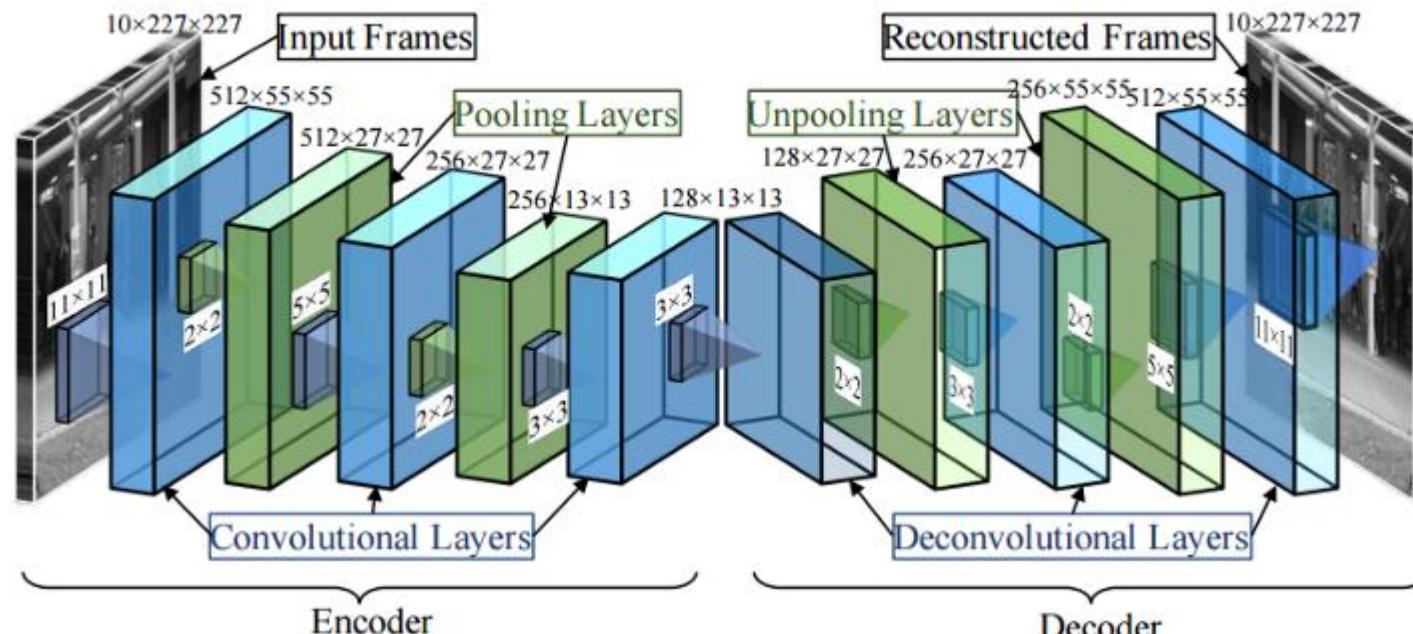
# ConvAE for Image Retrieval

- Hidden activation of ConvAE can be exploited as feature vector which is further used for task such as retrieval or supervised image classification etc.



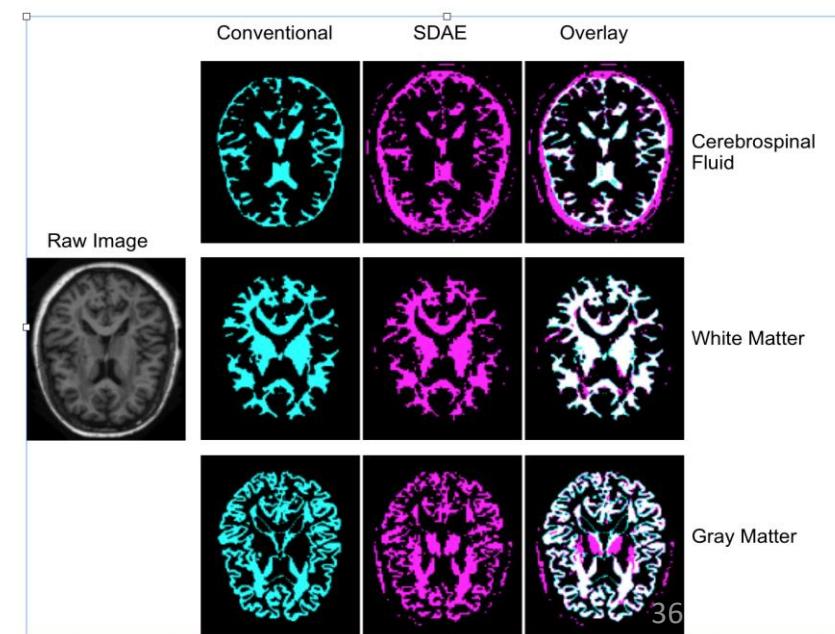
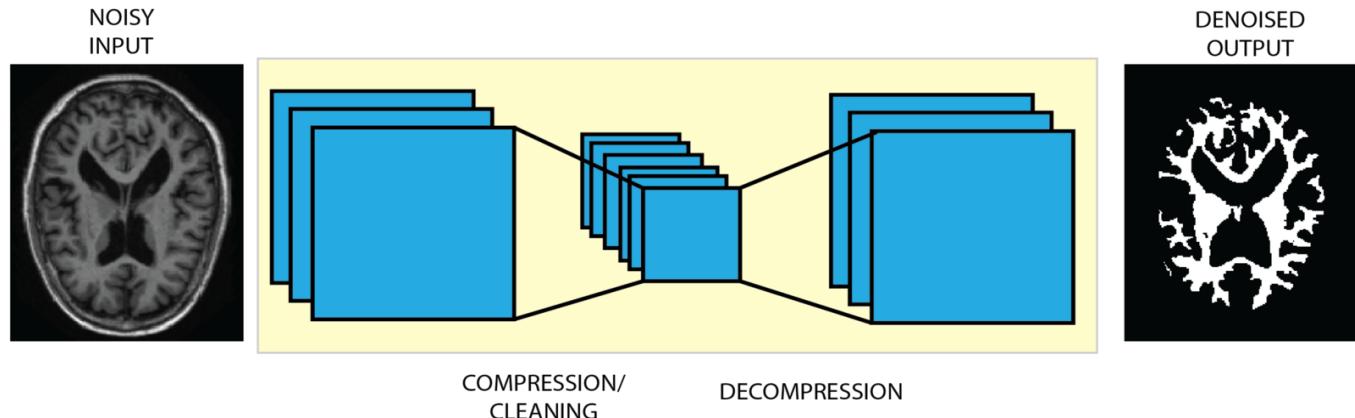
# Conv AE for Anomaly Detection

- **Training:** Video frames can be compressed and decompressed using ConvAE
- **Testing:** Frames with higher reconstruction error are labelled as anomalous frames



# SDAE for Image Denoising

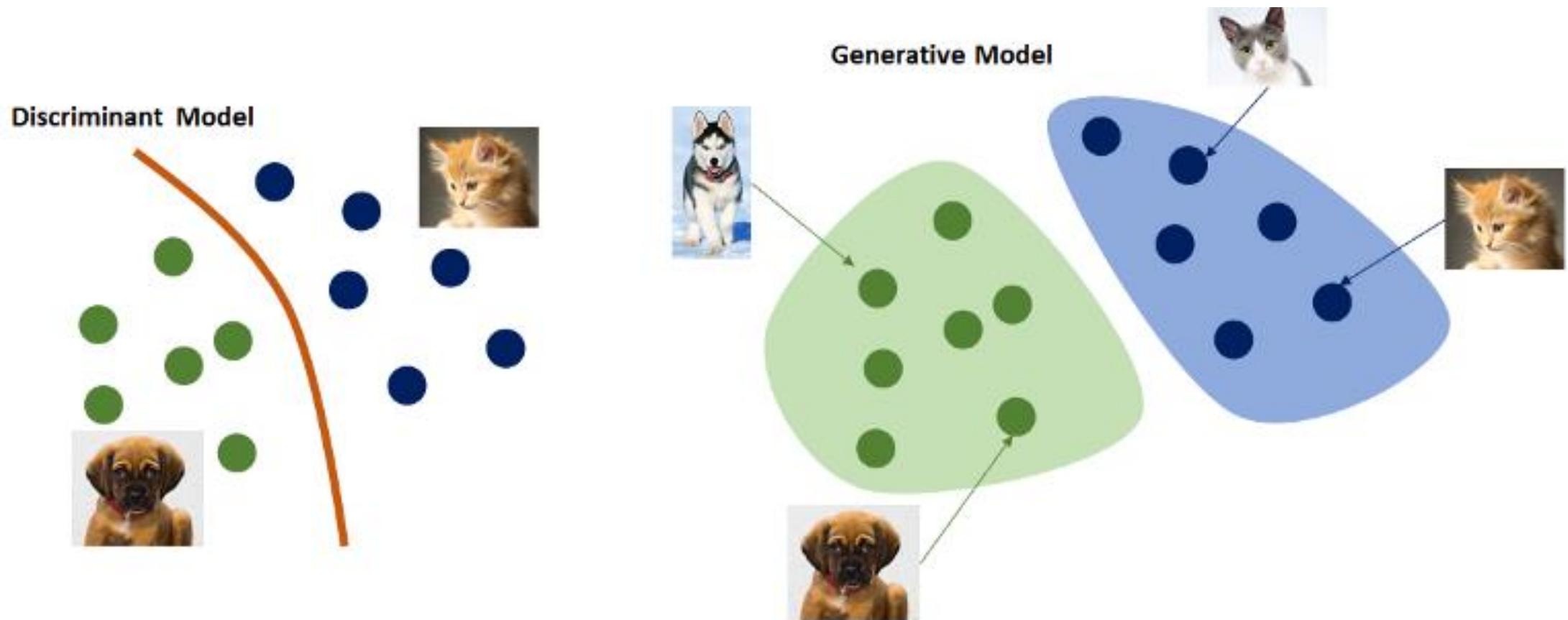
- The ‘Noisy Input’ images is first encoded into the latent space, which has lower dimensionality.
- This is similar to compression that leads to loss of information (in this case we should lose information about the noise).
- The latent space representation is then decoded (the decompression part), which gives us the ‘Denoised’ reconstructed image.



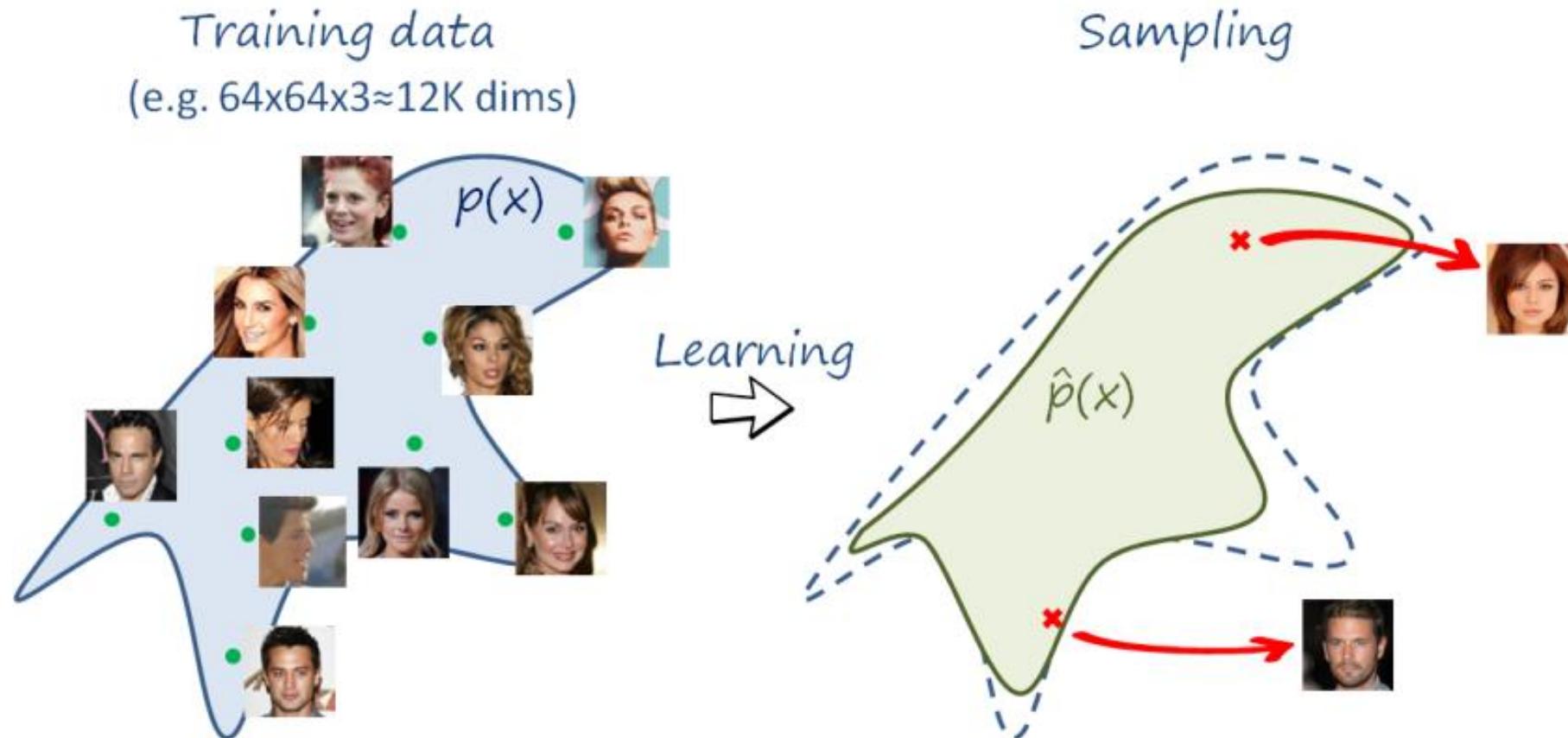
# Discriminative Vs Generative Learning

- **Discriminative models**
- A discriminative model **does not** care how the data is generated. Here we just care about  $P(y|x)$ .
- **Example:** Classifying whether image contains cat or not?
- **Generative models**
- Generative models describe **how data is generated** using probabilistic models. They predict  $P(y|x)$ , the probability of  $y$  given  $x$ , calculating the  $P(x,y)$ , the probability of  $x$  and  $y$ .
- **Example:** Generating a new cat image

# Discriminative Vs Generative Models



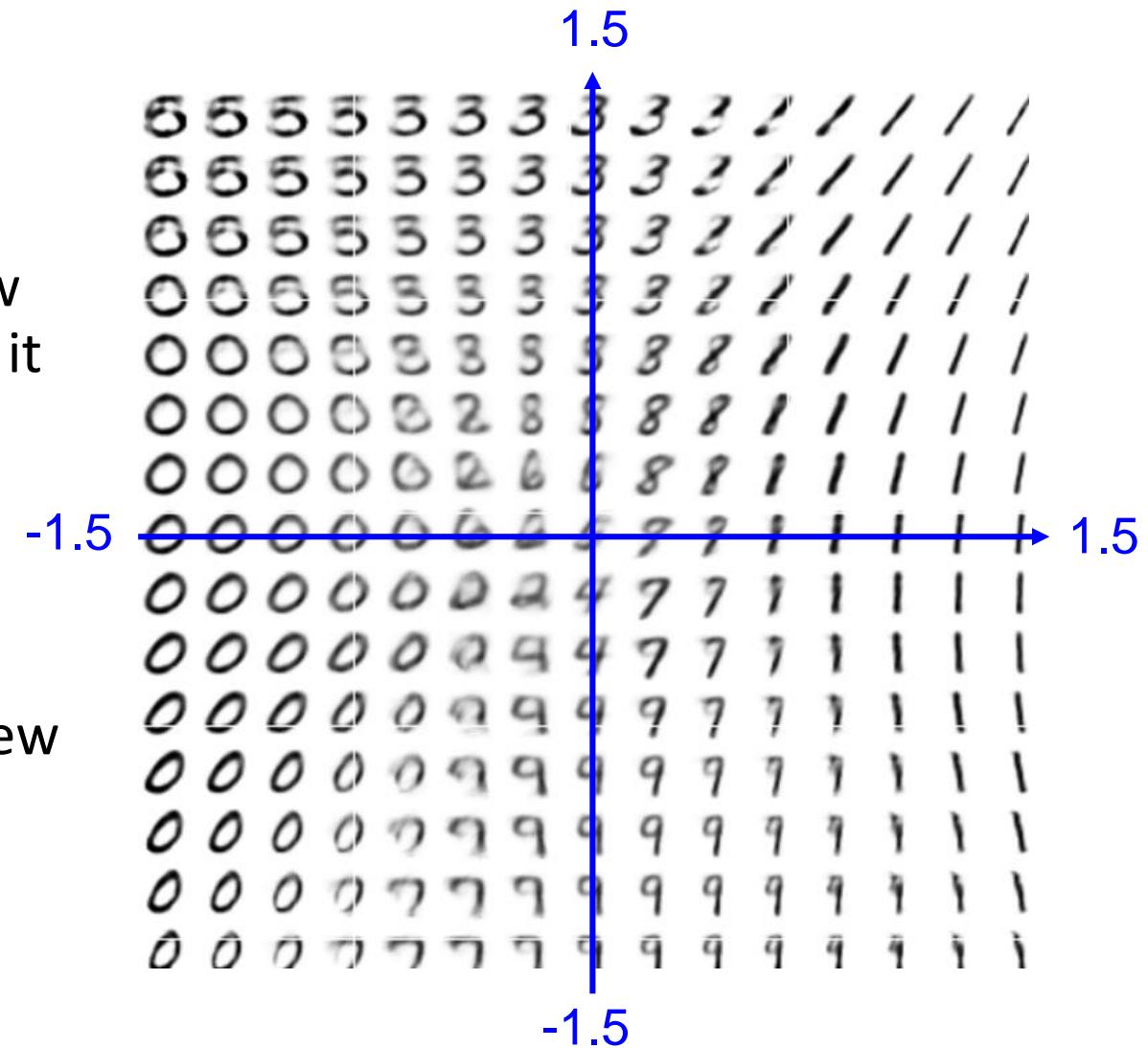
# Sampling New Data



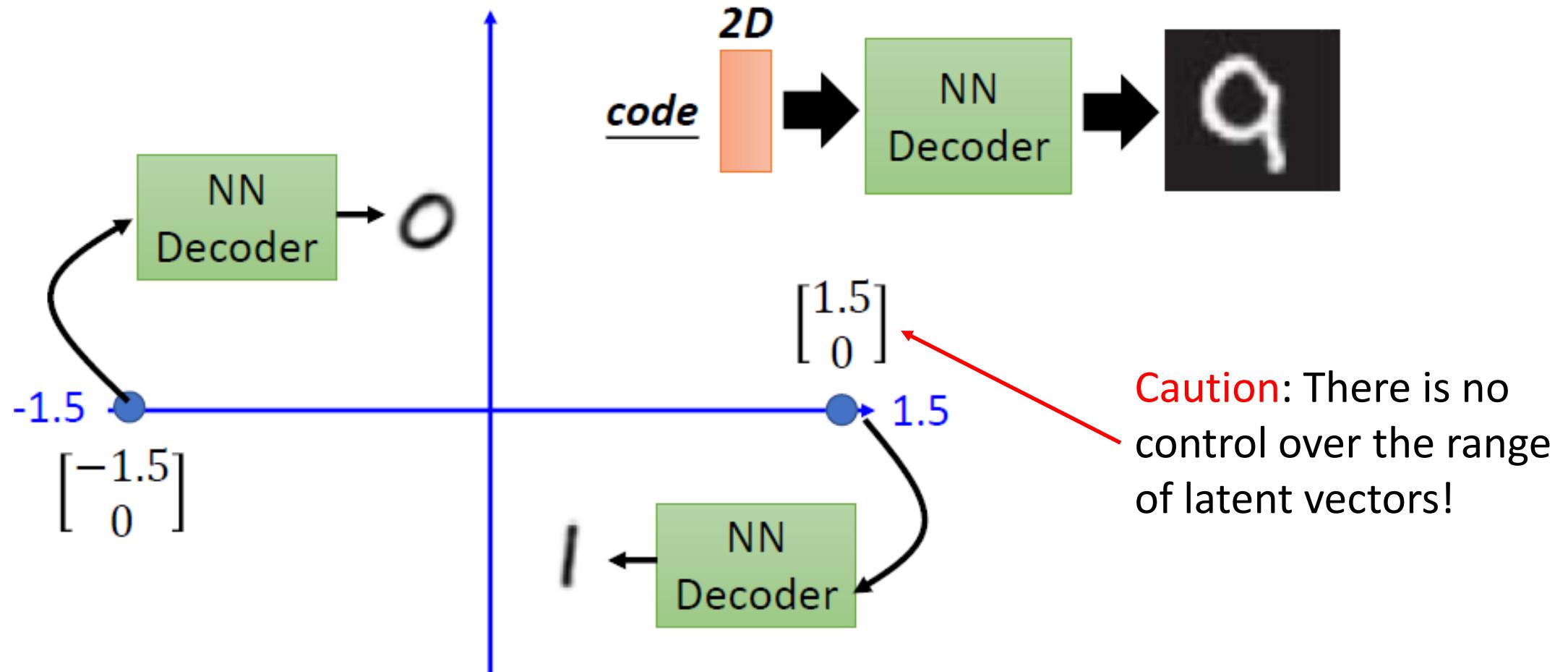
**Generate any kind of data such as Image, Painting, Speech, Music etc..**

# Autoencoder for Image Generation

- Lets imagine an autoencoder with two hidden nodes for MNIST
- Decoder can be used for generating new images if we give 2 numbers of input to it
- Each image in the dataset can be considered as a point in the 2-dimensional latent space
- New data can be sampled by picking a new point from then 2D space where new digit will look as interpolation between the neighbourhood points



# Autoencoder for Image Generation

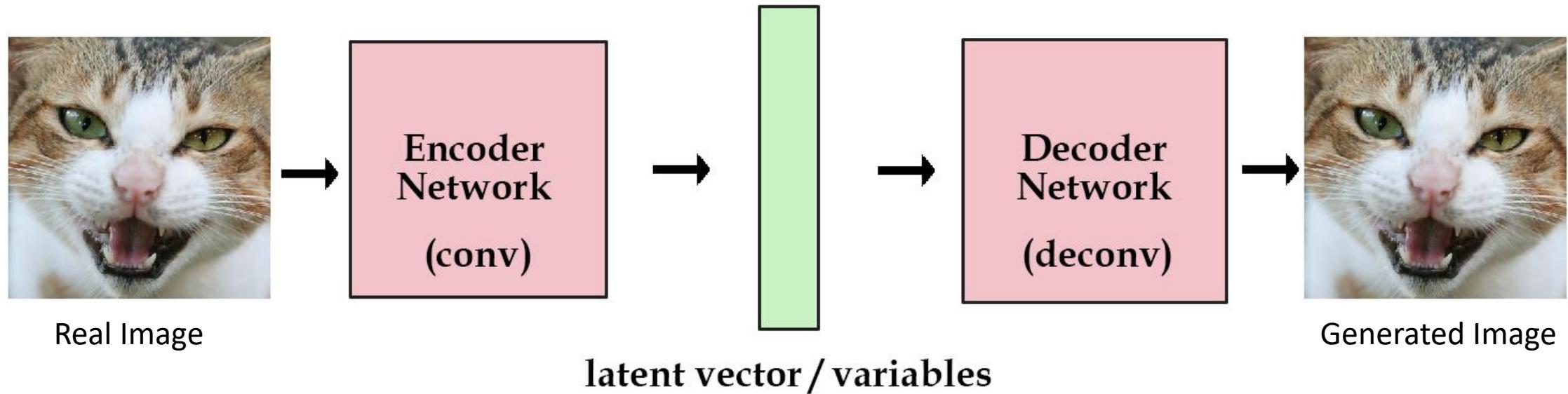


# From Autoencoder to Variational Autoencoder

- Autoencoders learn a “compressed representation” of input automatically by first compressing the input and decompressing it back
- AE has no control over the Latent activations of the network hence data generation becomes difficult
- Instead of learning a mapping function, Variational Auto Encoder learns the parameters of the probability distribution representing the data
- In VAE, we have the control over hidden activations because we keep it near the Unit Gaussian
- Add a Constraint in encoding network, that forces it to generate latent vector, that roughly follow the unit Gaussian distribution.
- This is the constraint that separates it with the standard auto encoder.
- For generation , it is easy now, sample a latent vector from unit Gaussian distribution, and pass it to the decoder. Decoder will generate the image for that latent vector.

# Loss Function

Unit Gaussian Distribution



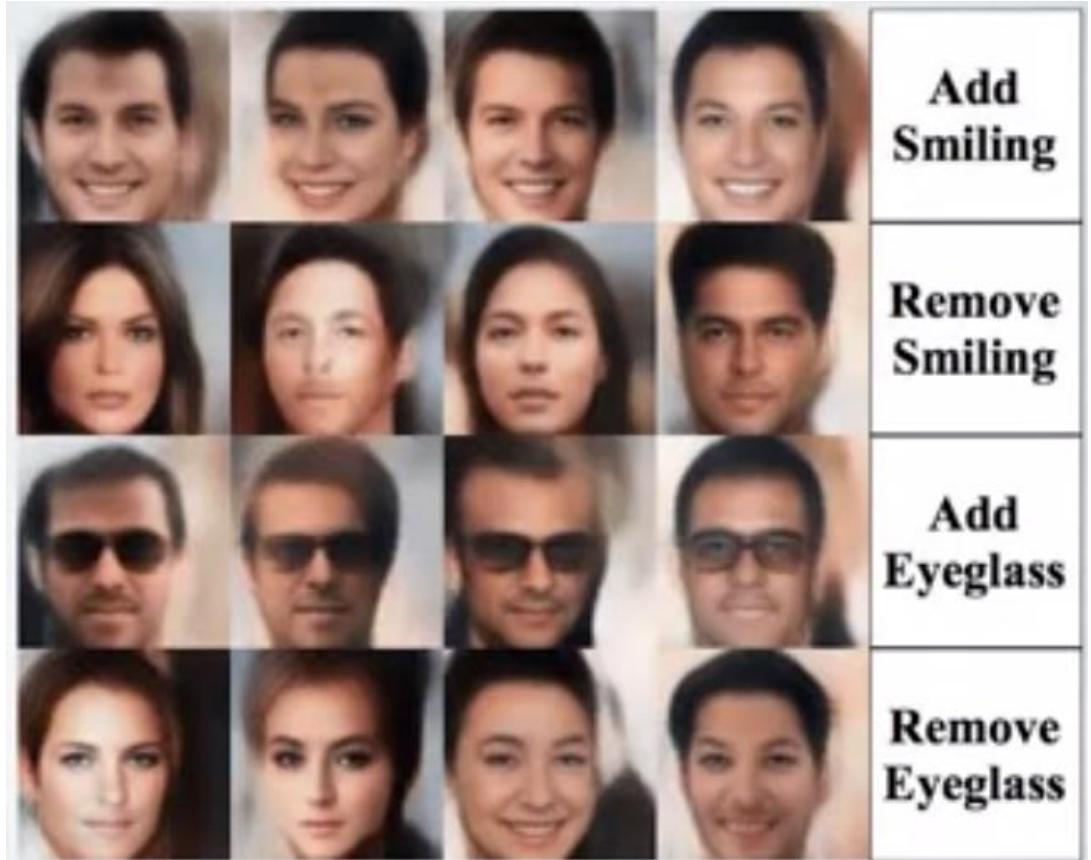
Generation Loss = Mean(Square(Generated Image – Real Image))

Latent Loss = KL-Divergence(Latent Variable, Unit Gaussian)

**Loss = Generation Loss + Latent Loss**

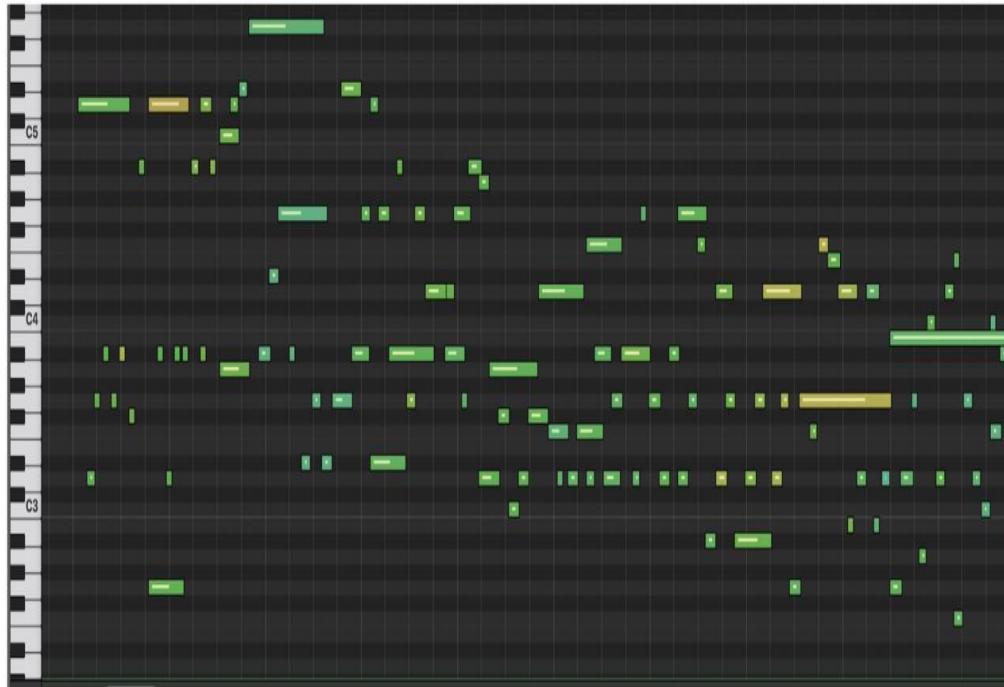
# VAE for Image Editing

- VAE can generate a latent vector that contains activations for specific features such smile, eye glass, beard etc. like [1,1,1]
- We can turn a specific feature on/off in the latent vector during new image generation.
- For example, smiley face with beard can be generated using [1,0,1]



# VAE for Music

- Google uses Autoencoders for creating new music and sounds



MusicVAE

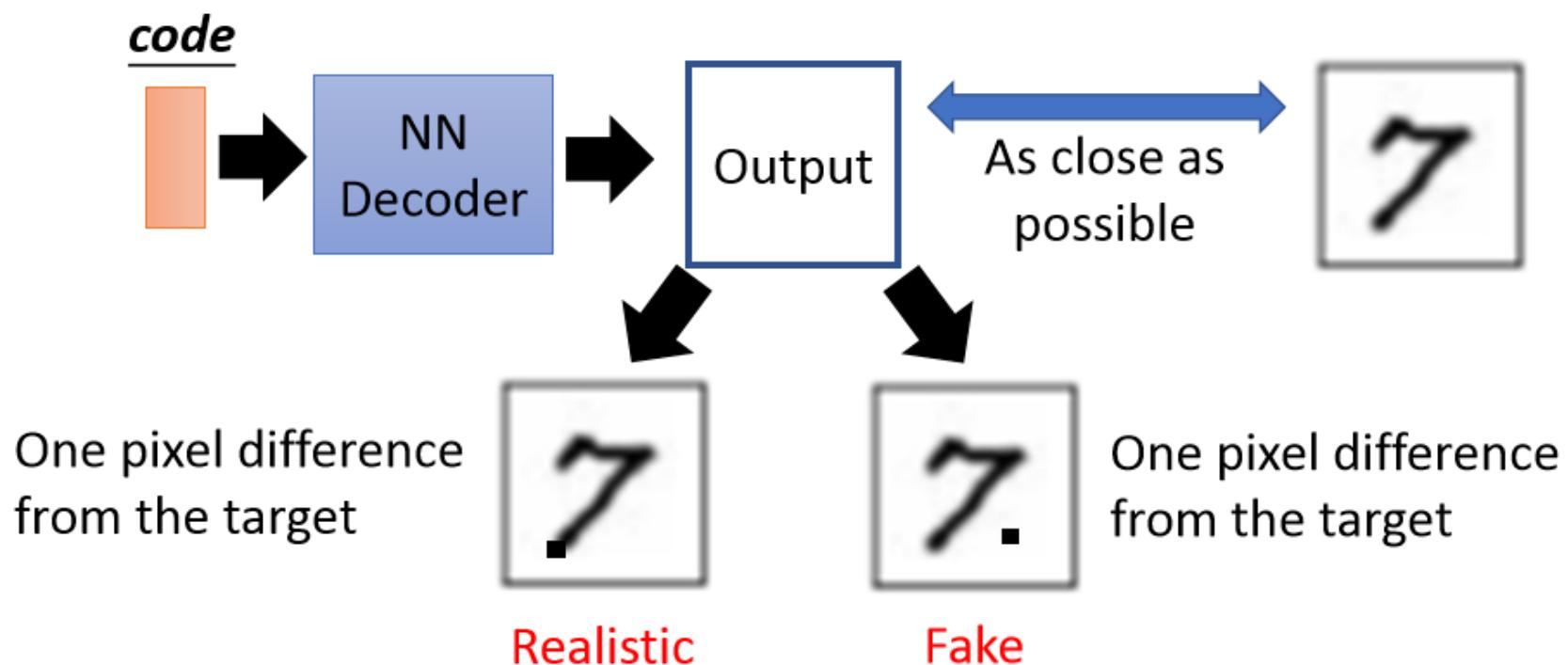


NSynth Super

Magenta Project <https://magenta.tensorflow.org/>

# Problem of VAE

- VAE **does not simulate realistic images** since the original objective of Autoencoders is to compress and decompress specific parts of images

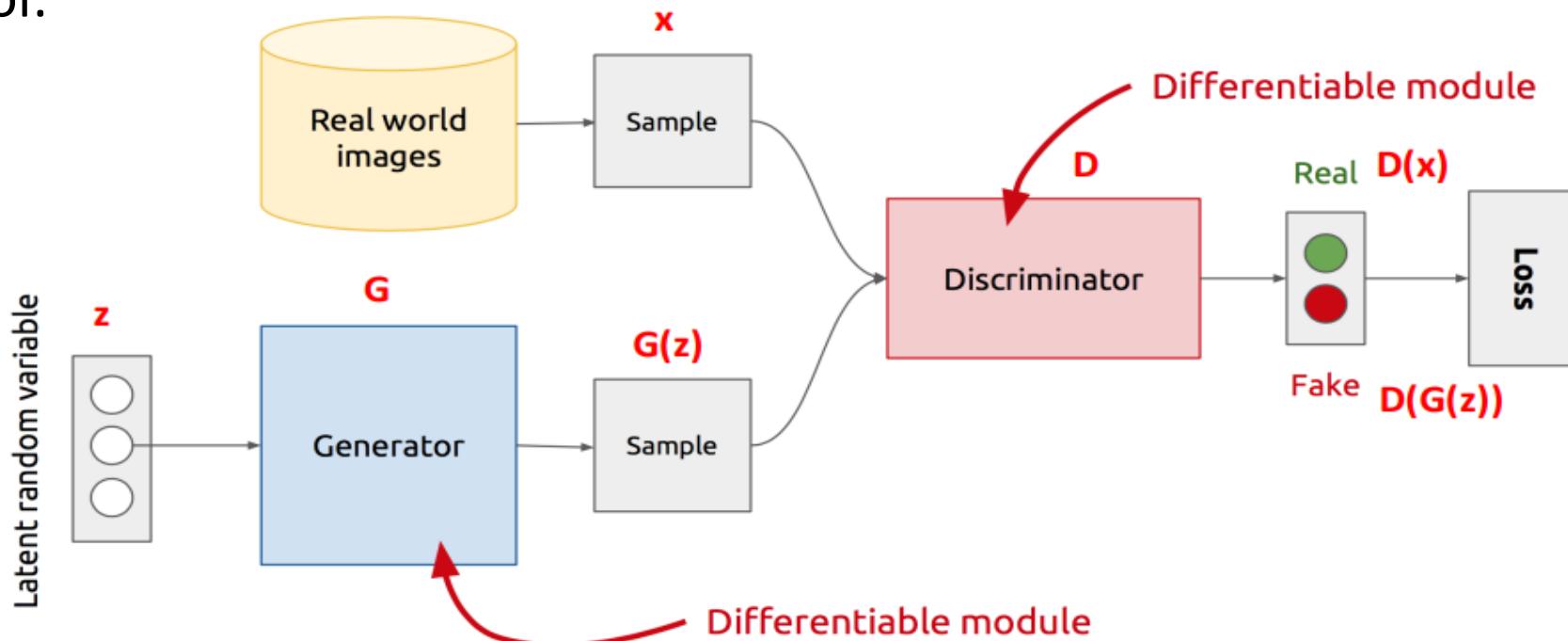


# VAE Images



# Generative Adversarial Network

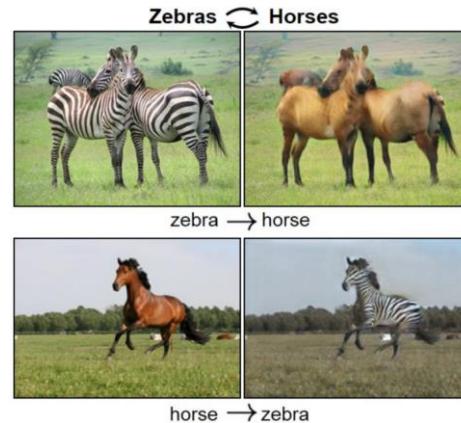
- A **Generator Model**  $G$  learns to capture the data distribution
- A **Discriminator Model**  $D$  estimates the probability that a sample came from the data distribution rather than model distribution.
- This can be think of the minimax game between two networks Discriminator and Generator.



# Recent GAN Applications



Anime Characters



CycleGAN



Image Super Resolution

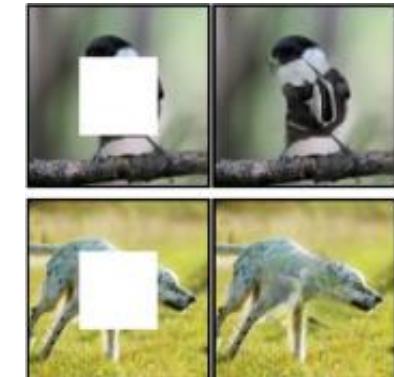


Image Inpainting



Pose Guided Person Image Generation

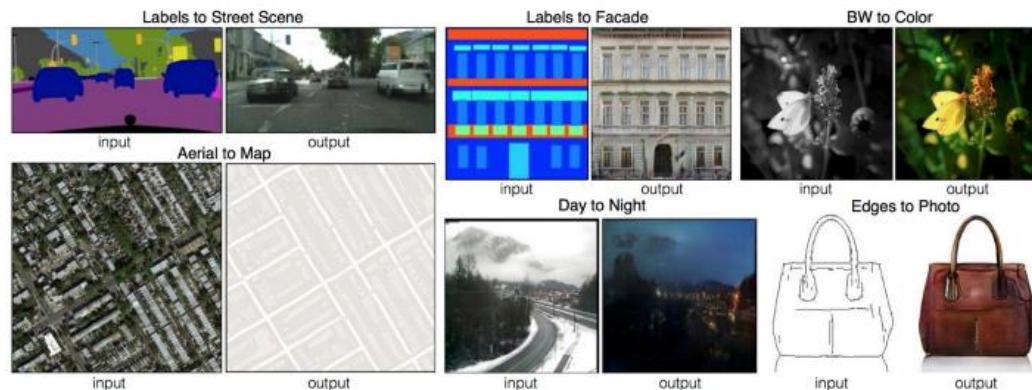
# Recent GAN Applications



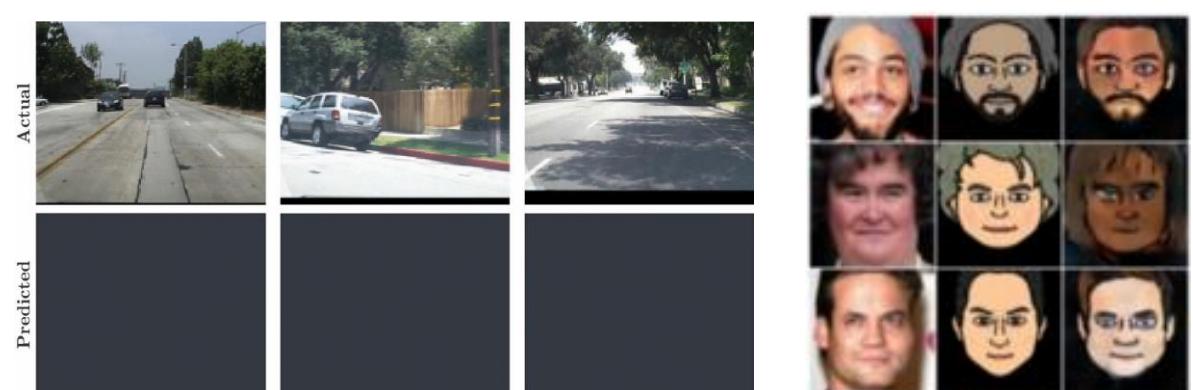
Fake Celebrities



Text to Image



Pix2Pix



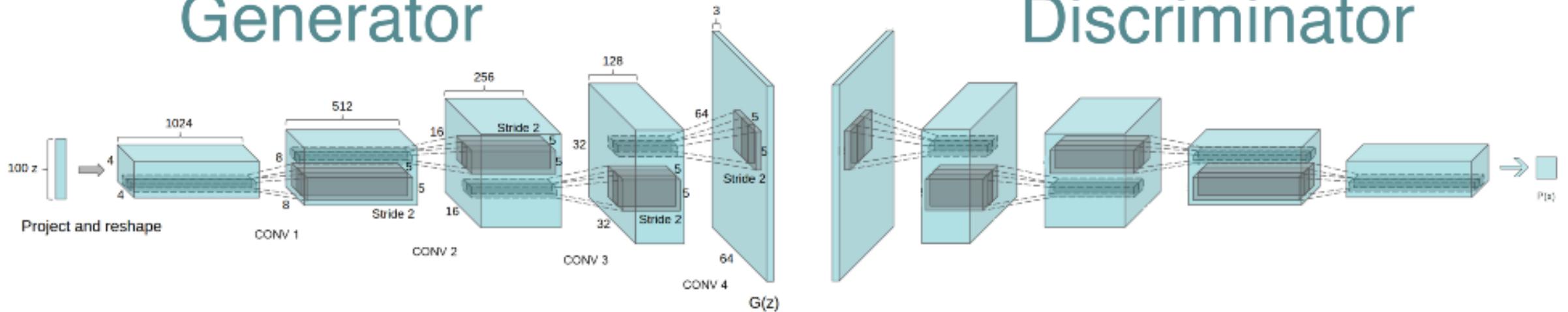
Next Frame Prediction



Emoji Generator

# Generator vs Discriminator

## Generator



Objective of the **Generator** is to  
**cheat the Discriminator** by  
simulating **Realistic** images

## Discriminator



Objective of the **Discriminator**  
is to find out whether image is  
**Real or Fake**

# Training a GAN

**Pdata(x)** - Distribution of real data

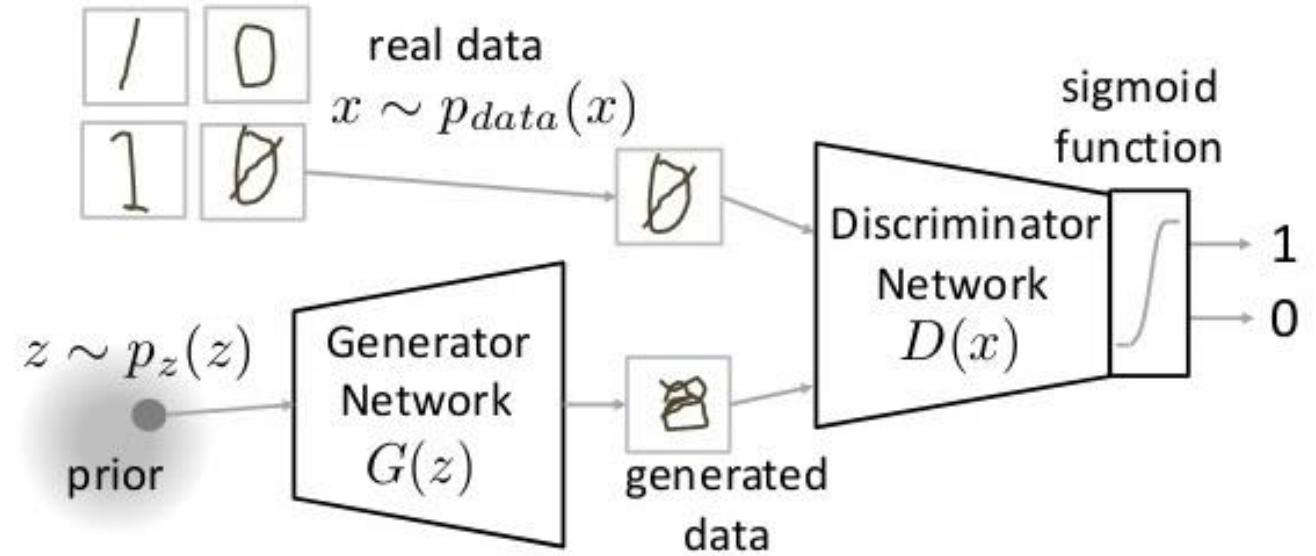
X - sample from pdata(x)

**P(z)** - distribution of generator

Z - sample from p(z)

**G(z)** - Generator Network

**D(x)** - Discriminator Network



GANs objective is to solving a minimax problem

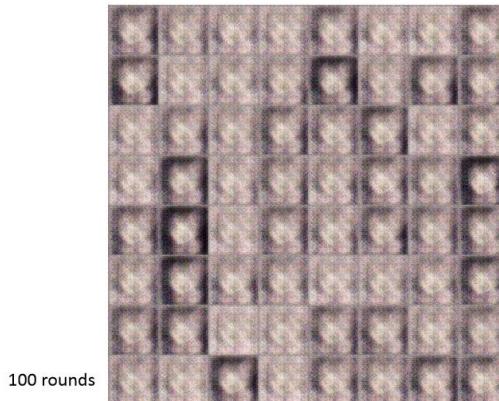
$$\min_G \max_D V(D, G)$$

$$V(D, G) = \mathbb{E}_{x \sim p_{data}(x)} [\log D(x)] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))]$$

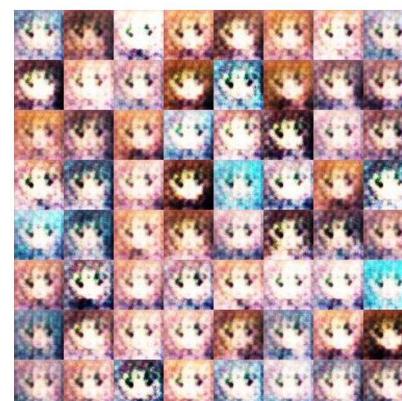
# Training a GAN

- In Function  $V(D, G)$ , the **first term** is entropy that the data from real distribution ( $p_{\text{data}}(x)$ ) passes through the discriminator (aka best case scenario). The discriminator tries to maximize this to 1.
- The **second term** is entropy that the data from random input ( $p(z)$ ) passes through the generator, which then generates a fake sample which is then passed through the discriminator to identify the fakeness (aka worst case scenario). In this term, discriminator tries to maximize it to 0 (i.e. the log probability that the data from generated is fake is equal to 0).
- **So overall, the discriminator is trying to maximize the function  $V$ .** On the other hand, **the task of generator is exactly opposite, i.e. it tries to minimize the function  $V$**  so that the differentiation between real and fake data is bare minimum.
- This, in other words is a cat and mouse game between generator and discriminator!

# GAN Training Example



100 rounds



1000 rounds



2000 rounds



5000 rounds



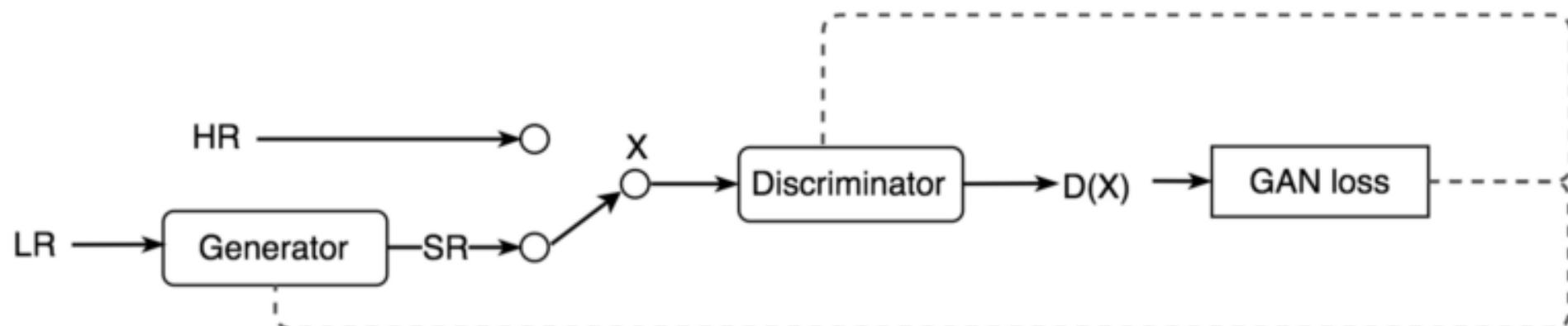
10,000 rounds



50,000 rounds

# GANs for Image Super-resolution: SRGAN

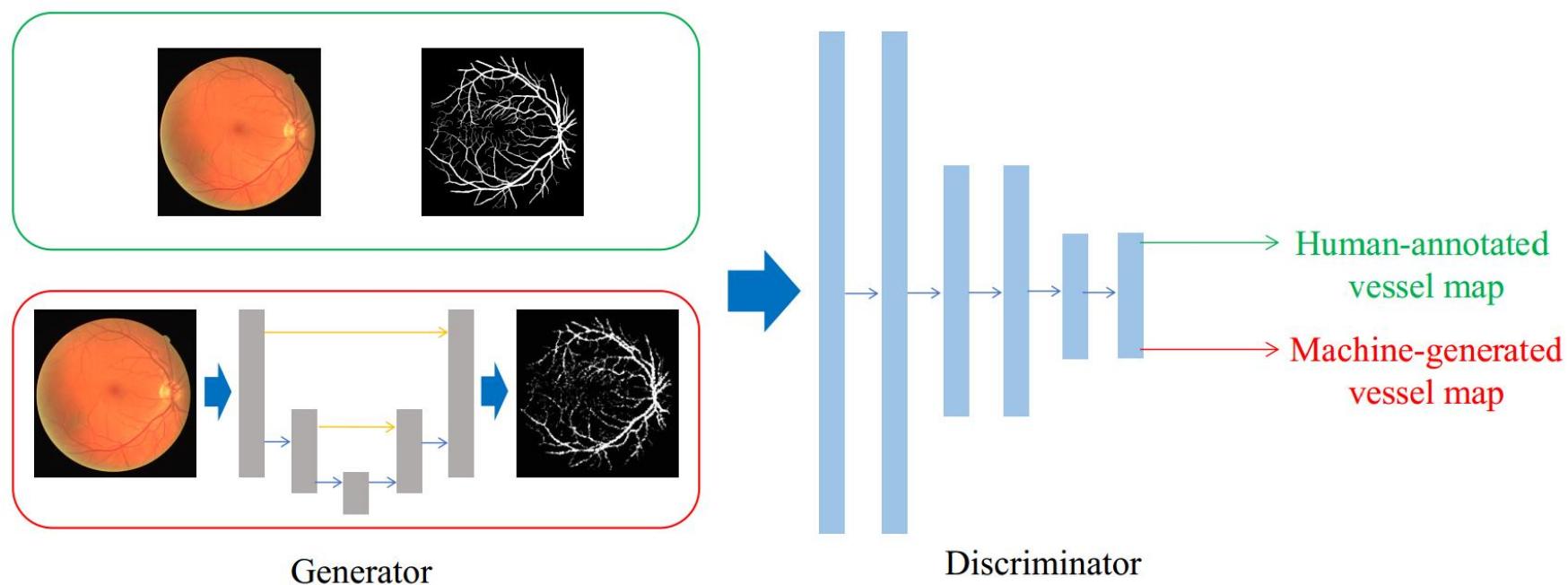
- Generator: gets low-resolution image as input and super-resolution image as output
- Discriminator: aims to differentiate between high-resolution image vs super-resolution image



Architecture of SRGAN

# GANs for Image Segmentation

- Recently GANs are highly applied for medical image segmentation
- Generator is a segmentation network where discriminator will predict whether it is real segmentation or generated from segmentation



Questions ?

Thanks