ECE 884 Deep Learning

Lecture 21: Autoencoder

04/13/2021

Review of last lecture

- Attention without RNN
- Self-Attention without RNN
- Transformer Model

Today’s lecture

- Autoregressive Model
- Autoencoder
- Variational Autoencoder

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Classification

Cat

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Unsupervised Learning

Data: x

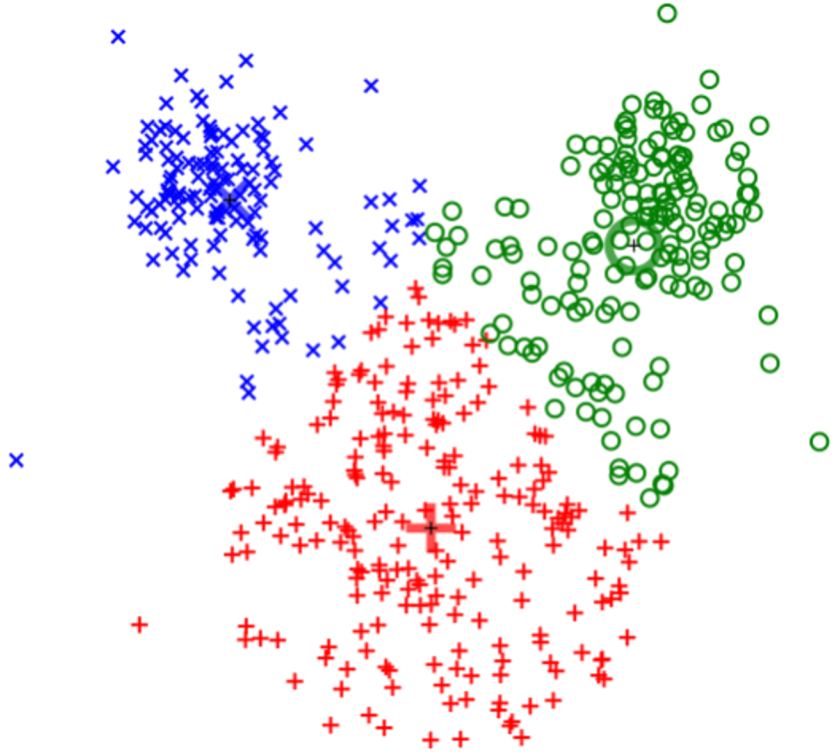
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Supervised vs Unsupervised Learning

Clustering
(e.g. K-Means)



Unsupervised Learning

Data: x

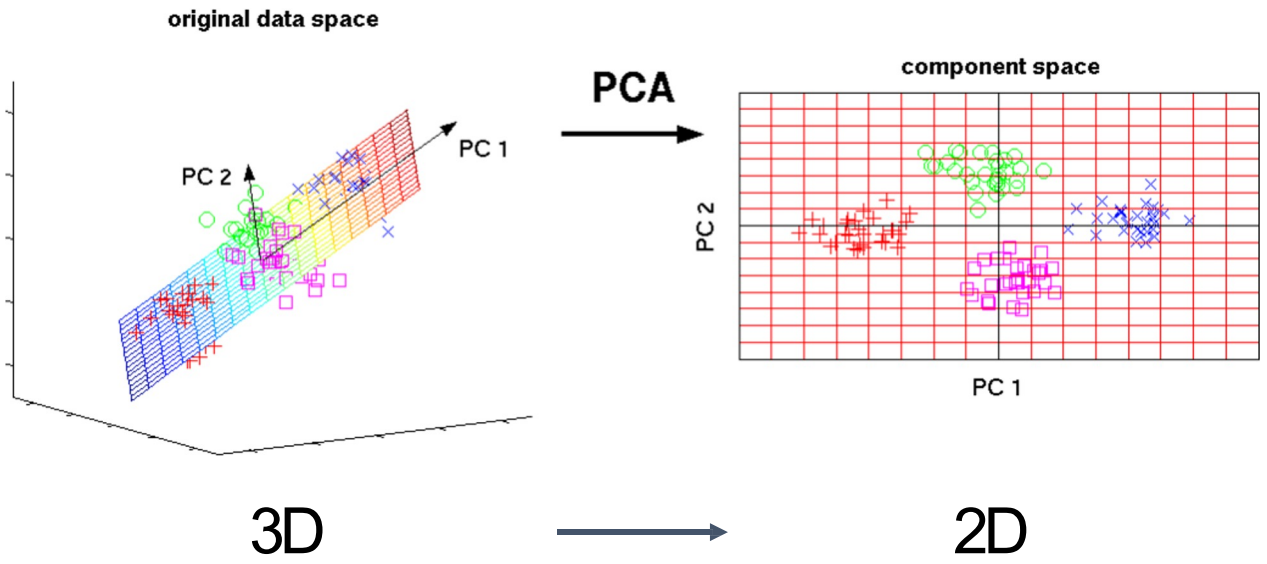
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Supervised vs Unsupervised Learning

Dimensionality Reduction (e.g. Principal Components Analysis)



Unsupervised Learning

Data: x

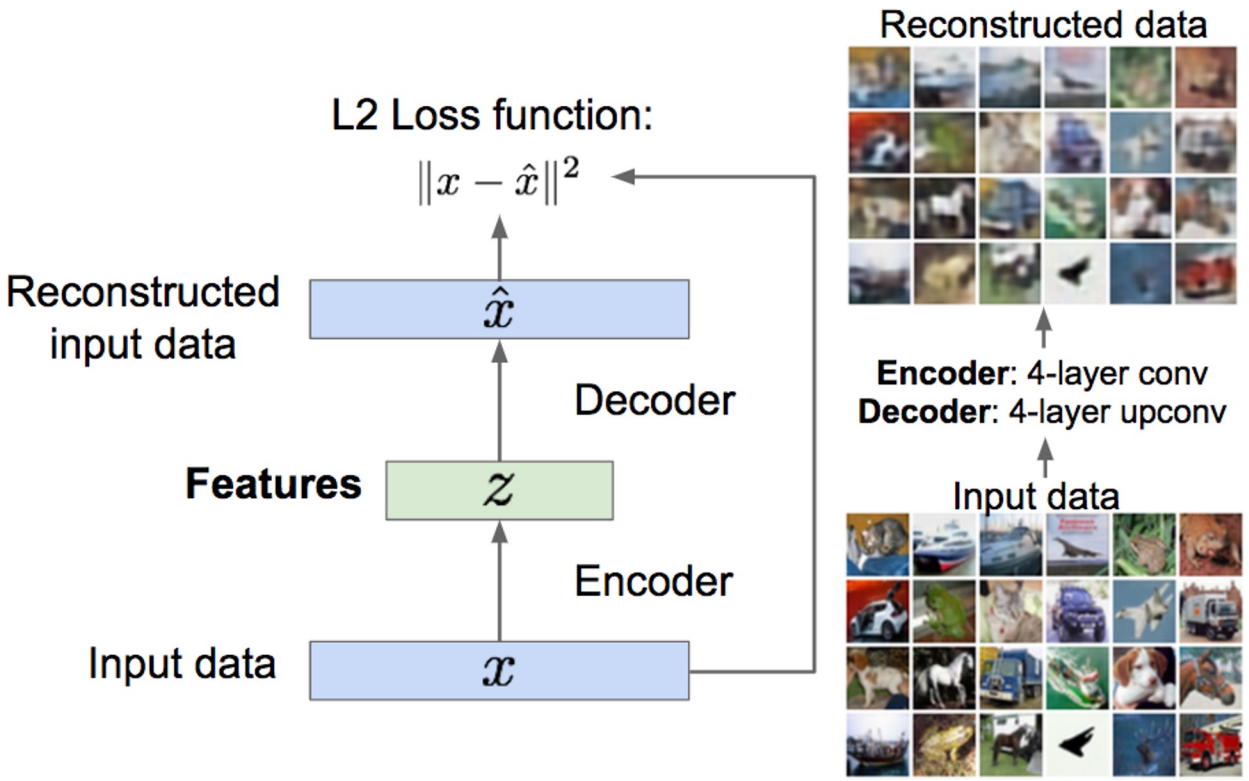
Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Supervised vs Unsupervised Learning

Feature Learning (e.g. autoencoders)



Unsupervised Learning

Data: x

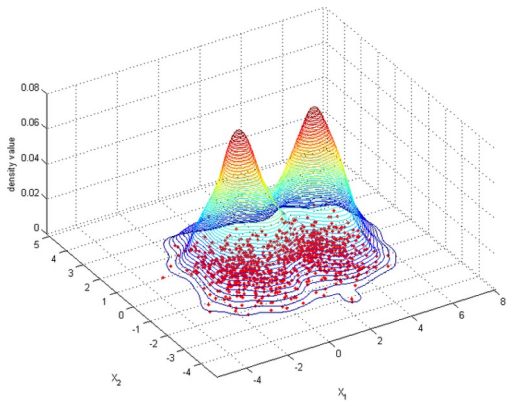
Just data, no labels!

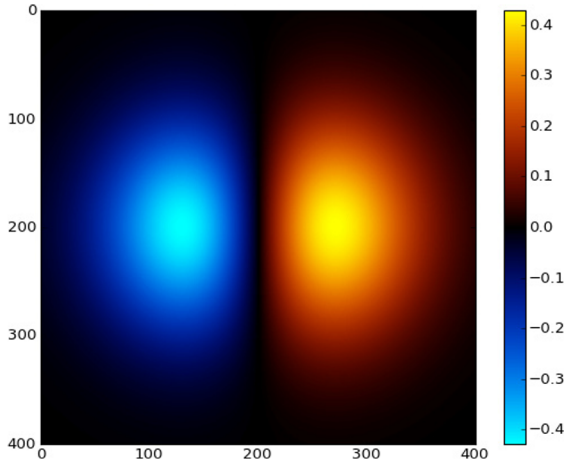
Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Supervised vs Unsupervised Learning

Density Estimation





Images left and right are CC0 public domain

Unsupervised Learning

Data: x

Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map x -> y

Examples: Classification, regression, object detection, semantic segmentation, image captioning, etc.

Unsupervised Learning

Data: x

Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, feature learning, density estimation, etc.

Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

Data: x

Label: y

Cat

Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

Data: x

Label: y

Cat

Probability Recap:

Density Function

p(x) assigns a positive number to each possible x; higher numbers mean x is more likely

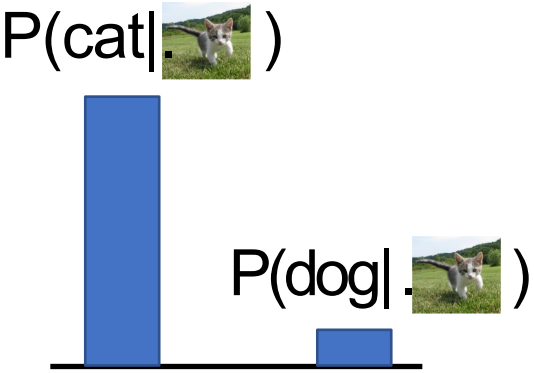
Density functions are **normalized**:

$$\int_X p(x)dx = 1$$

Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Data: x



Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

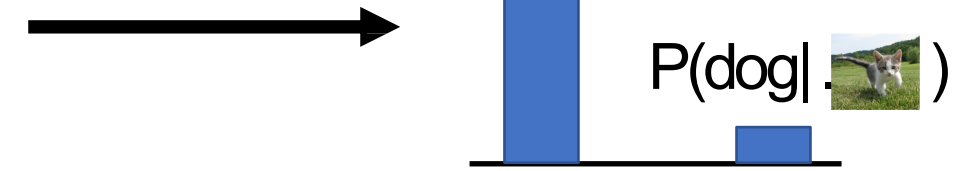
Density Function
p(x) assigns a positive number to each possible x; higher numbers mean x is more likely

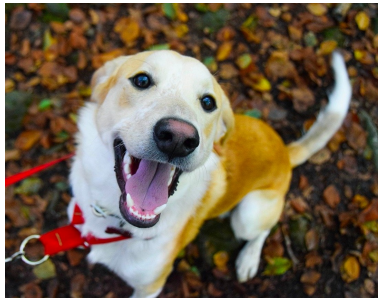
Density functions are **normalized**:

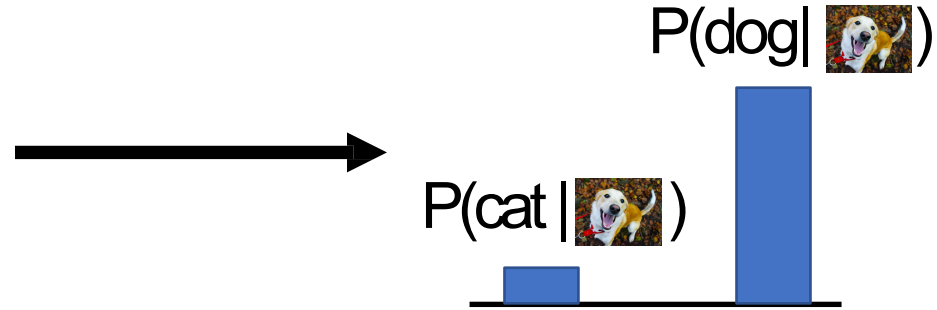
$$\int_X p(x)dx = 1$$

Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)







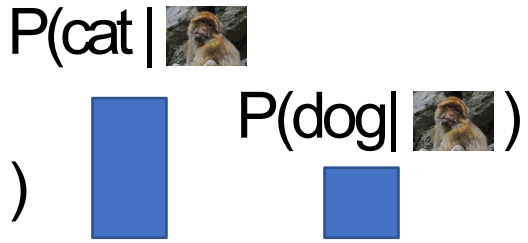
Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

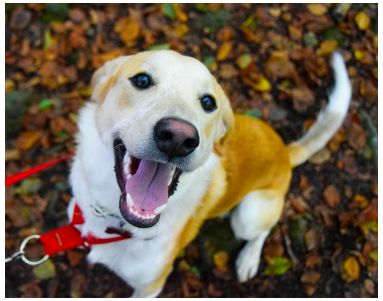
Discriminative model: the possible labels for each input "compete" for probability mass. But no competition between **images**

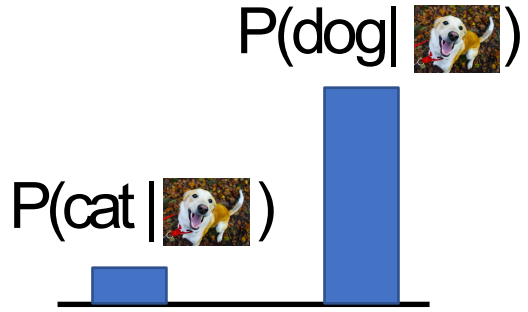
Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)



Generative Model: Learn a probability distribution p(x)



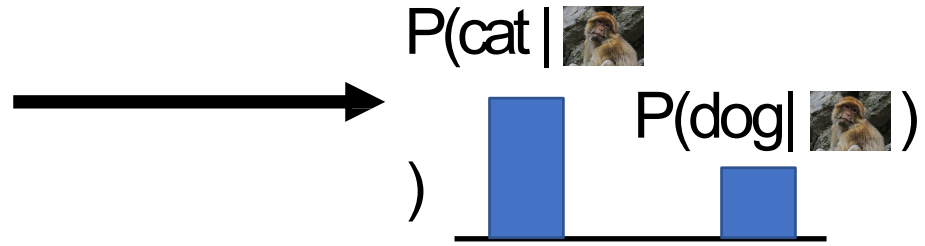


Conditional Generative Model: Learn p(x|y)

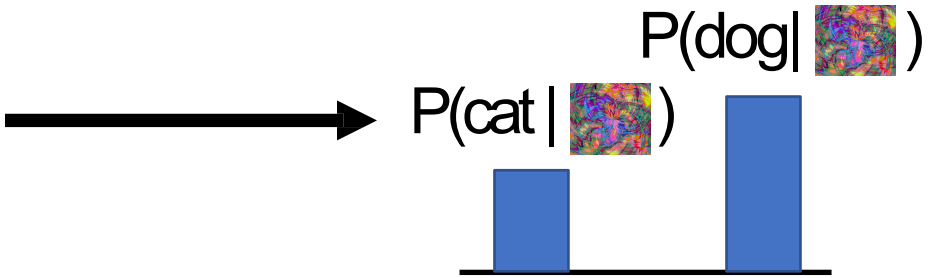
Discriminative model: No way for the model to handle unreasonable inputs; it must give label distributions for all images

Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)



Generative Model: Learn a probability distribution p(x)



Conditional Generative Model: Learn p(x|y)

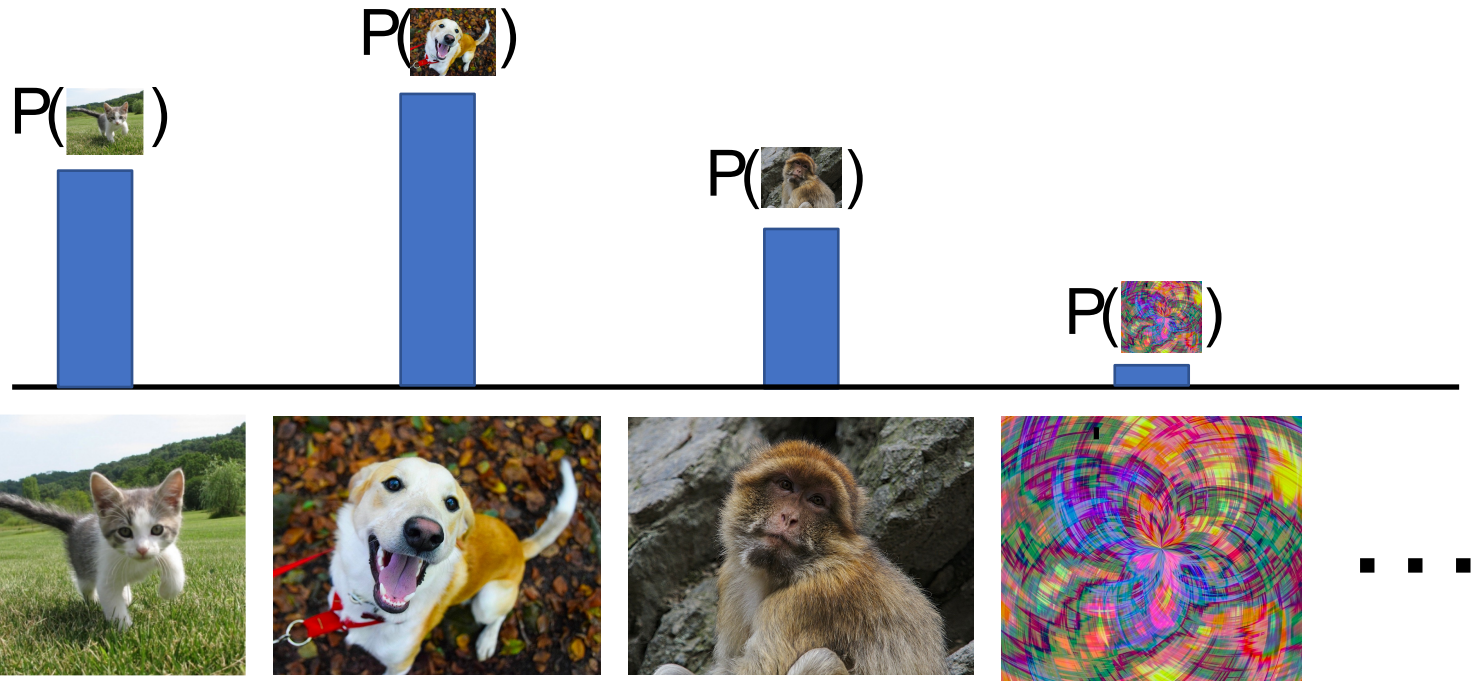
Discriminative model: No way for the model to handle unreasonable inputs; it must give label distributions for all images

Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



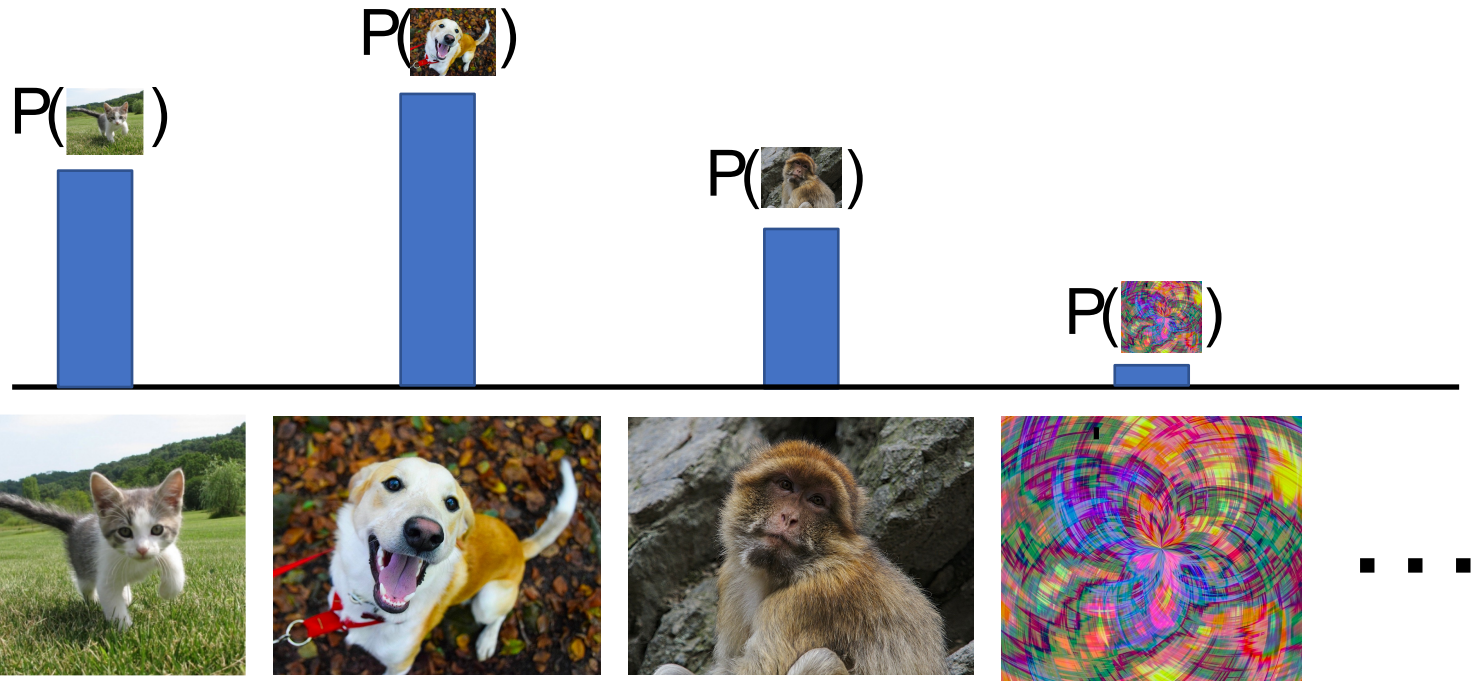
Generative model: All possible images compete with each other for probability mass

Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



Generative model: All possible images compete with each other for probability mass

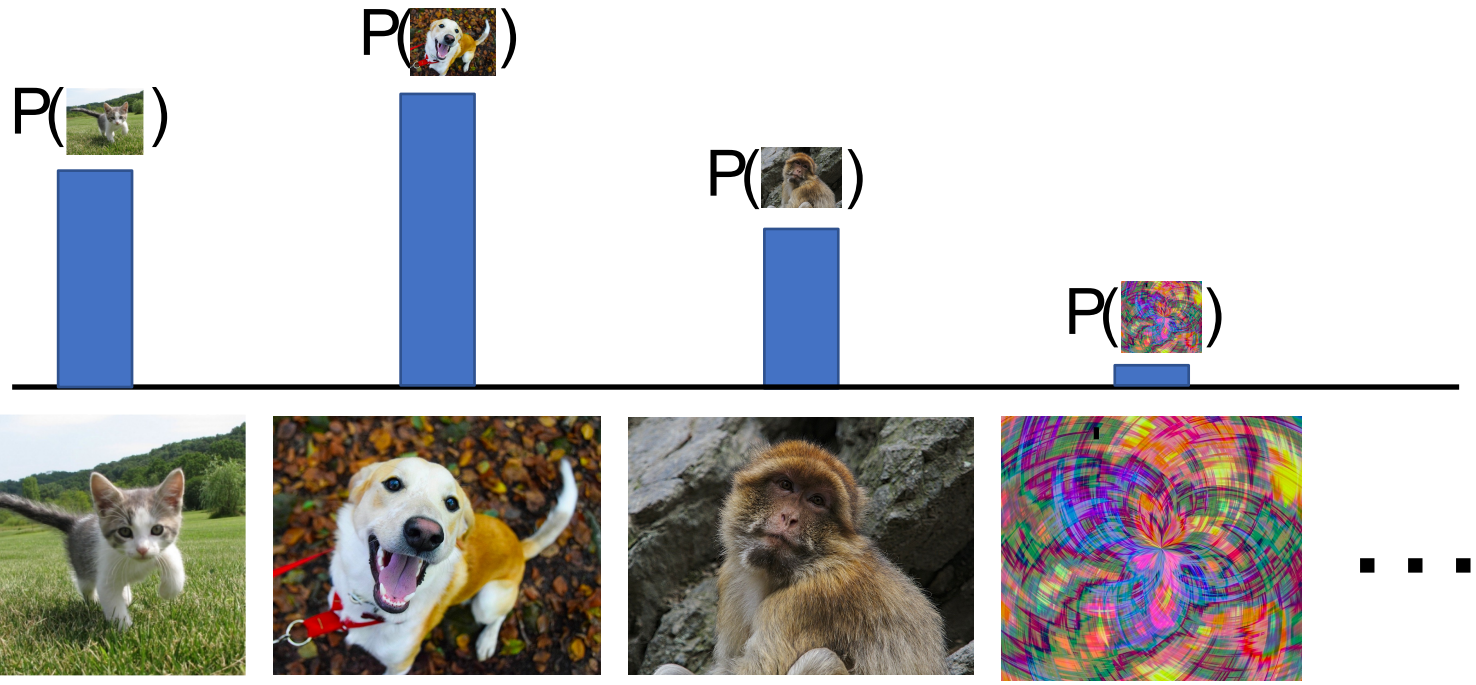
Requires deep image understanding! Is a dog more likely to sit or stand? How about 3-legged dog vs 3-armed monkey?

Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



Generative model: All possible images compete with each other for probability mass

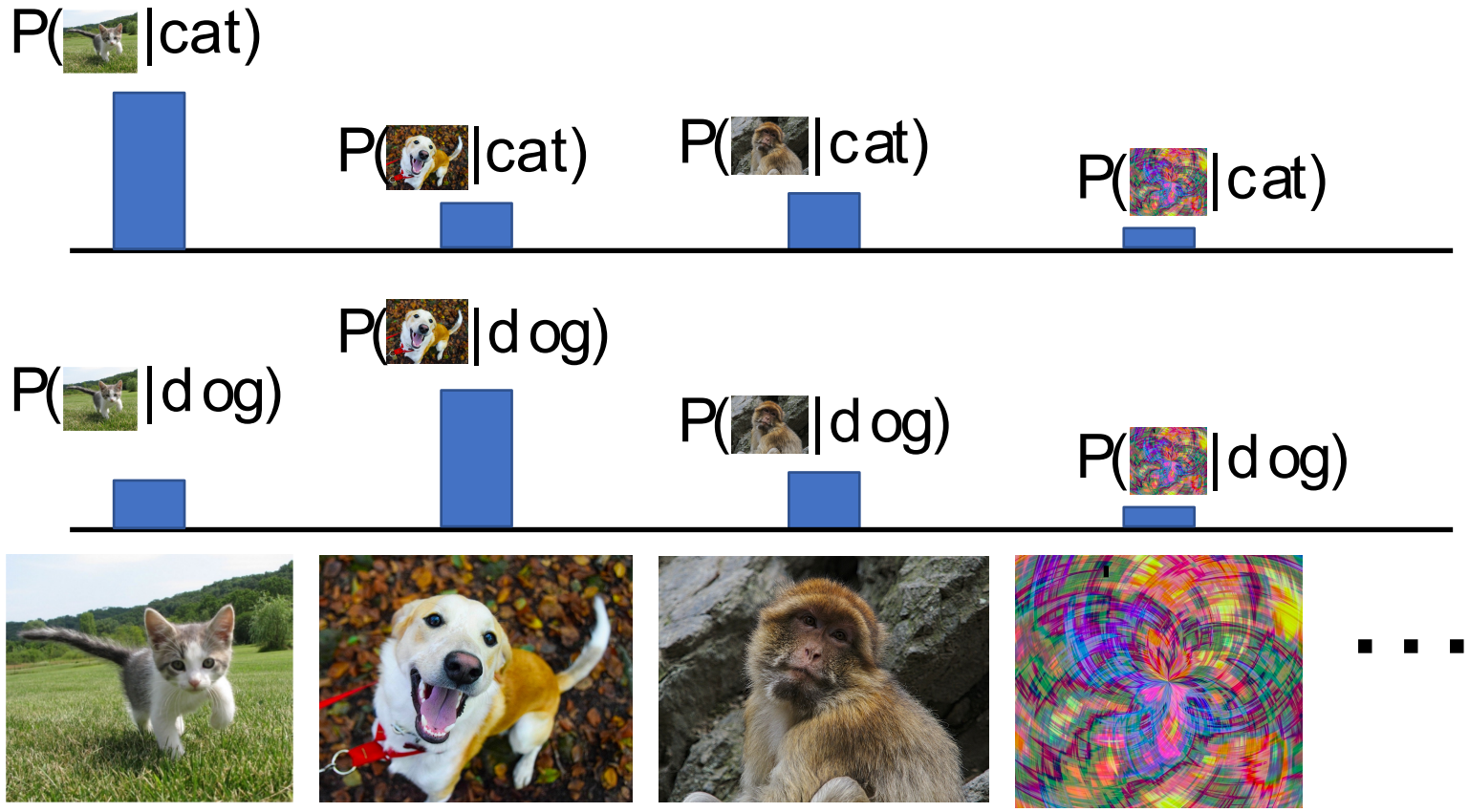
Model can "reject" unreasonable inputs by assigning them small values

Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)



Conditional Generative Model: Each possible label induces a competition among all images

Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

Recall **Bayes' Rule:**

$$P(x \mid y) = \frac{P(y \mid x)}{P(y)} P(x)$$

Discriminative vs Generative Models

Discriminative Model: Learn a probability distribution p(y|x)

Generative Model: Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

Recall **Bayes' Rule:**

$$P(x \mid y) = \frac{P(y \mid x)}{P(y)} P(x)$$

Conditional Generative Model

Discriminative Model

Prior over labels

(Unconditional) Generative Model

We can build a conditional generative model from other components!

What can we do with a discriminative model?

Discriminative Model:
Learn a probability distribution p(y|x)

Assign labels to data
Feature learning (with labels)

Generative Model:
Learn a probability distribution p(x)

Conditional Generative Model: Learn p(x|y)

What can we do with a generative model?

Discriminative Model:
Learn a probability distribution p(y|x)

→ Assign labels to data
Feature learning (with labels)

Generative Model:
Learn a probability distribution p(x)

→ Detect outliers
Feature learning (without labels)
Sample to **generate** new data

Conditional Generative Model: Learn p(x|y)

What can we do with a generative model?

Discriminative Model:
Learn a probability distribution p(y|x)

→ Assign labels to data
Feature learning (with labels)

Generative Model:
Learn a probability distribution p(x)

→ Detect outliers
Feature learning (without labels)
Sample to **generate** new data

Conditional Generative Model: Learn p(x|y)

→ Assign labels, while rejecting outliers!
Generate new data conditioned on input labels

Taxonomy of Generative Models

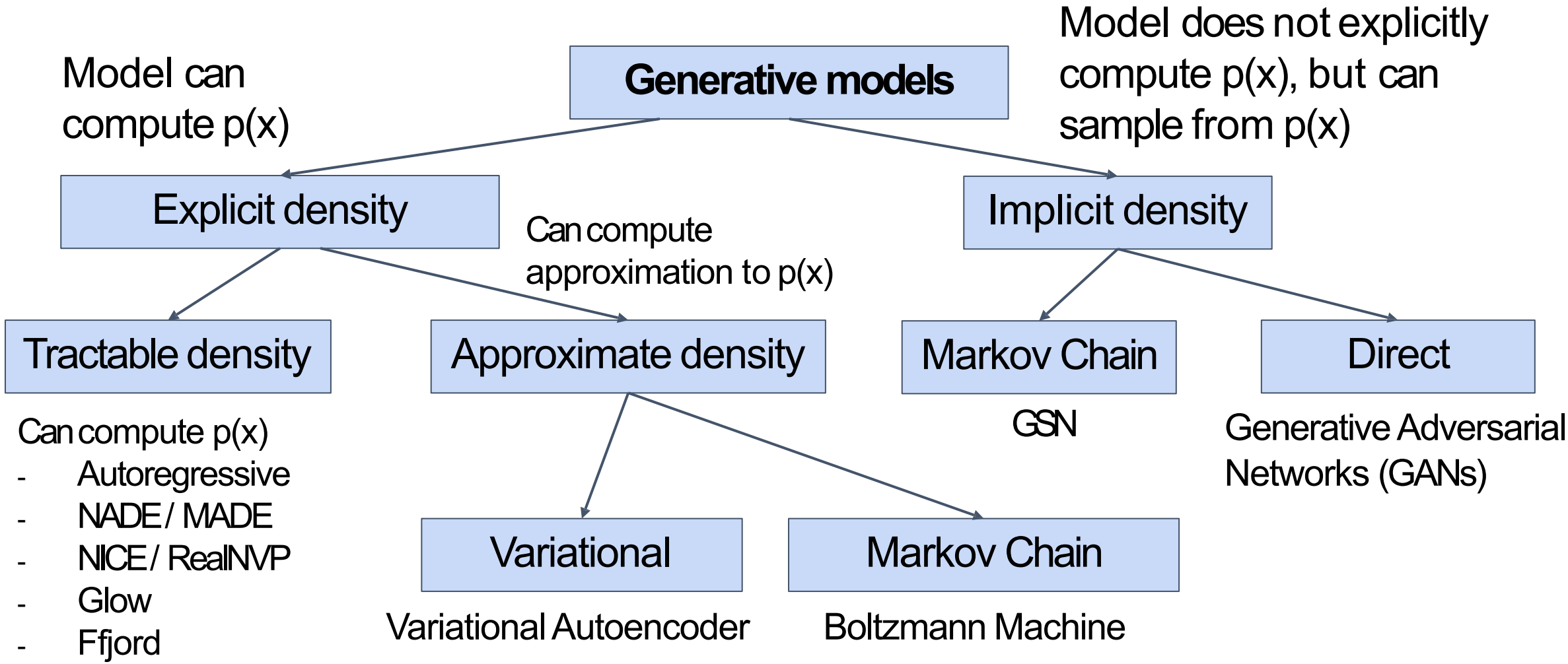


Figure adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Taxonomy of Generative Models

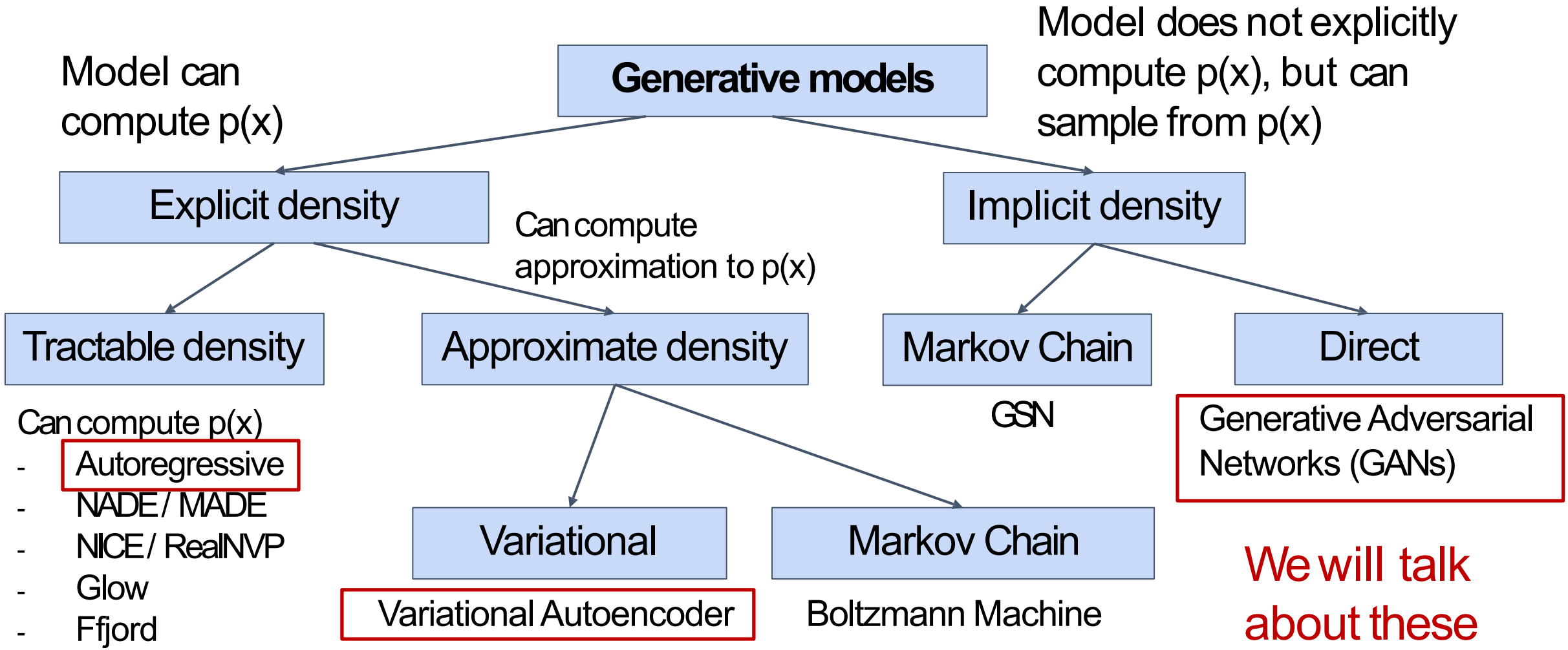


Figure adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Autoregressive models

Explicit Density Estimation

Goal: Write down an explicit function for $p(x) = f(x, W)$

Given dataset $x^{(1)}, x^{(2)}, \ldots x^{(N)}$, train the model by solving:

$$W^* = \arg\max_W \prod_i p(x^{(i)})$$

Maximize probability of training data
(Maximum likelihood estimation)

$$= \arg\max_W \sum_i \log p(x^{(i)})$$

Log trick to exchange product for sum

$$= \arg\max_W \sum_i \log f(x^{(i)}, W)$$

This will be our loss function!
Train with gradient descent

Explicit Density: Autoregressive Models

Goal: Write down an explicit function for $p(x) = f(x, W)$

Assume x consists of multiple subparts:

$$x = (x_1, x_2, x_3, \ldots, x_T)$$

Explicit Density: Autoregressive Models

Goal: Write down an explicit function for $p(x) = f(x, W)$

Assume x consists of multiple subparts:

$$x = (x_1, x_2, x_3, \ldots, x_T)$$

Break down probability using the chain rule:

$$\begin{aligned} p(x) &= p(x_1, x_2, x_3, \ldots, x_T) \\ &= p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \ldots \\ &= \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1}) \end{aligned}$$

Probability of the next subpart given all the previous subparts

Explicit Density: Autoregressive Models

Goal: Write down an explicit function for $p(x) = f(x, W)$

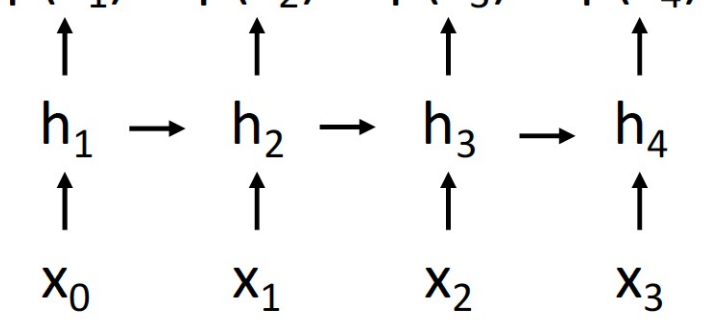
Assume x consists of multiple subparts:

$$x = (x_1, x_2, x_3, \ldots, x_T)$$

Break down probability using the chain rule:

$$\begin{aligned} p(x) &= p(x_1, x_2, x_3, \ldots, x_T) \\ &= p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_1, x_2) \ldots \\ &= \prod_{t=1}^{T} p(x_t \mid x_1, \ldots, x_{t-1}) \end{aligned}$$

Probability of the next subpart given all the previous subparts



We've already seen this! Language modeling with an RNN!

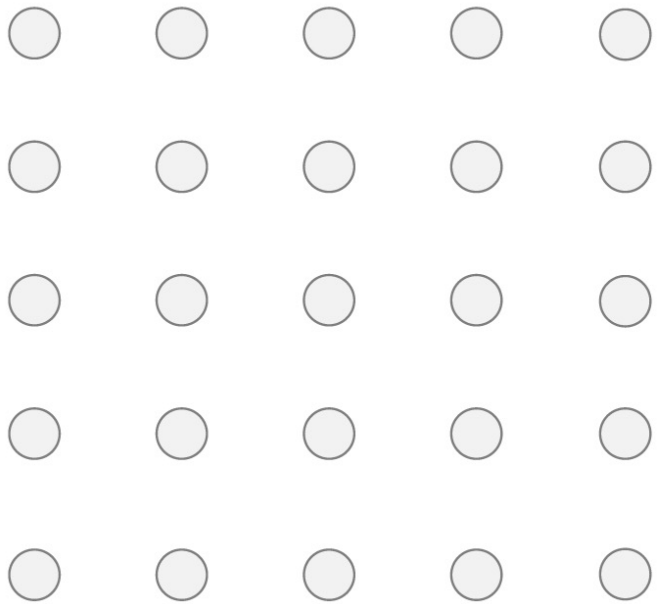
PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]



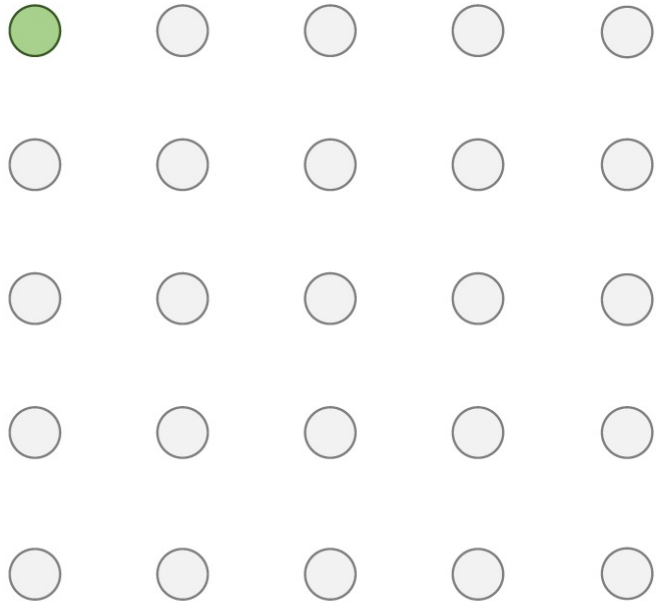
PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]



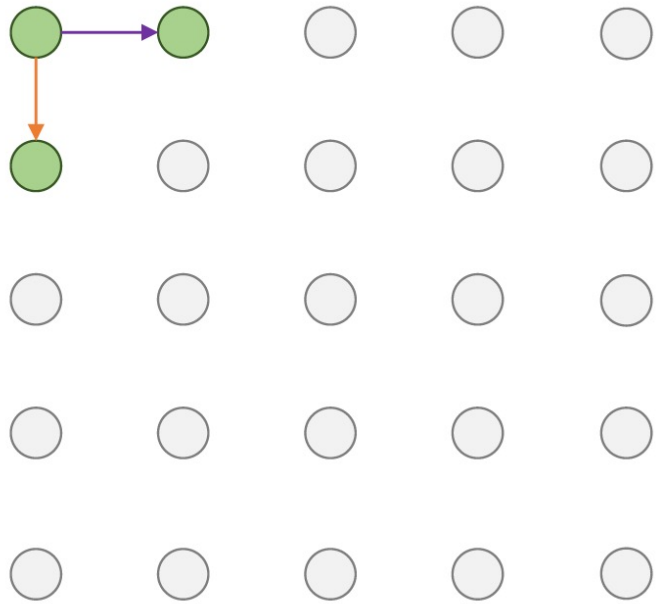
PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]



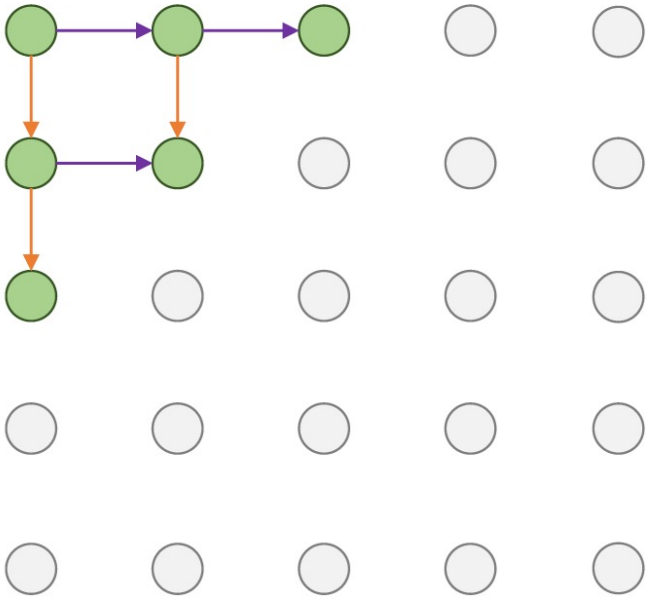
PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]



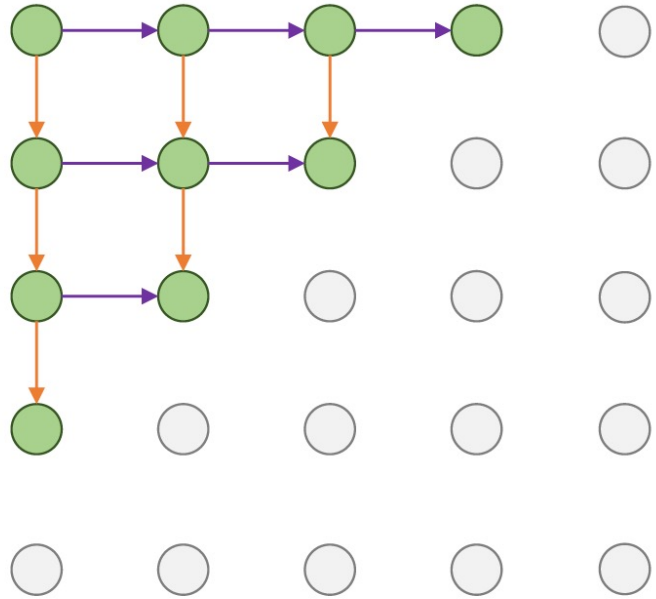
PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]



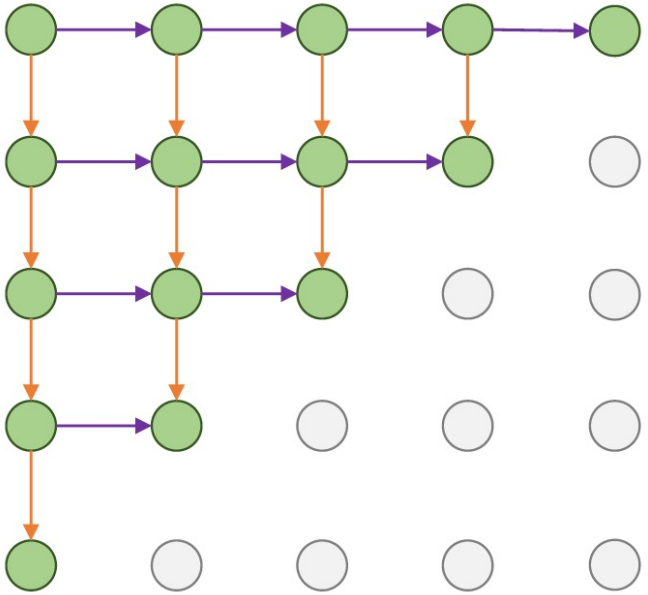
PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]



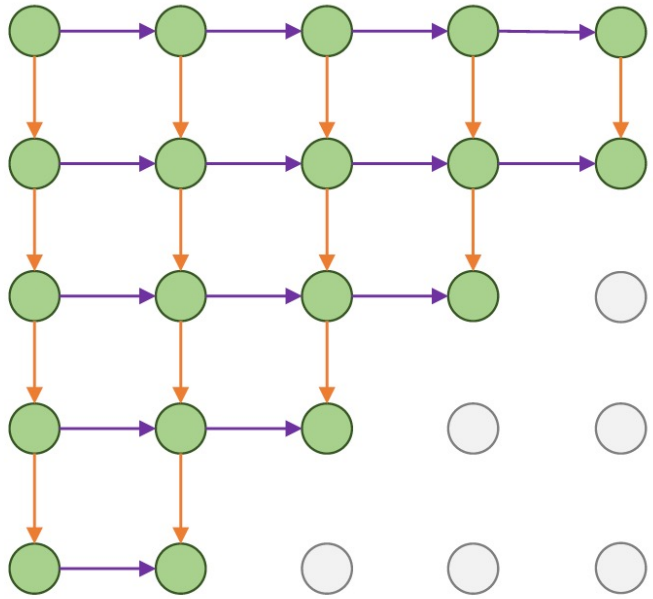
PixelRNN

Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]



PixelRNN

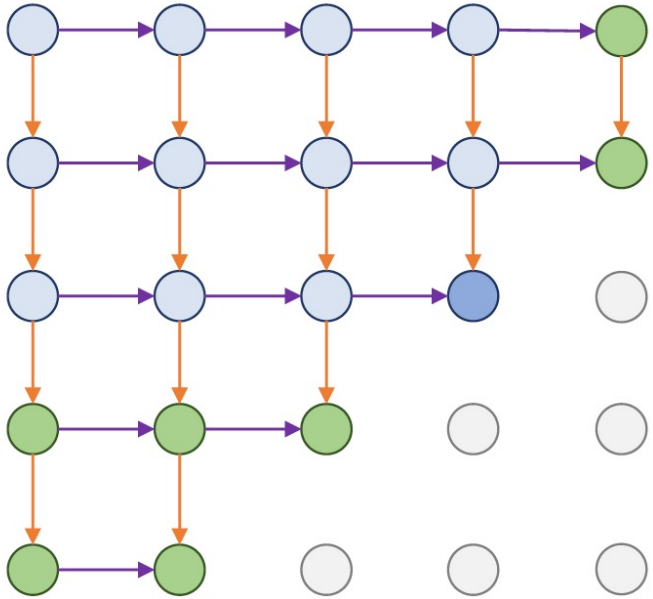
Generate image pixels one at a time, starting at the upper left corner

Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]

Each pixel depends **implicity** on all pixels above and to the left:



PixelRNN

Generate image pixels one at a time, starting at the upper left corner

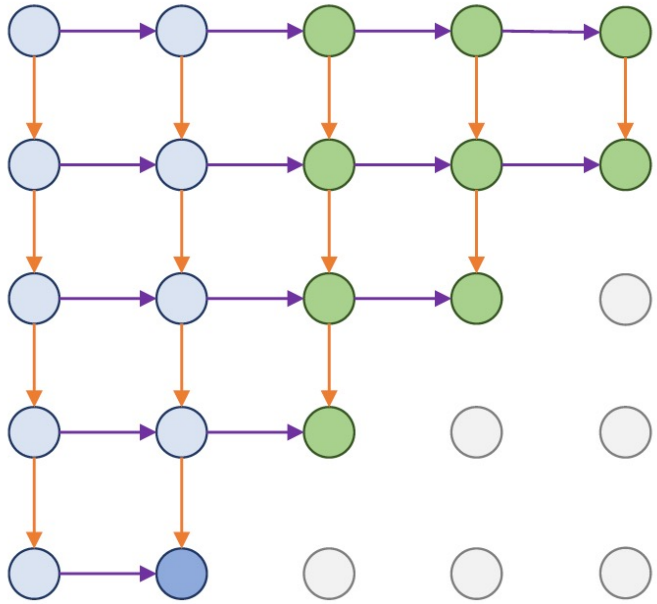
Compute a hidden state for each pixel that depends on hidden states and RGB values from the left and from above (LSTM recurrence)

$$h_{x,y} = f(h_{x-1,y}, h_{x,y-1}, W)$$

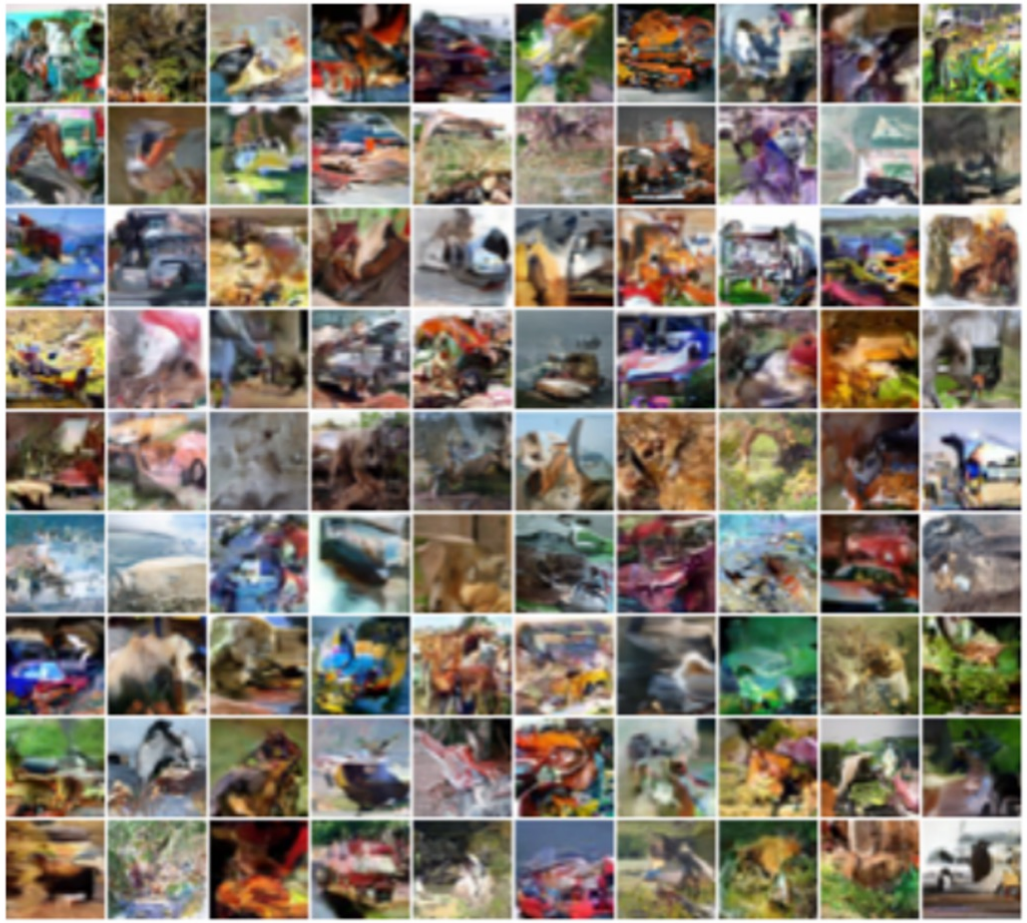
At each pixel, predict red, then blue, then green: softmax over [0, 1, …, 255]

Each pixel depends **implicity** on all pixels above and to the left:

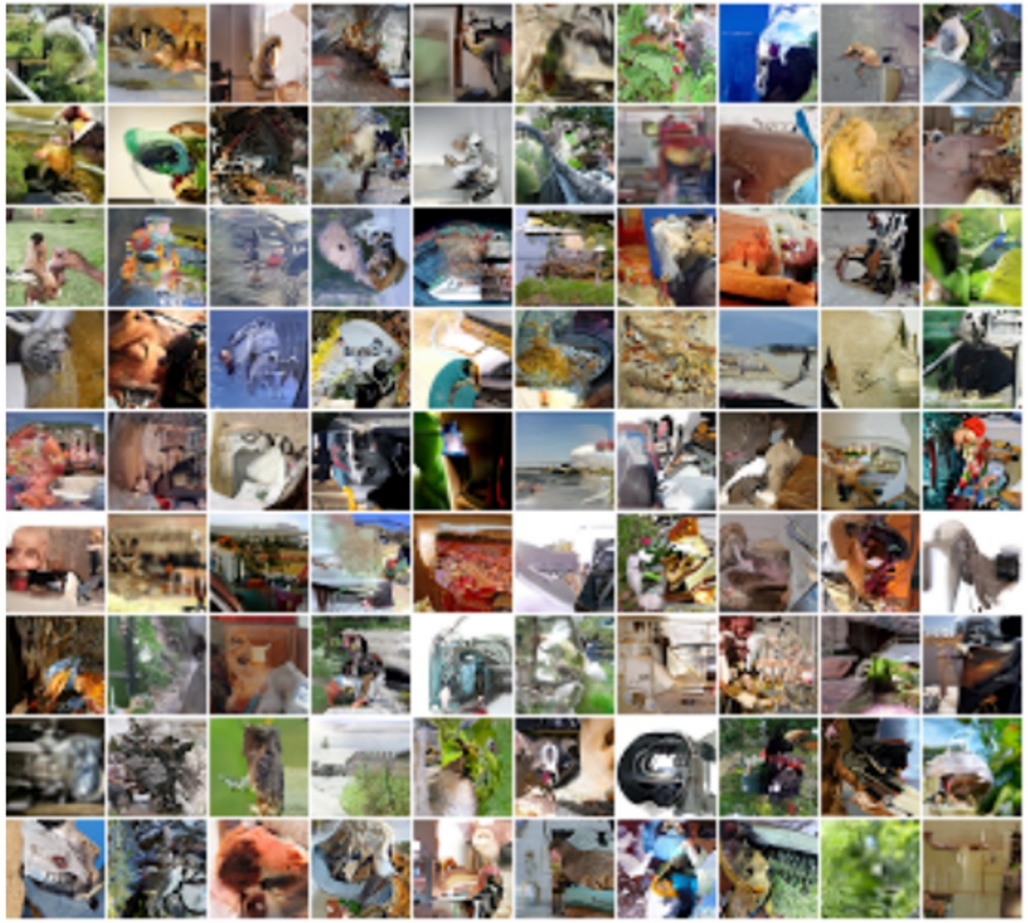
Problem: Very slow during both training and testing; N x N image requires 2N-1 sequential steps



PixelRNN: Generated Samples



32x32 CIFAR-10



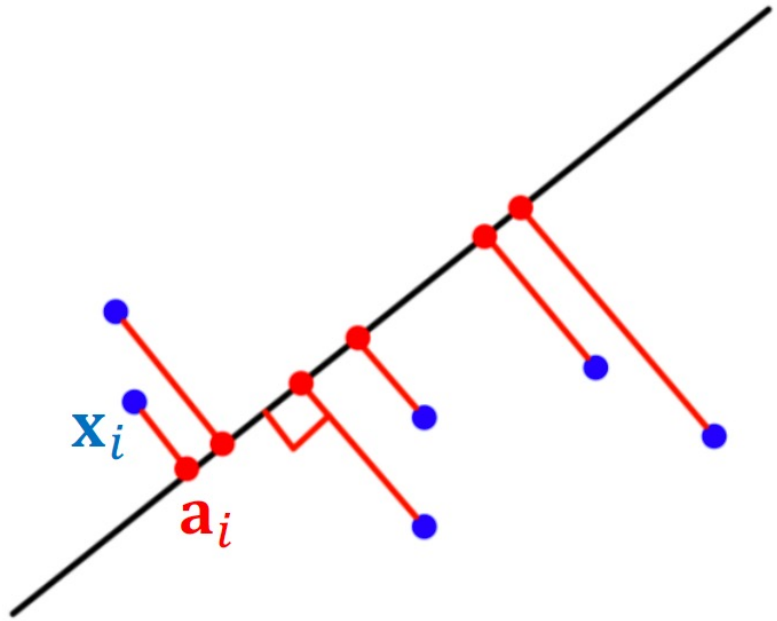
32x32 ImageNet

Autoencoders

Principal Component Analysis (PCA)

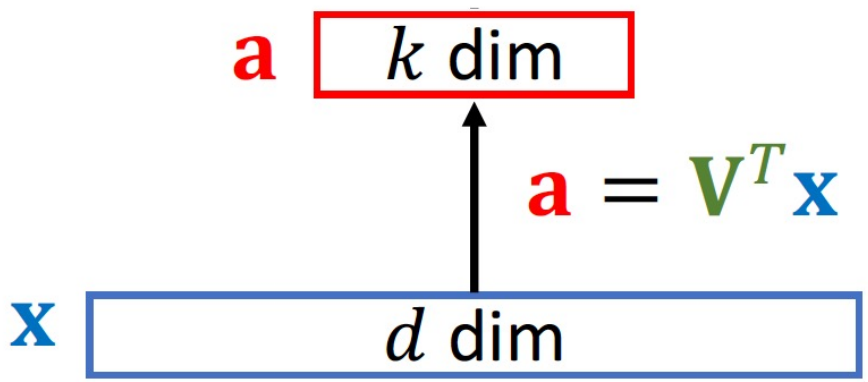
- Input: $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ (with zero-mean).

- Output: $\mathbf{a}_1, \cdots, \mathbf{a}_n \in \mathbb{R}^k$ $(k \ll d)$.



Principal Component Analysis (PCA)

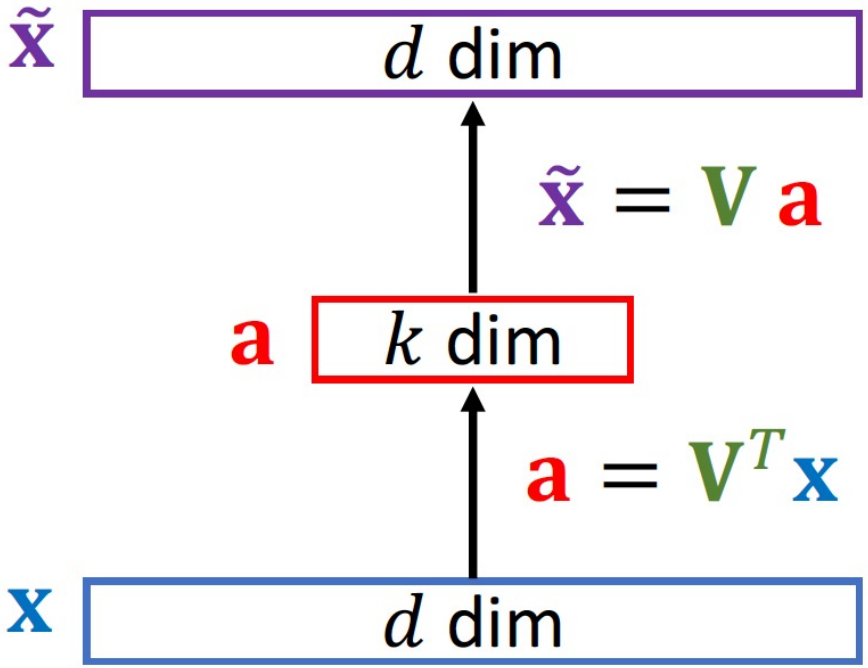
- Input: $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ (with zero-mean).
- Output: $\mathbf{a}_1, \cdots, \mathbf{a}_n \in \mathbb{R}^k$ $(k \ll d)$.
- PCA finds a $d \times k$ column orthogonal matrix $\mathbf{V}$



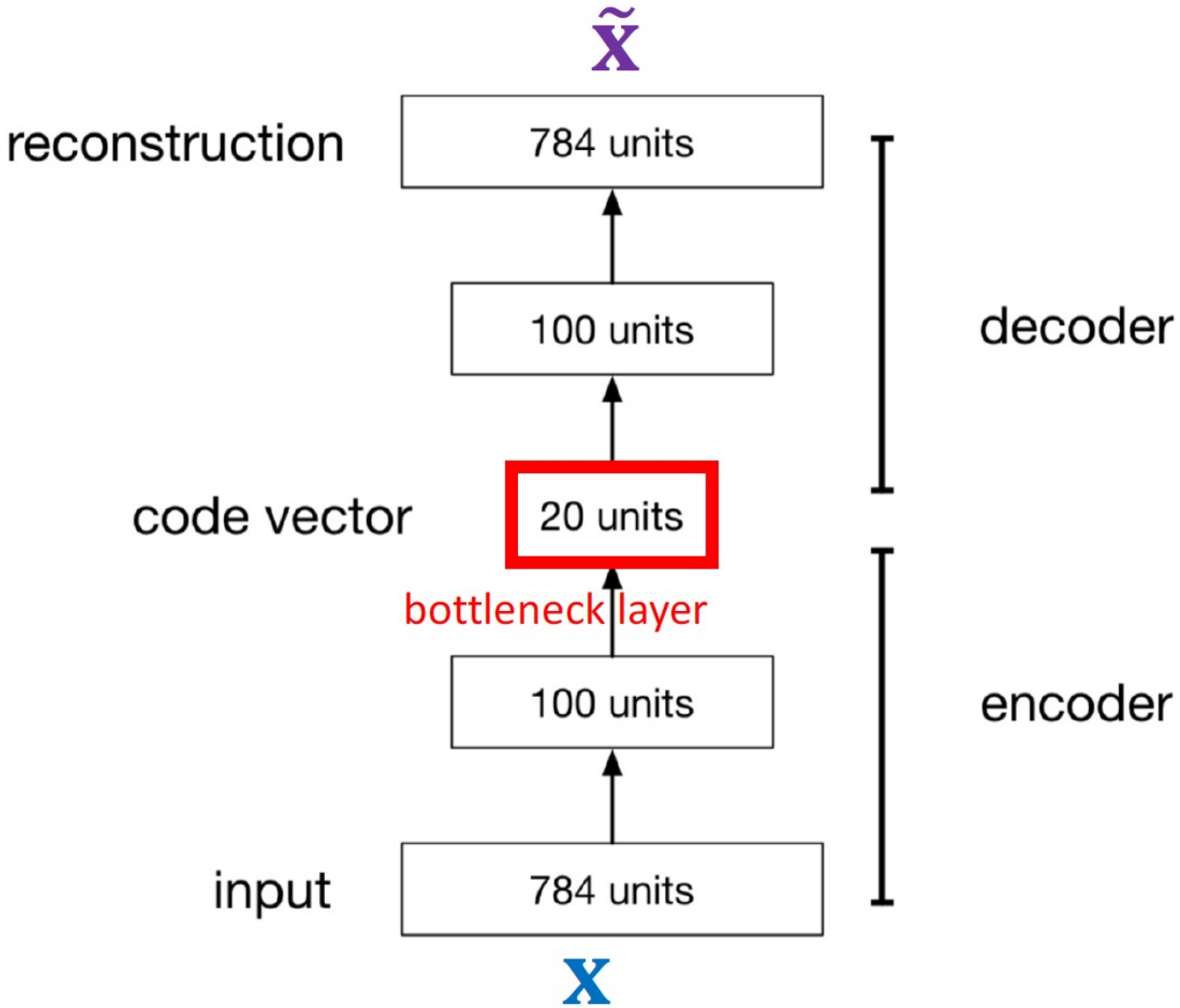
Principal Component Analysis (PCA)

- Input: $\mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d$ (with zero-mean).
- Output: $\mathbf{a}_1, \cdots, \mathbf{a}_n \in \mathbb{R}^k$ $(k \ll d)$.
- PCA finds a $d \times k$ column orthogonal matrix $\mathbf{V}$ such that

$$\min_{\mathbf{V}} \sum_{j=1}^{n} \left|\left| \mathbf{x}_j - \tilde{\mathbf{x}}_j \right|\right|_2^2, \quad \text{s.t. } \mathbf{V}^T\mathbf{V} = \mathbf{I}_k.$$



Autoencoder



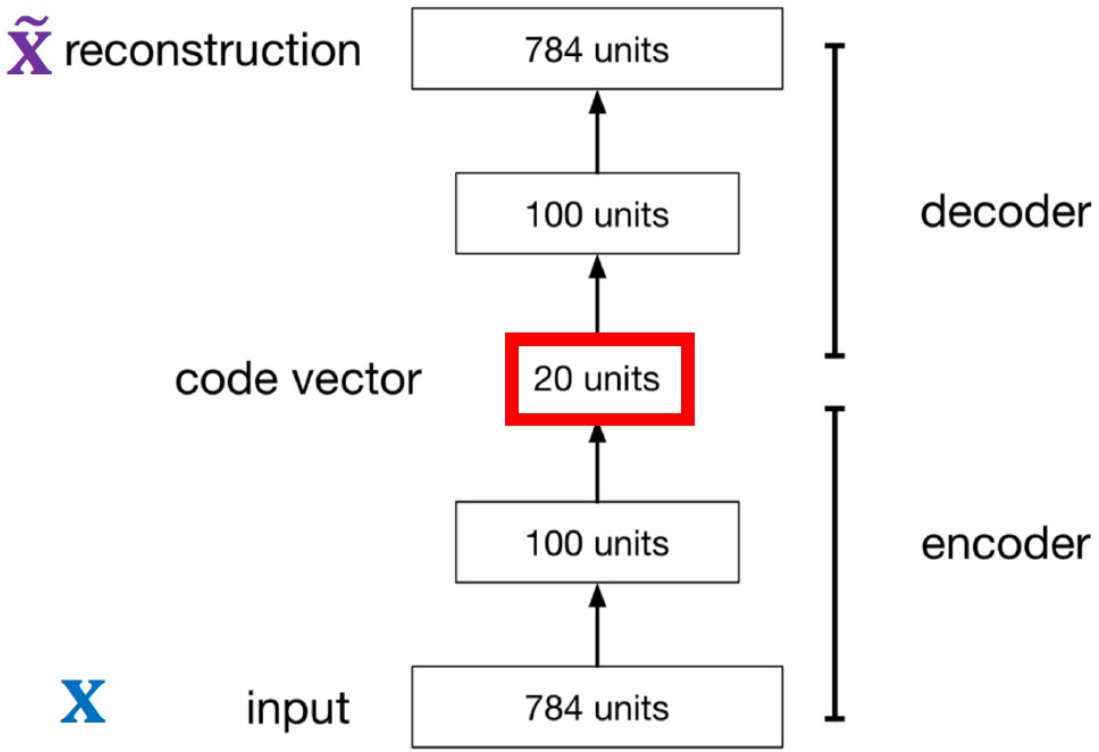
- An autoencoder is a neural net taking an input $\mathbf{x}$ and reconstruct $\mathbf{x}$.
- For dim reduction, we need a bottleneck layer whose dim is much smaller than the input.
- Loss function:

$$\sum_{j=1}^{n} \left|\left|\mathbf{x}_j - \tilde{\mathbf{x}}_j\right|\right|_2^2.$$

Autoencoder v.s. PCA

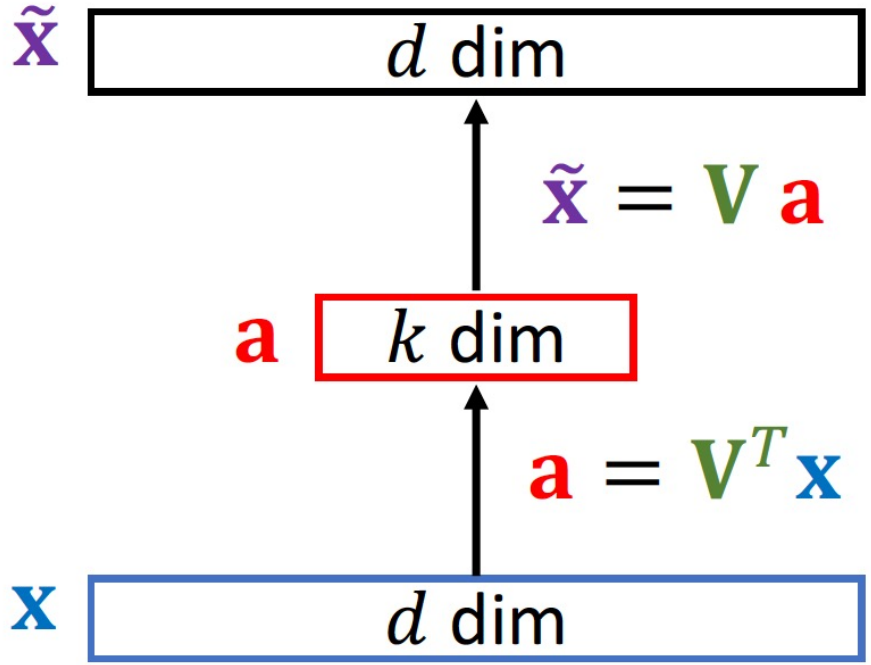
Autoencoder is nonlinear

$$\tilde{\mathbf{x}} = \text{decoder}\big(\text{encoder}(\mathbf{x})\big)$$



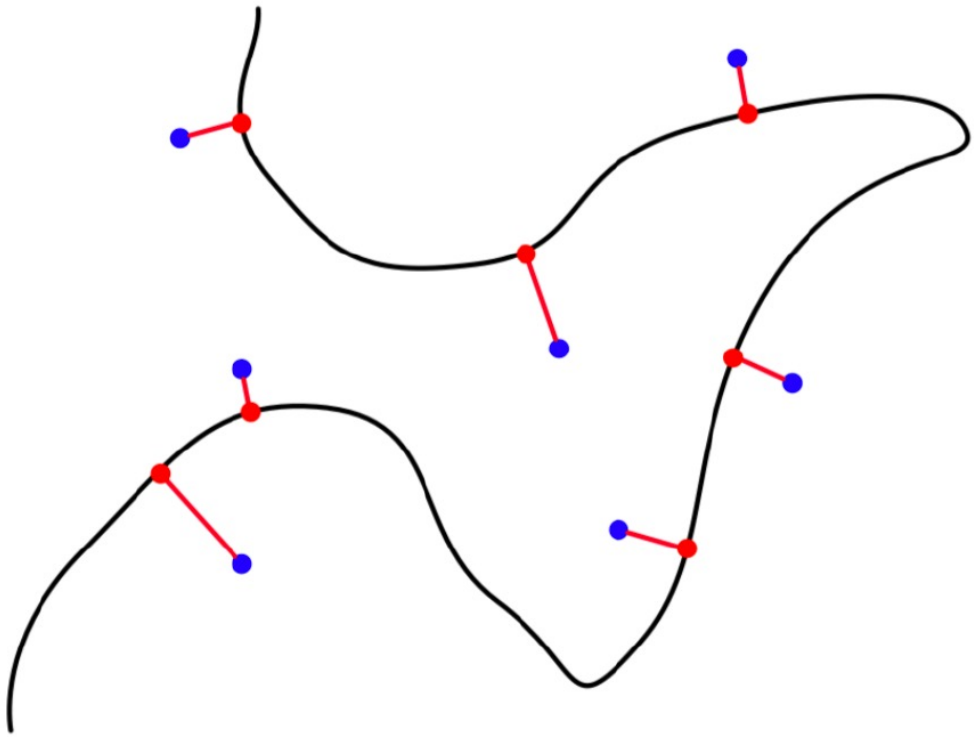
PCA is linear

$$\tilde{\mathbf{x}} = \mathbf{V}\mathbf{V}^T\mathbf{x}$$

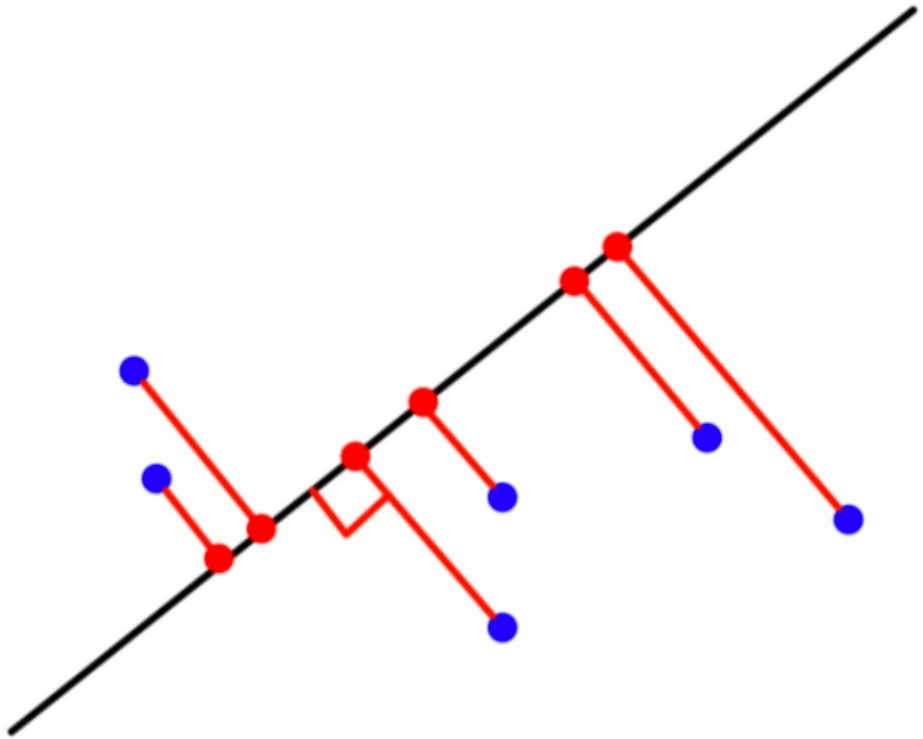


Autoencoder v.s. PCA

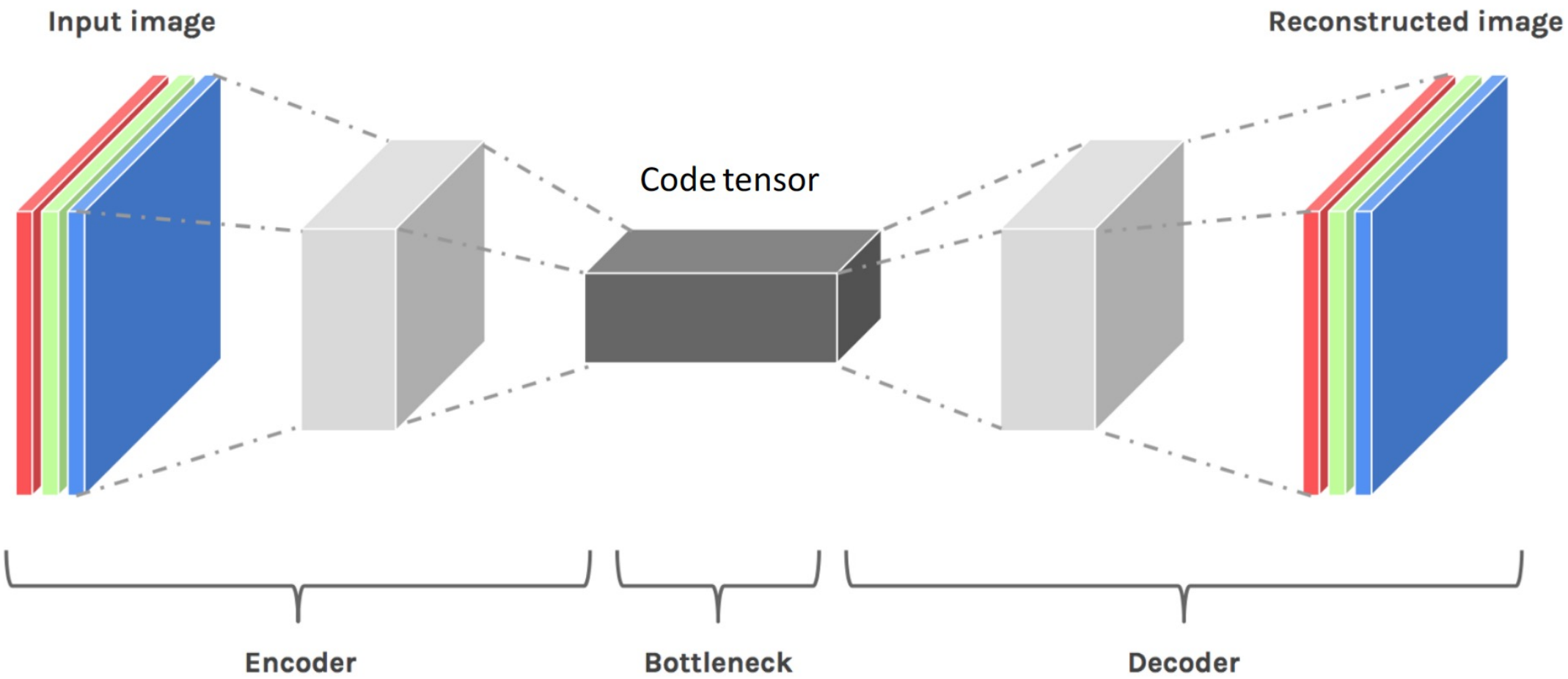
Autoencoder projects data onto nonlinear manifold.



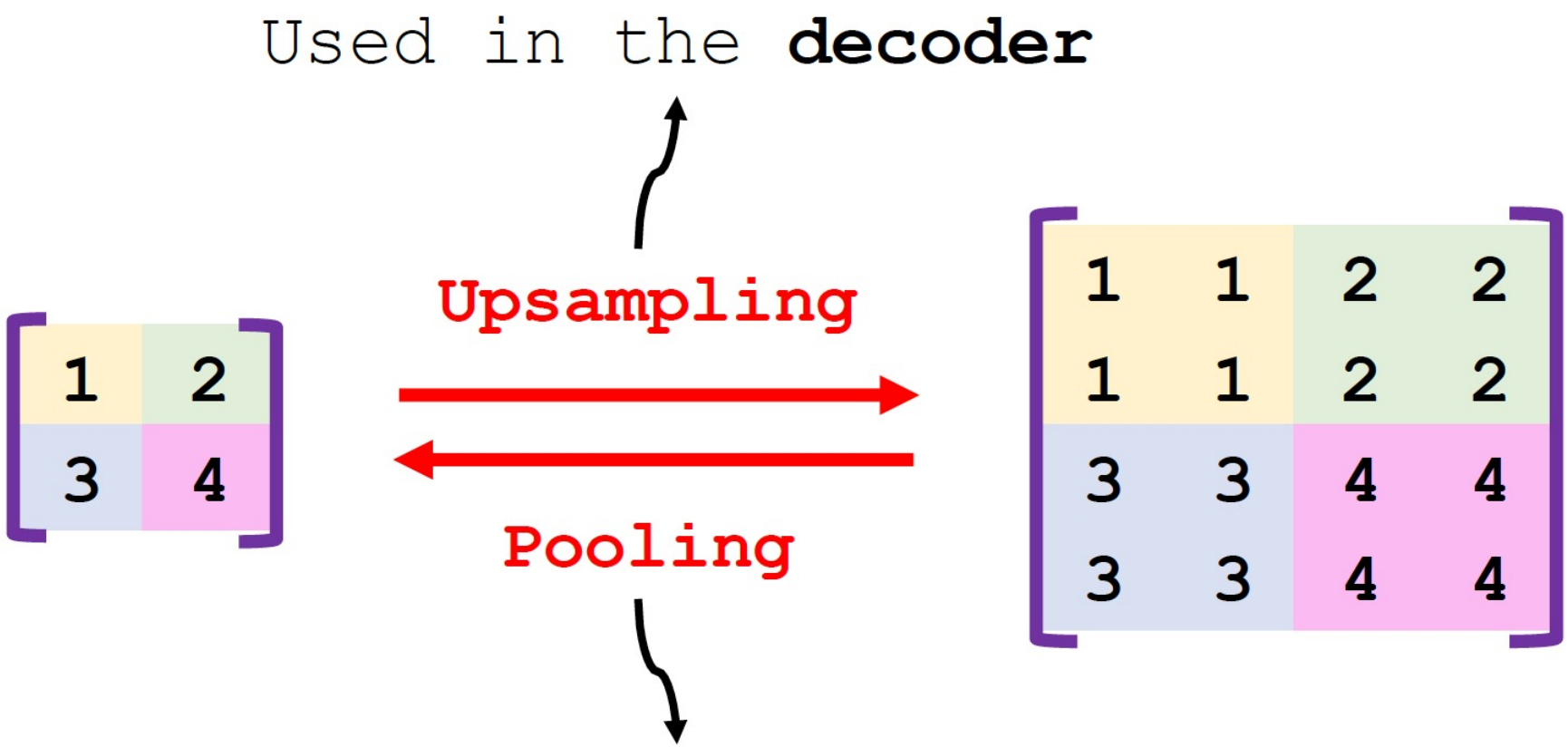
PCA projects data onto a subspace.



Convolutional Autoencoder

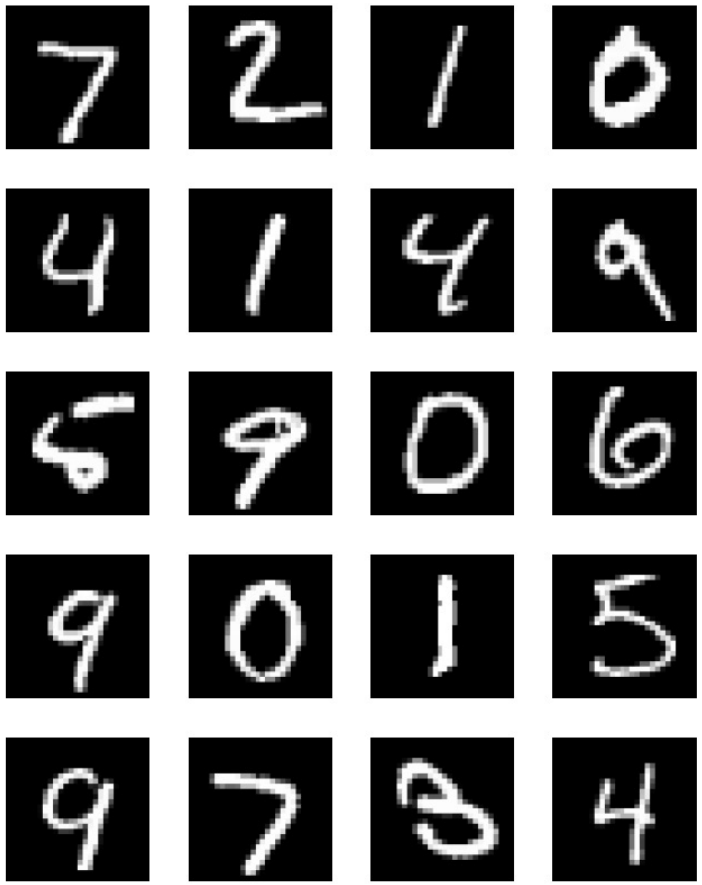


Pooling v.s. Upsampling

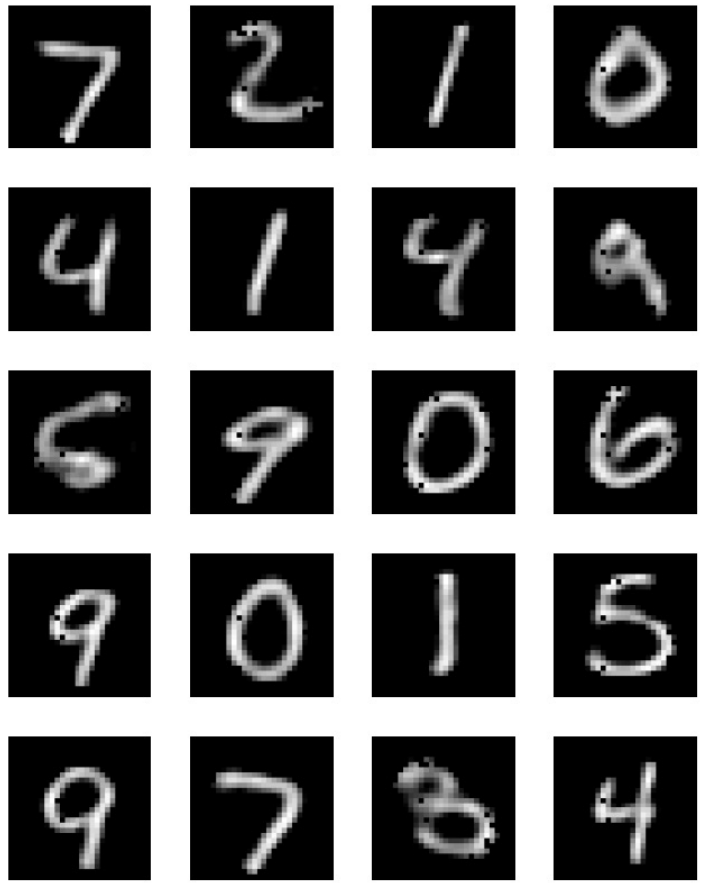


Reconstructions (on the Test Set)

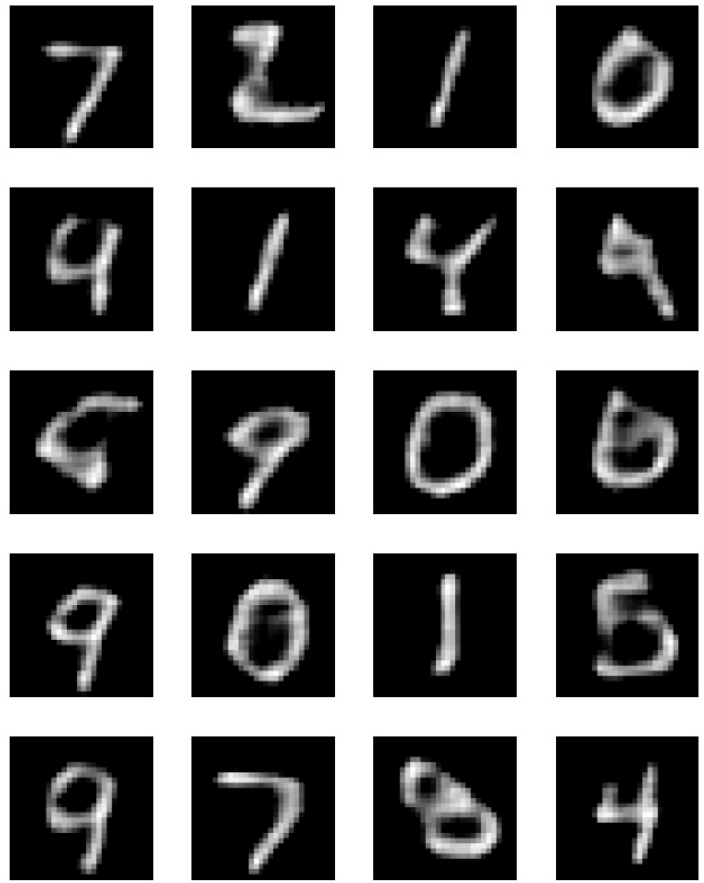
Original Input



Dense Autoencoder

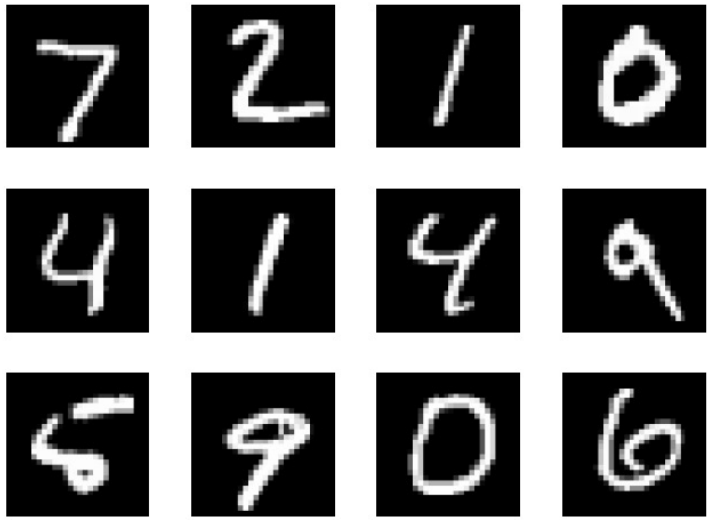


Conv Autoencoder



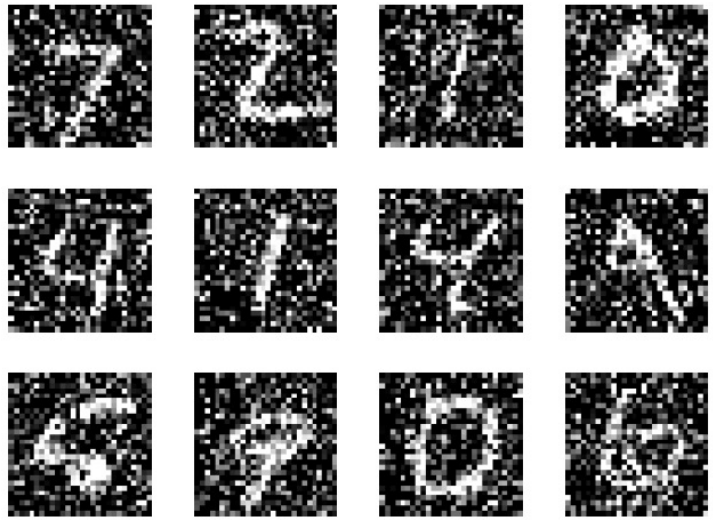
Denoising Autoencoder

Original



add noise

Noisy

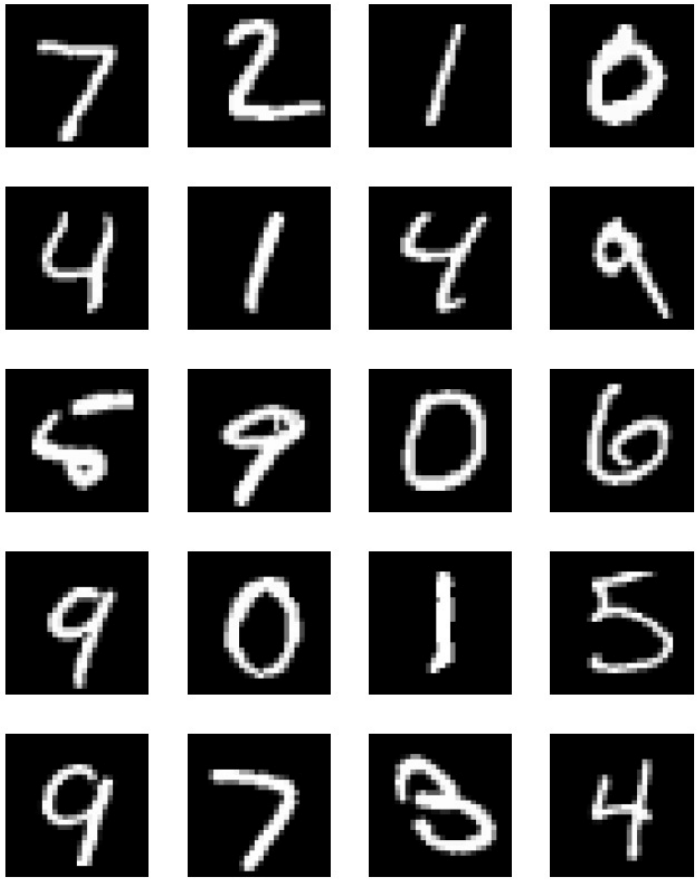


Used as targets

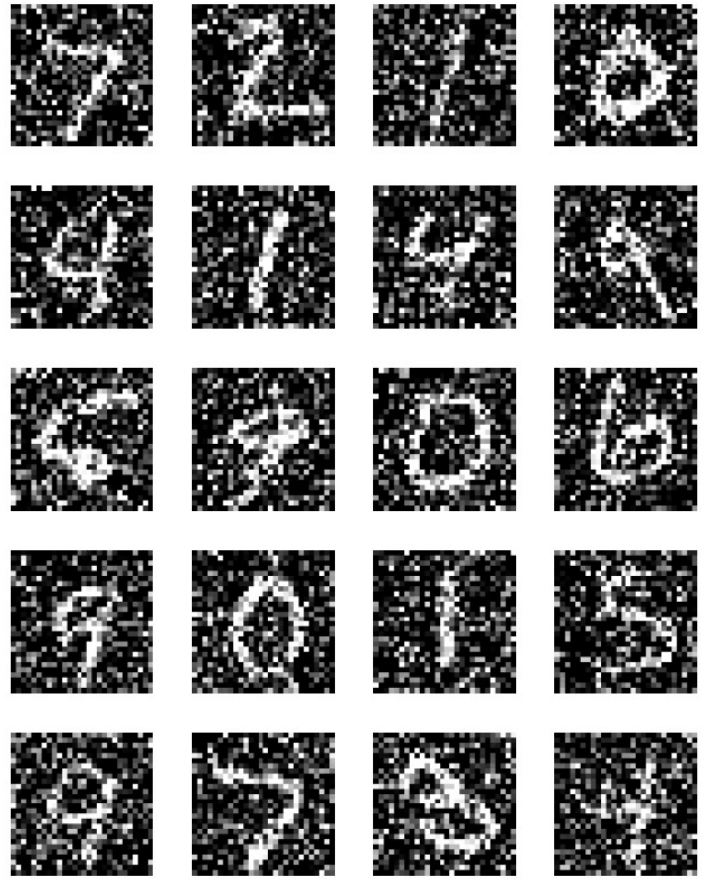
Used as inputs

Results on the Test Set

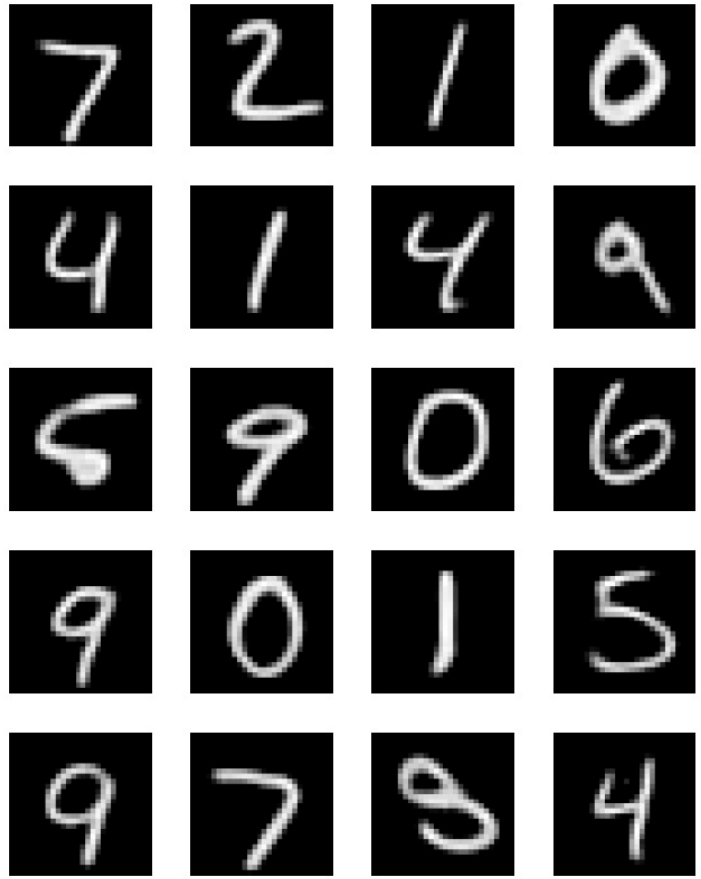
Original



Noisy



Reconstructed

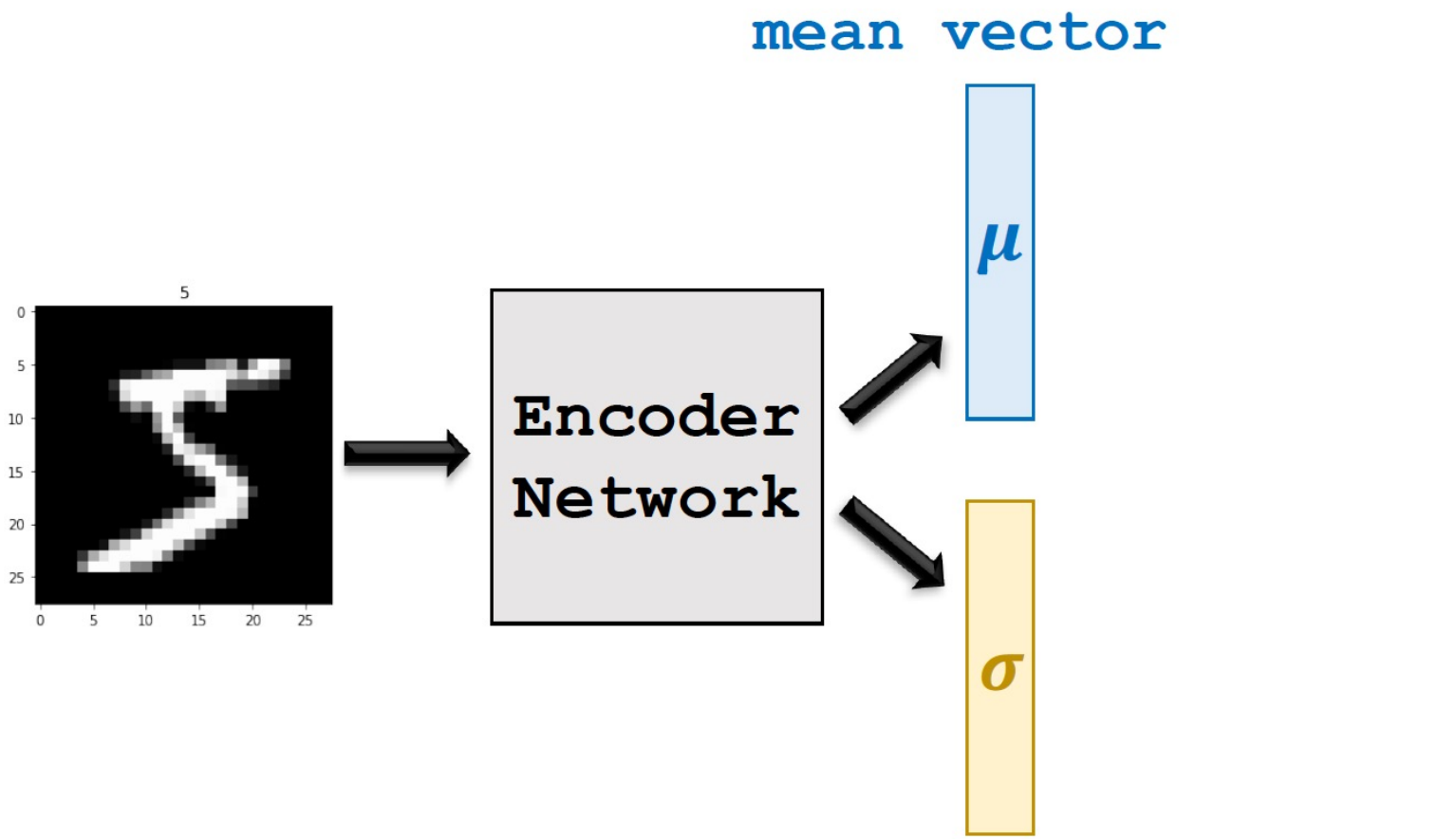


Summary

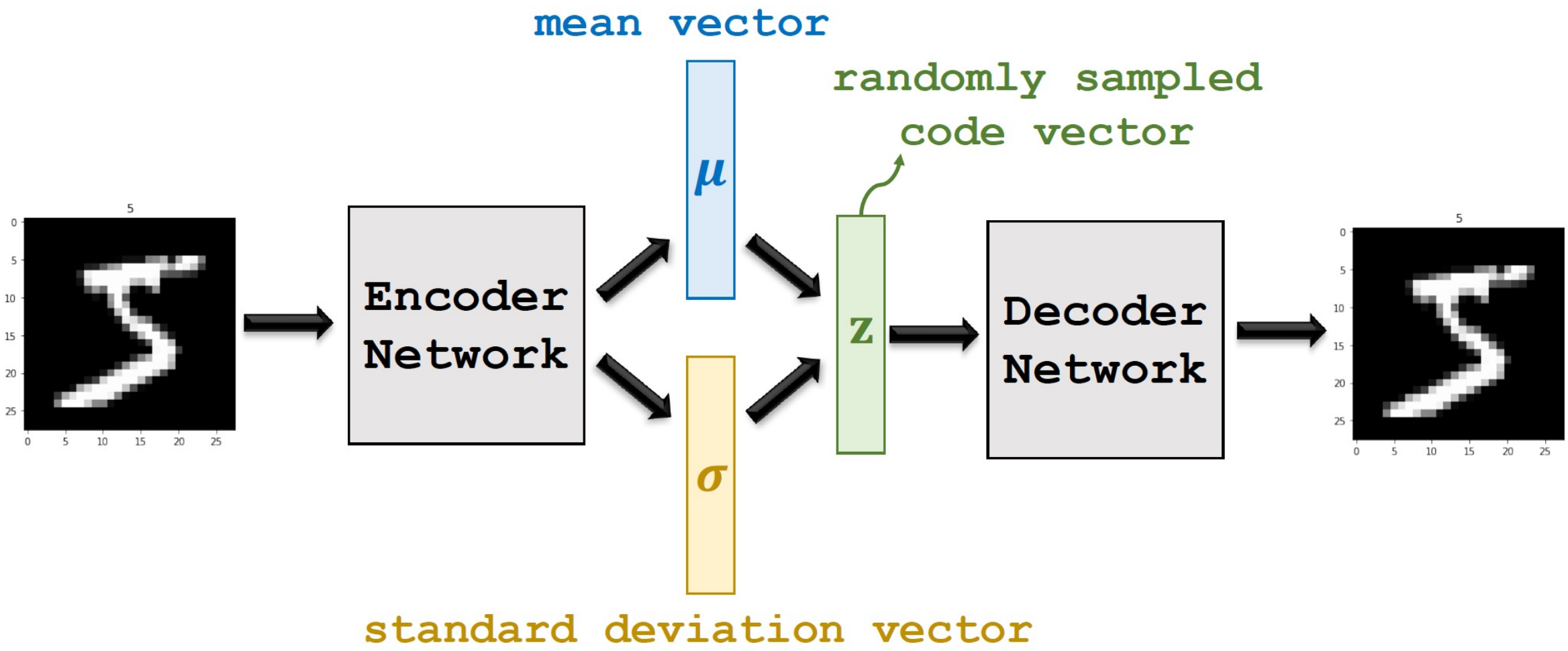
- Autoencoder: generalizations of (linear) PCA.
- Autoencoder = Encoder + Decoder.
- Training:
  - Inputs: original or noisy images.
  - targets: original images.
- Application 1: Dimensionality reduction (using the code vector).
- Application 2: Denoising.

Variational Autoencoders

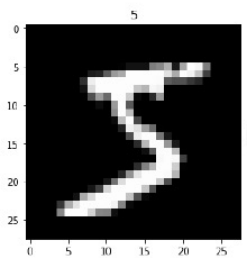
Variational Autoencoder (VAE)



Variational Autoencoder (VAE)



VAE Sampling

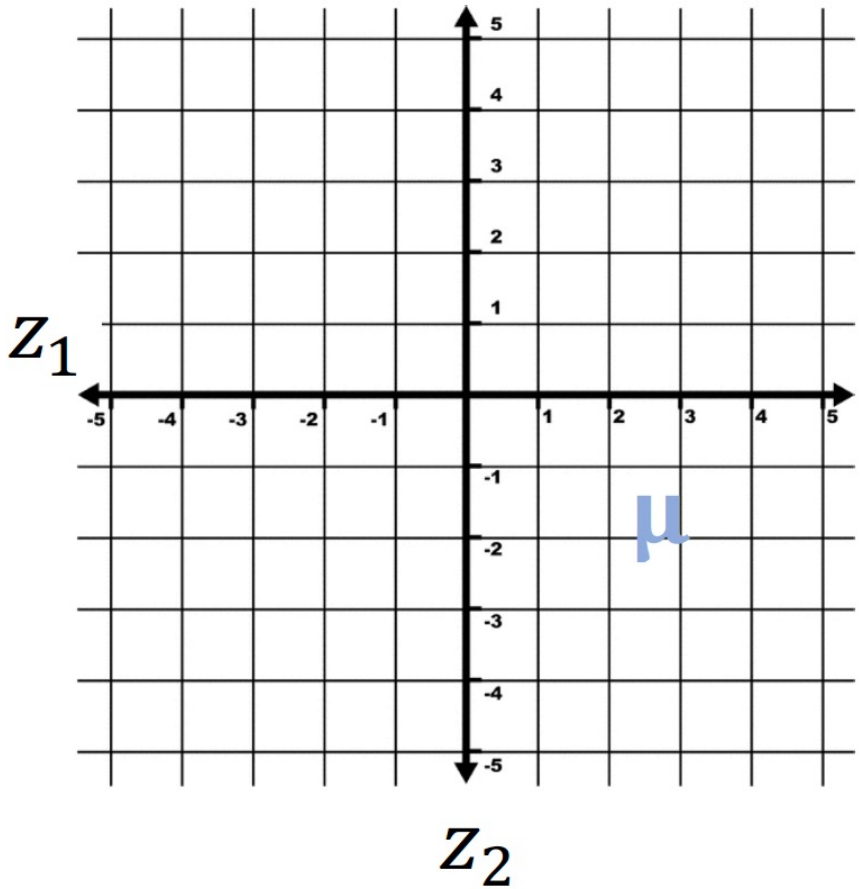


Encoder Network

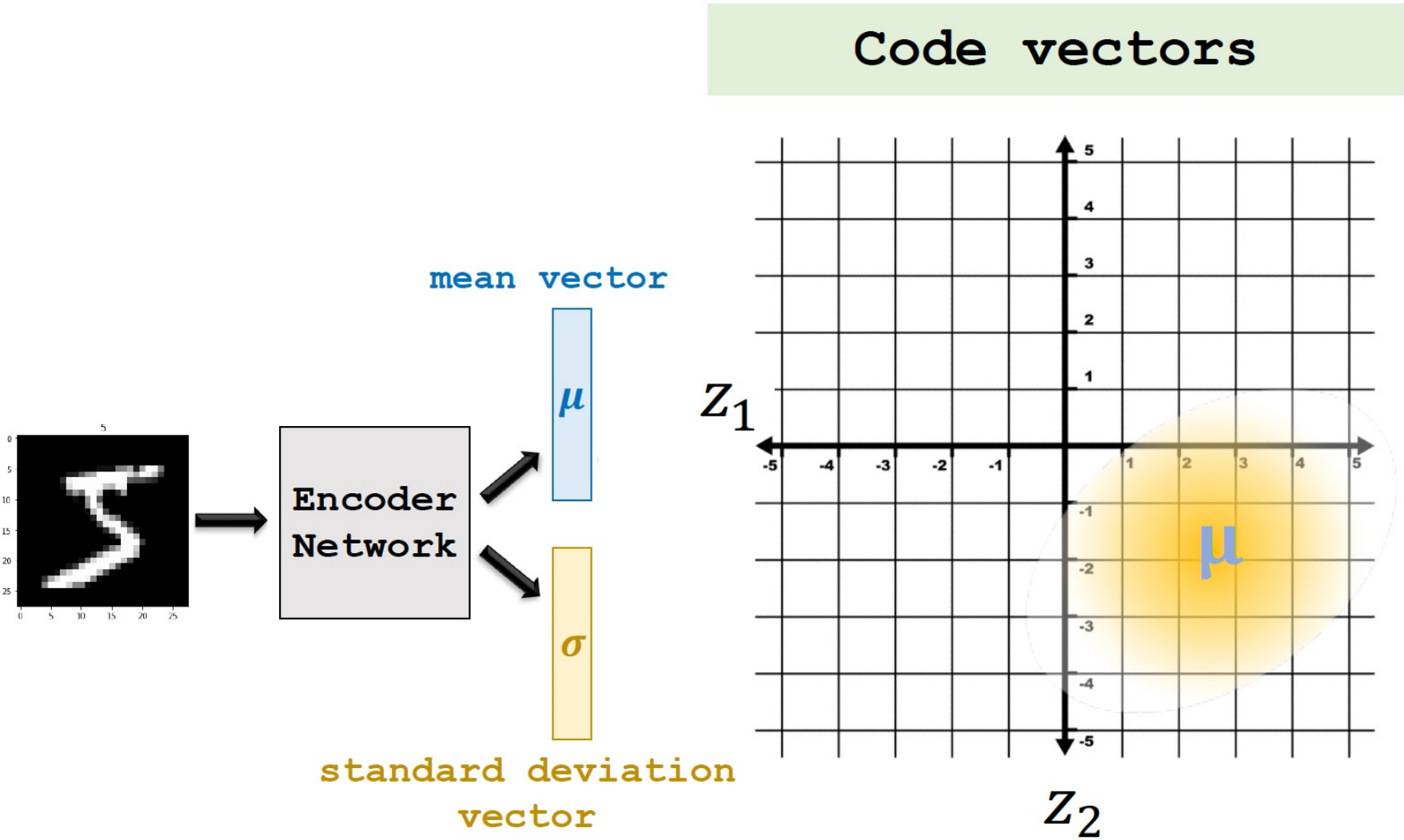
mean vector

μ

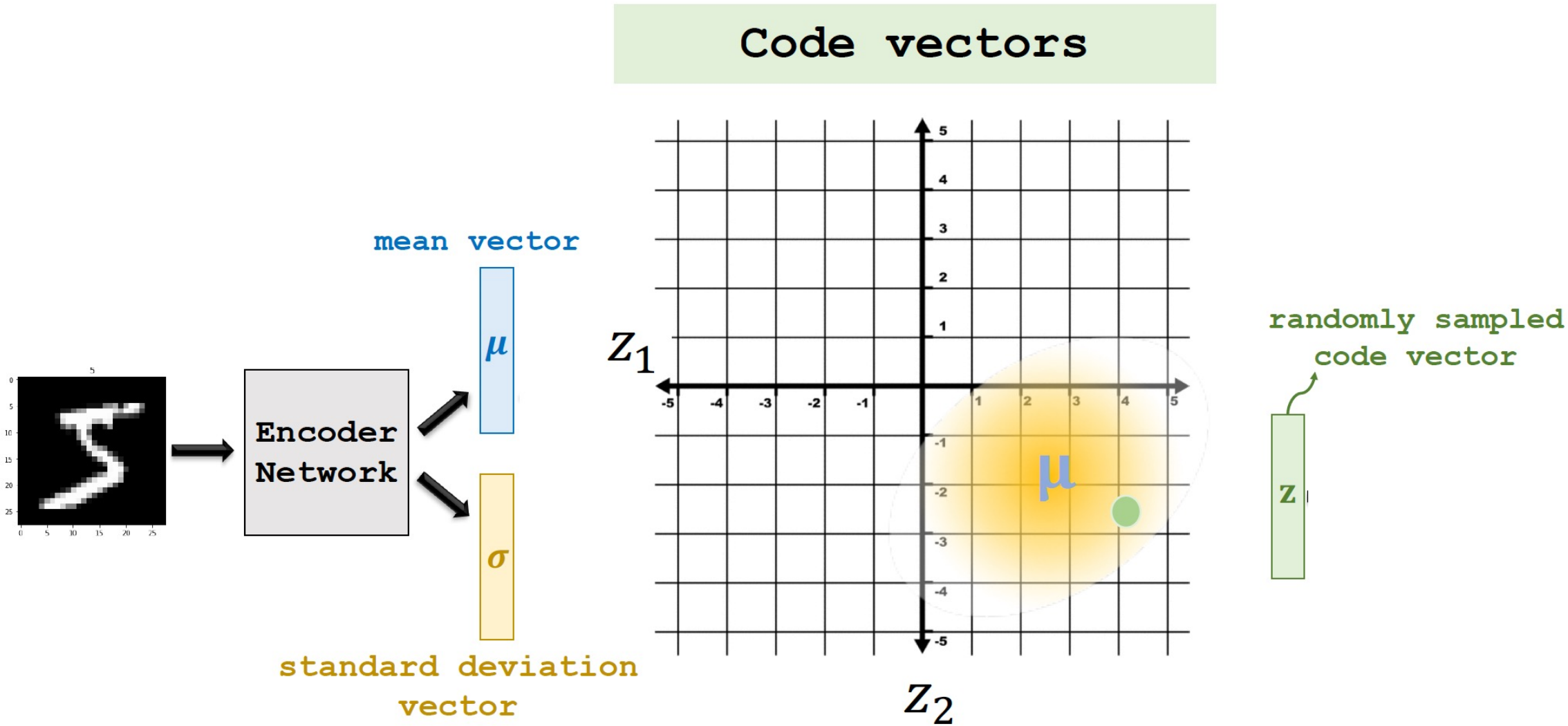
Code vectors



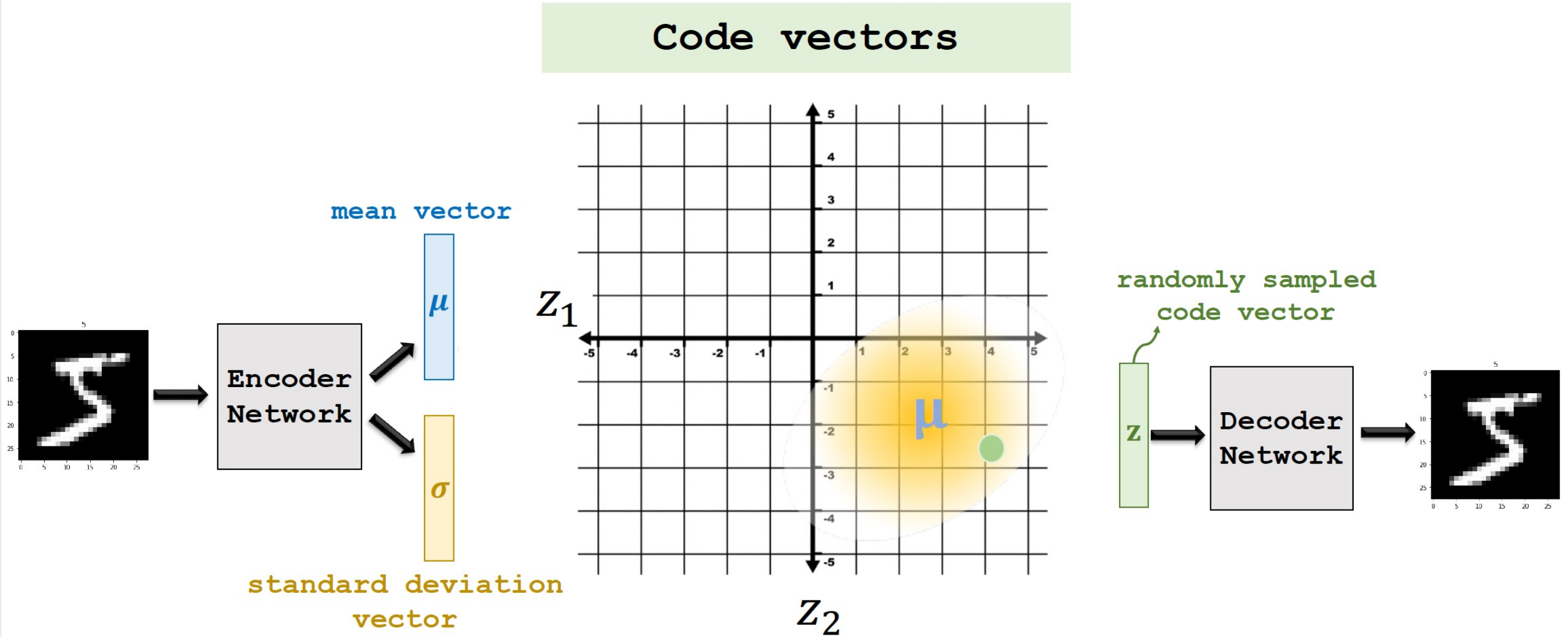
VAE Sampling



VAE Sampling

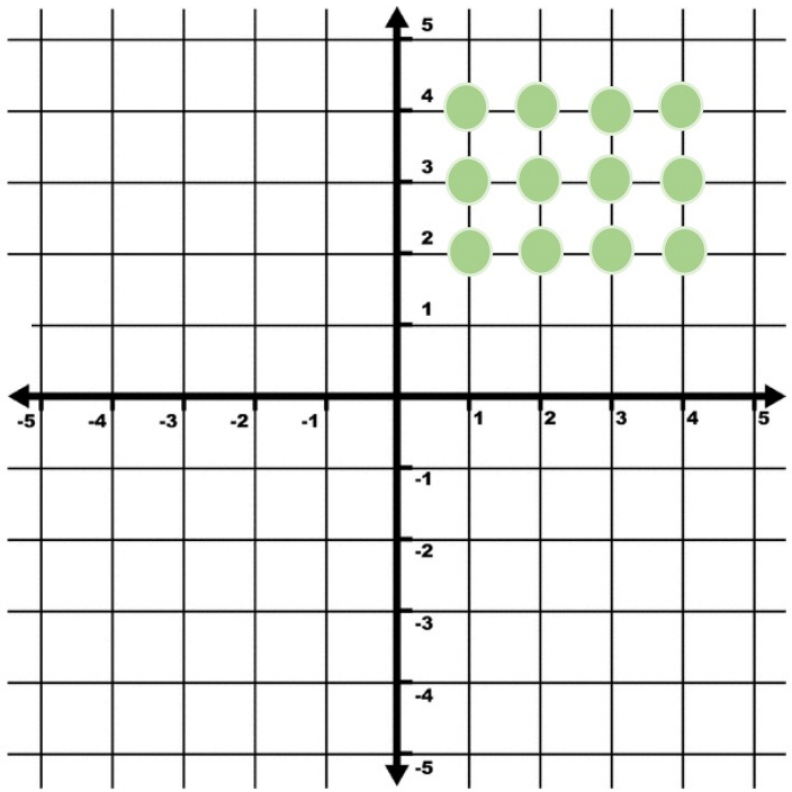


VAE Sampling



Visualize code vectors.

Code vectors

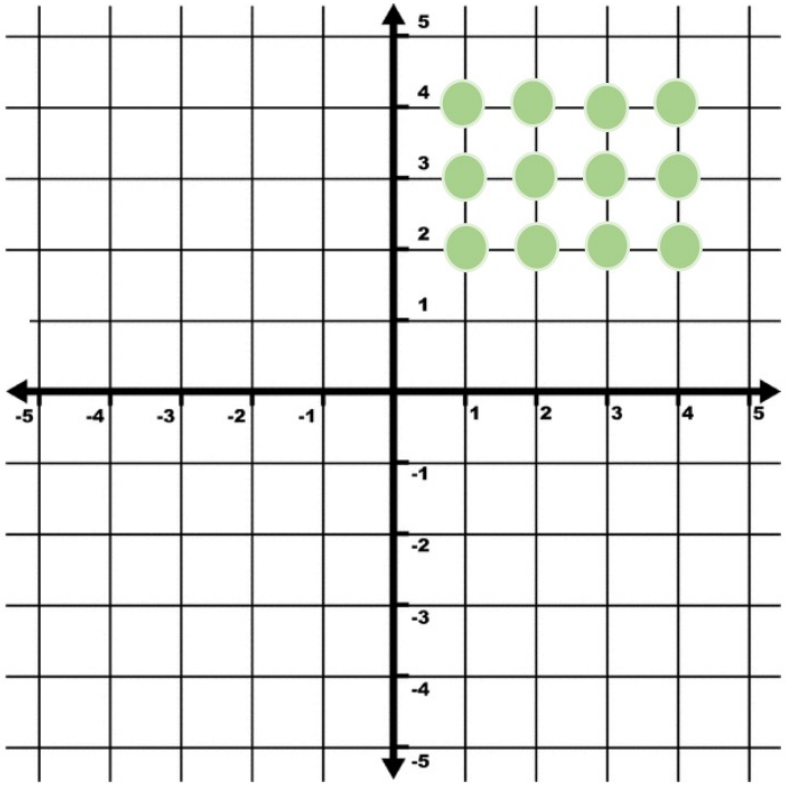


- Get a set of code vectors from the grid:

$$\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3, \mathbf{z}_4, \mathbf{z}_5, \cdots$$

Visualize code vectors.

Code vectors



- Get a set of code vectors from the grid:

$$\mathbf{z}_1, \mathbf{z}_2, \mathbf{z}_3, \mathbf{z}_4, \mathbf{z}_5, \cdots$$

- For every code vector $\mathbf{z}_i$, map it to an image using the decoder:

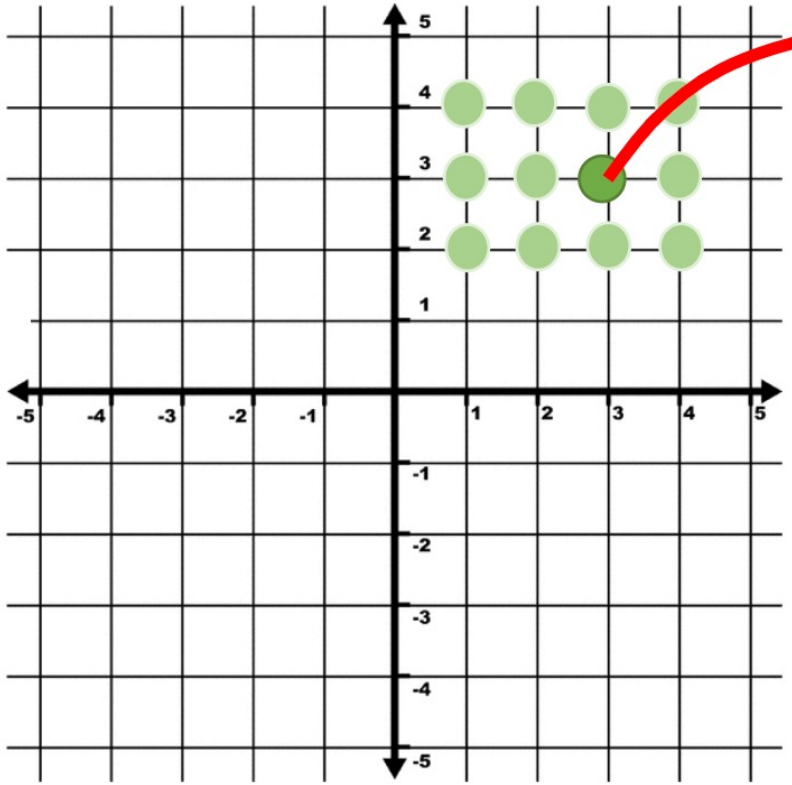
$$\text{image}_i = \text{Decoder}(\mathbf{z}_i).$$

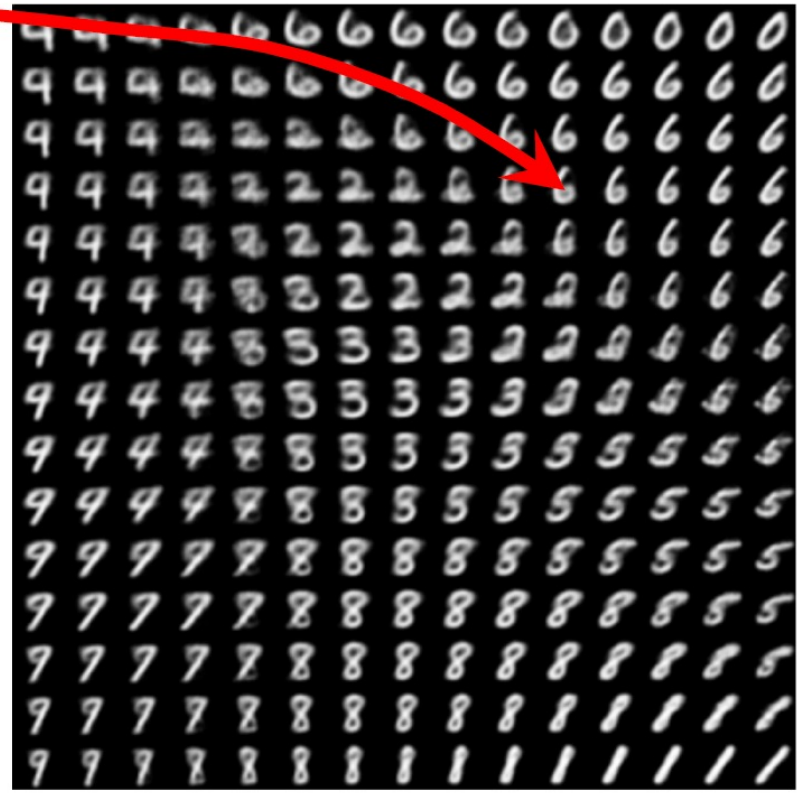
Visualize code vectors.

Code vectors

Images

Decoder





The decoder behaves like a continuous function.

- Function f is continuous if a small change in z (input) results in a small change in $f(z)$ (function value).

- The decoder network is trained to be (almost) continuous.
- If the code vectors $\mathbf{z}$ and $\mathbf{z}'$ are similar, then the images

$$\text{Decoder}(\mathbf{z}) \quad \text{and} \quad \text{Decoder}(\mathbf{z}')$$

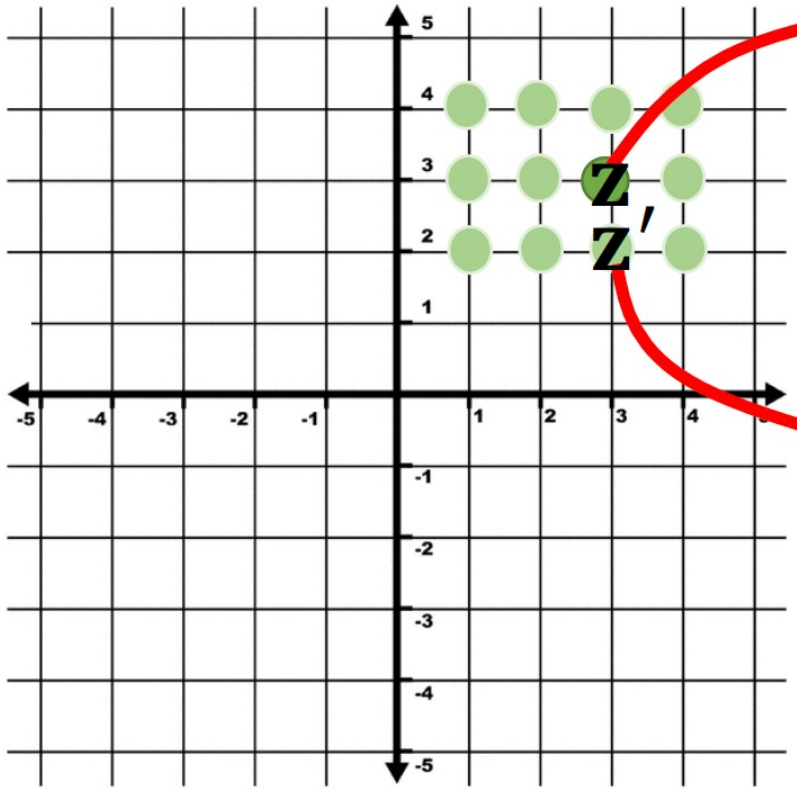
are similar as well.

The decoder behaves like a continuous function.

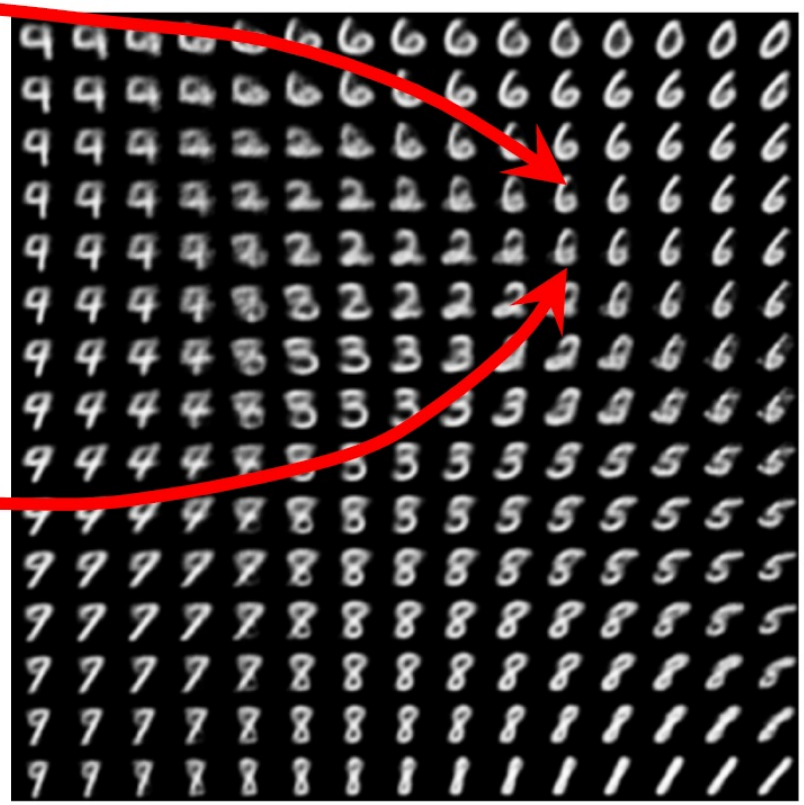
Code vectors

Images

Decoder



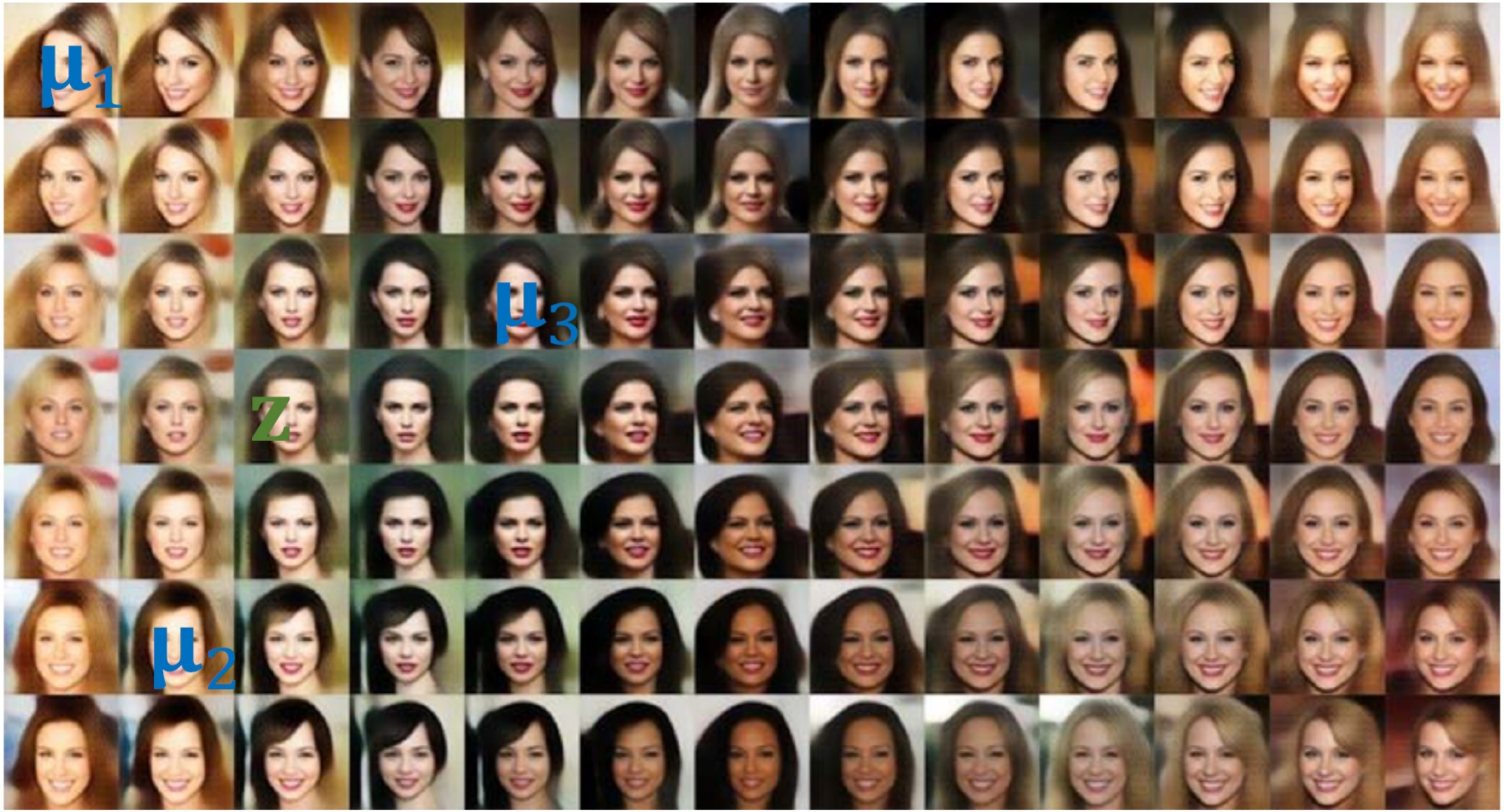
Decoder



The decoder behaves like a continuous function.

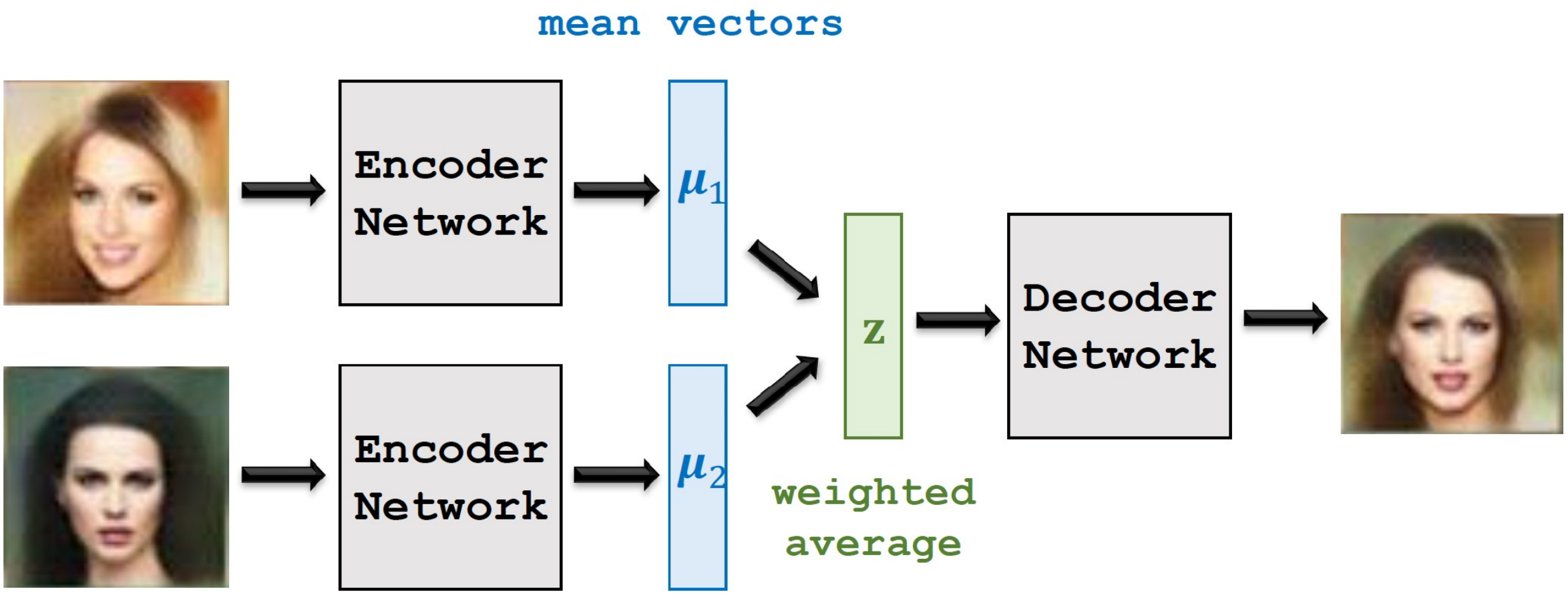
Visualization of the generated faces (by Tom White)

Averaging images via averaging code vectors

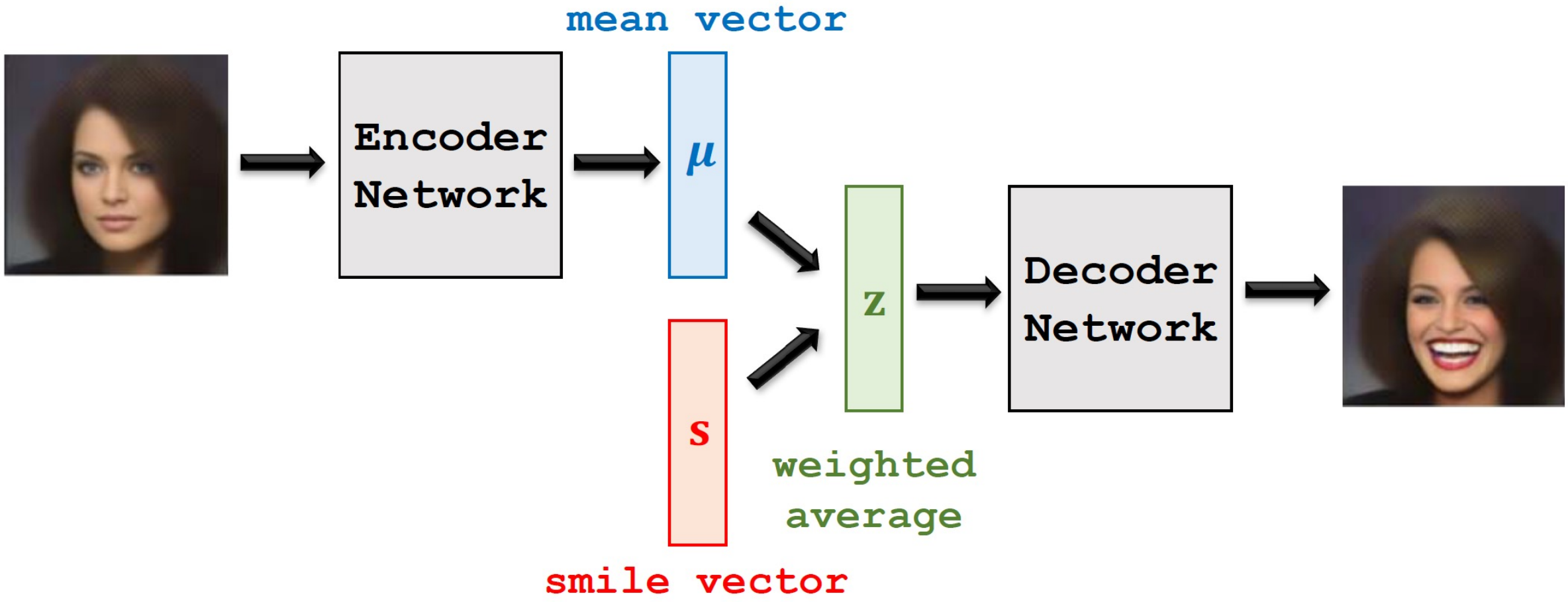


Visualization of the generated faces (by Tom White)

Averaging images via averaging code vectors



Editing images via editing code vectors



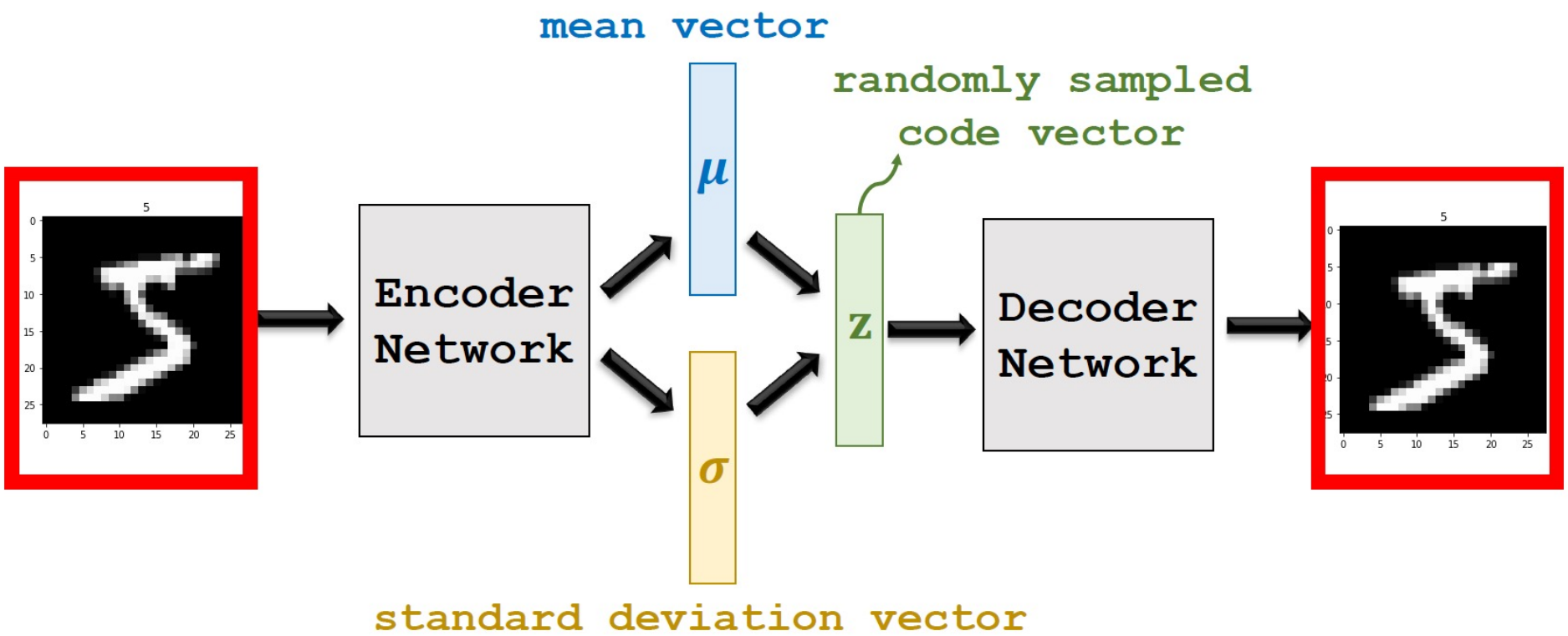
Editing images via editing code vectors

μ $\mu + s$ $\mu + 2s$ $\mu + 3s$ $\mu + 4s$

μ $\mu - s$ $\mu - 2s$ $\mu - 3s$ $\mu - 4s$

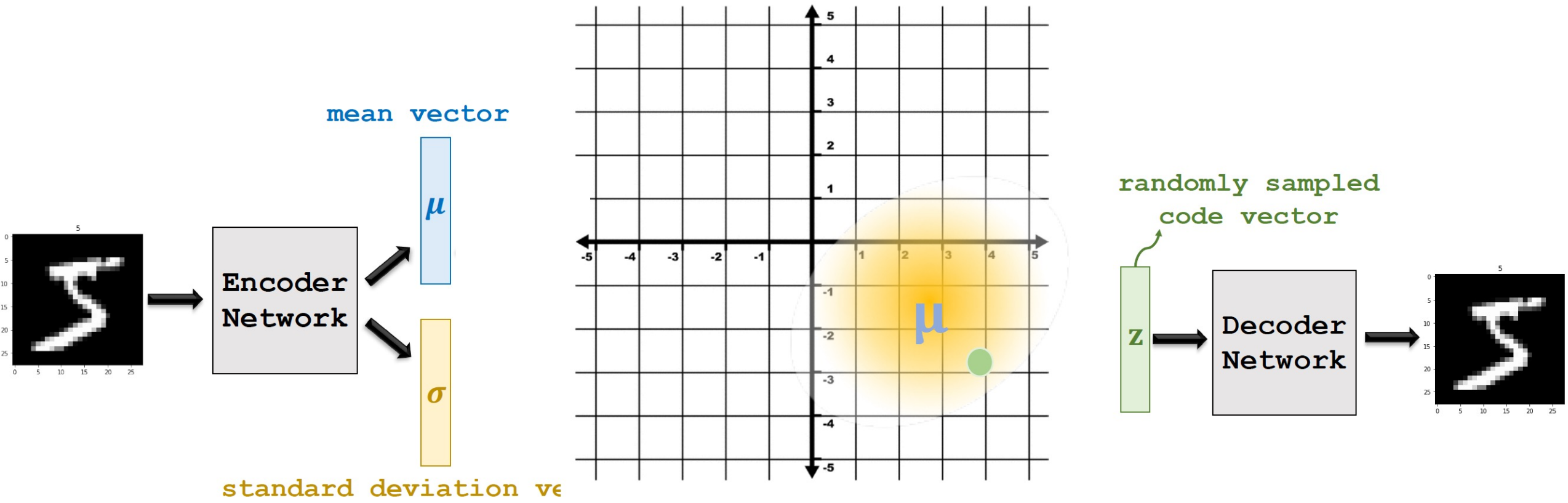
Generation Loss

- `GenLoss = dist(input_img, generated_img)`.
- E.g., the ℓ_2 distance, the cross-entropy, etc.



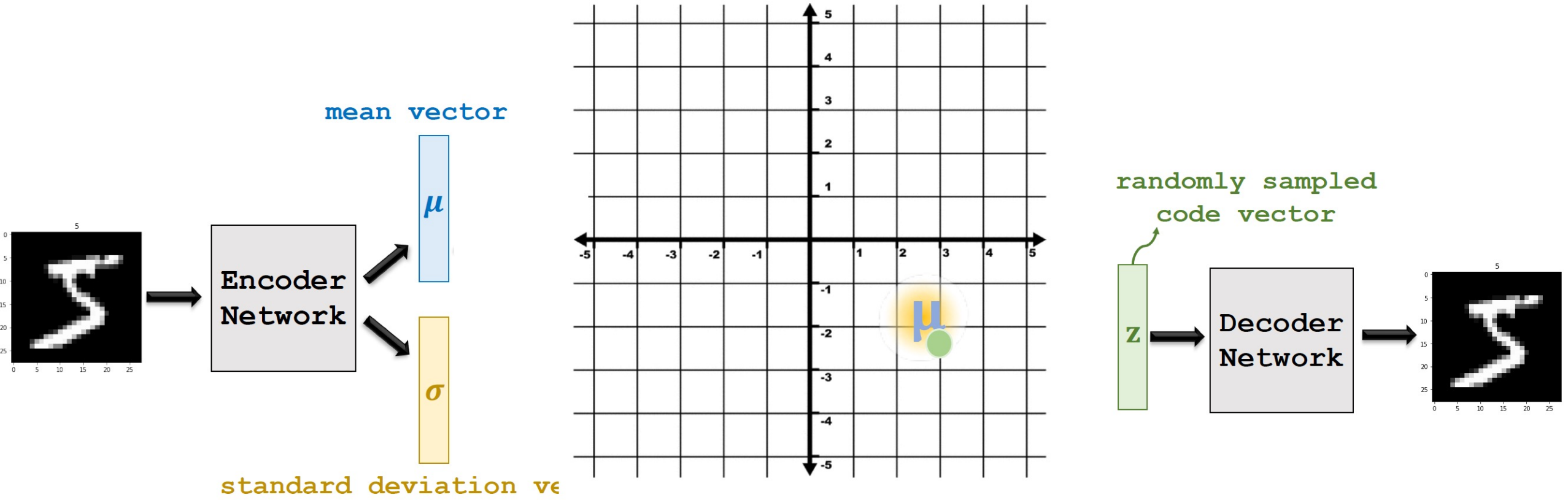
A problem with generation loss

Difficulty: The encoder network will learn an std vector $\sigma \to 0$.



A problem with generation loss

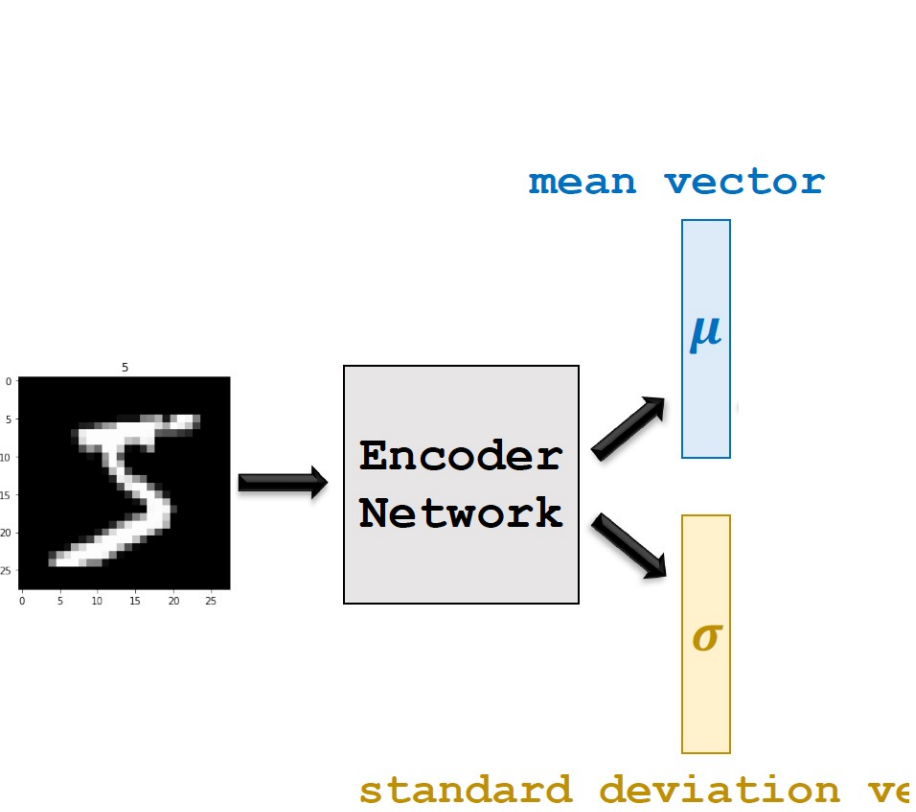
Difficulty: The encoder network will learn an std vector $\sigma \to 0$.

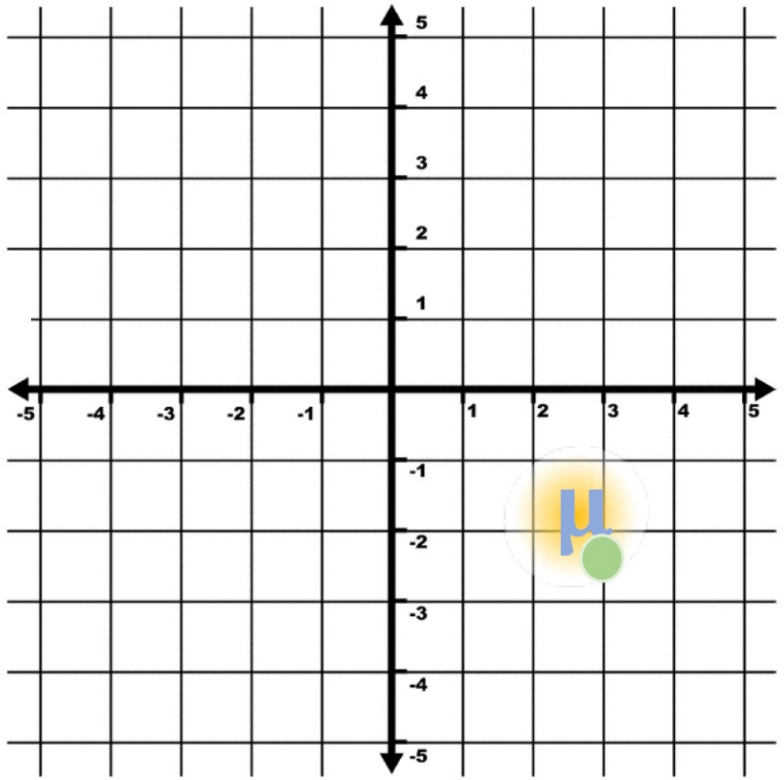


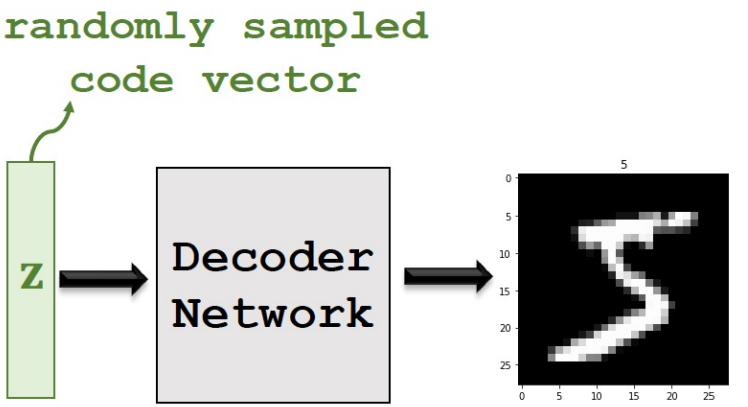
A problem with generation loss

Difficulty: The encoder network will learn an std vector $\sigma \to 0$. (Why?)

- Minimize `GenLoss` ➔ encourage z close to μ ➔ encourage small σ.
- VAE degrades to the standard AE (VAE with $\sigma = 0$ is exactly AE).



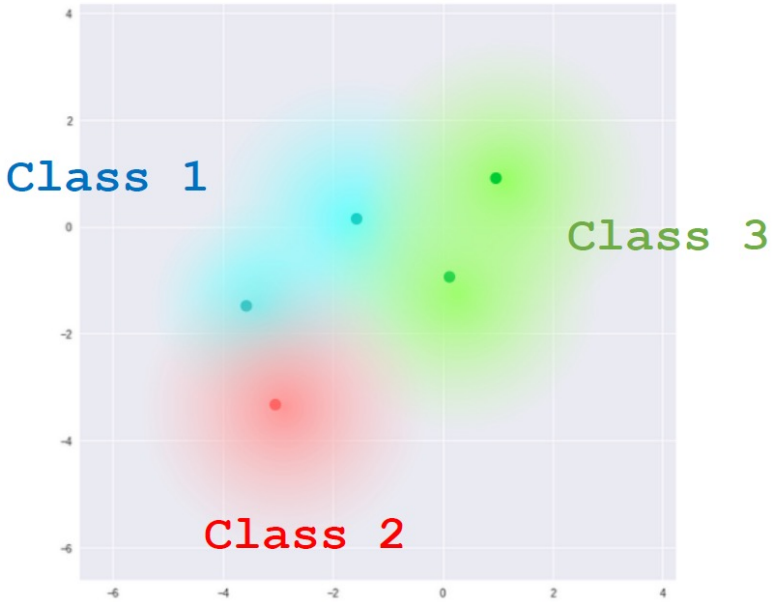




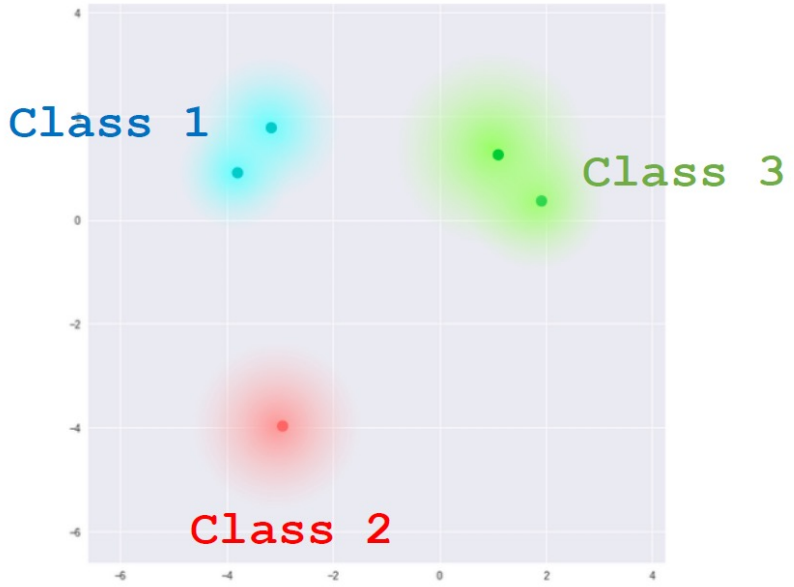
A problem with generation loss

Difficulty: By minimizing generation loss, VAE becomes standard AE.

Dots are code vectors ($\boldsymbol{\mu}$); shadows illustrate std ($\boldsymbol{\sigma}$).



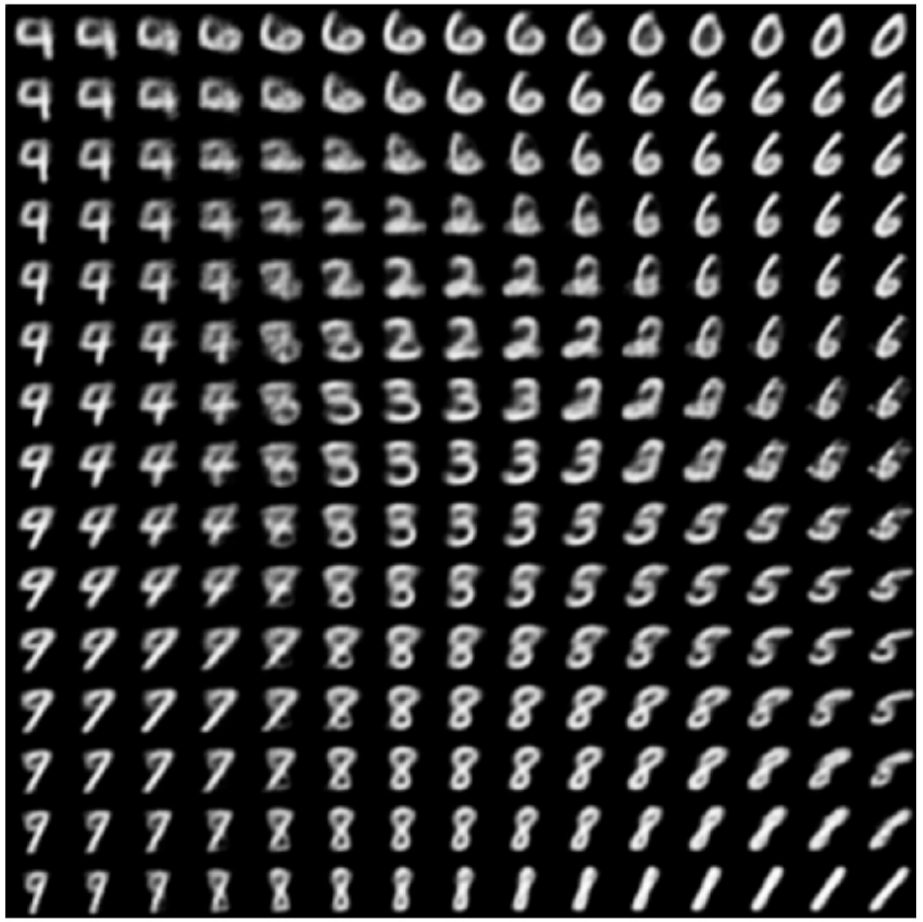
What we hope to learn.



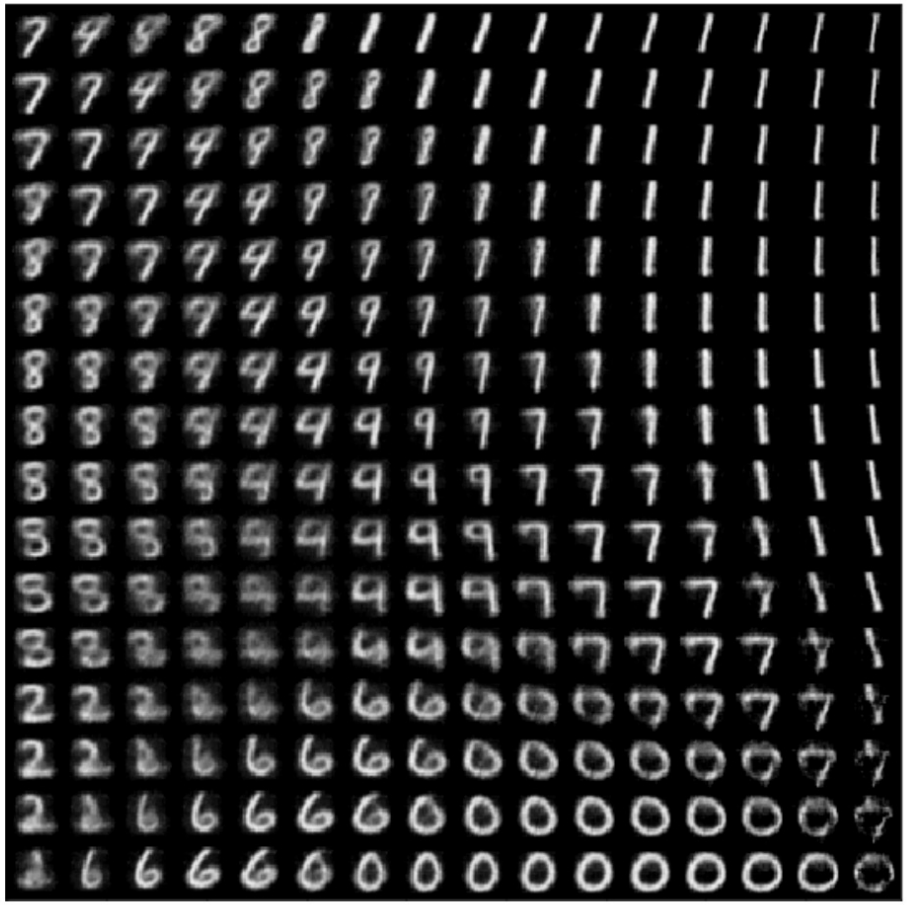
What we actually learn by minimizing `GenLoss`.

A problem with generation loss

Difficulty: By minimizing generation loss, VAE becomes standard AE.



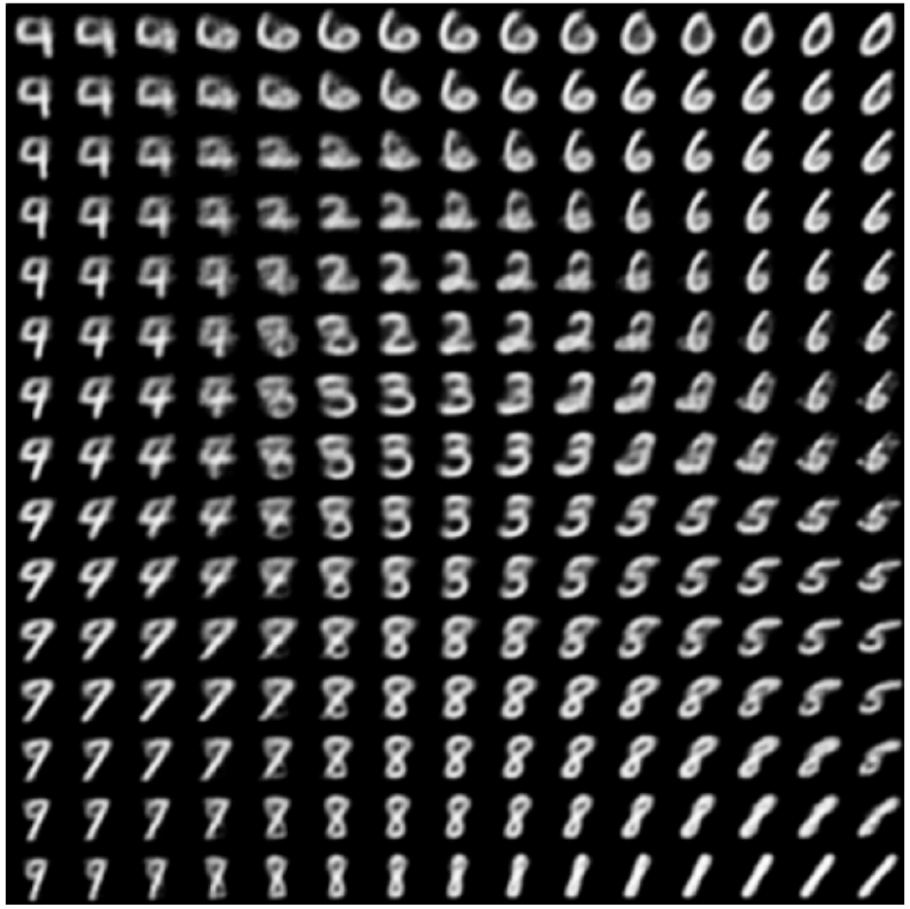
What we hope to learn.



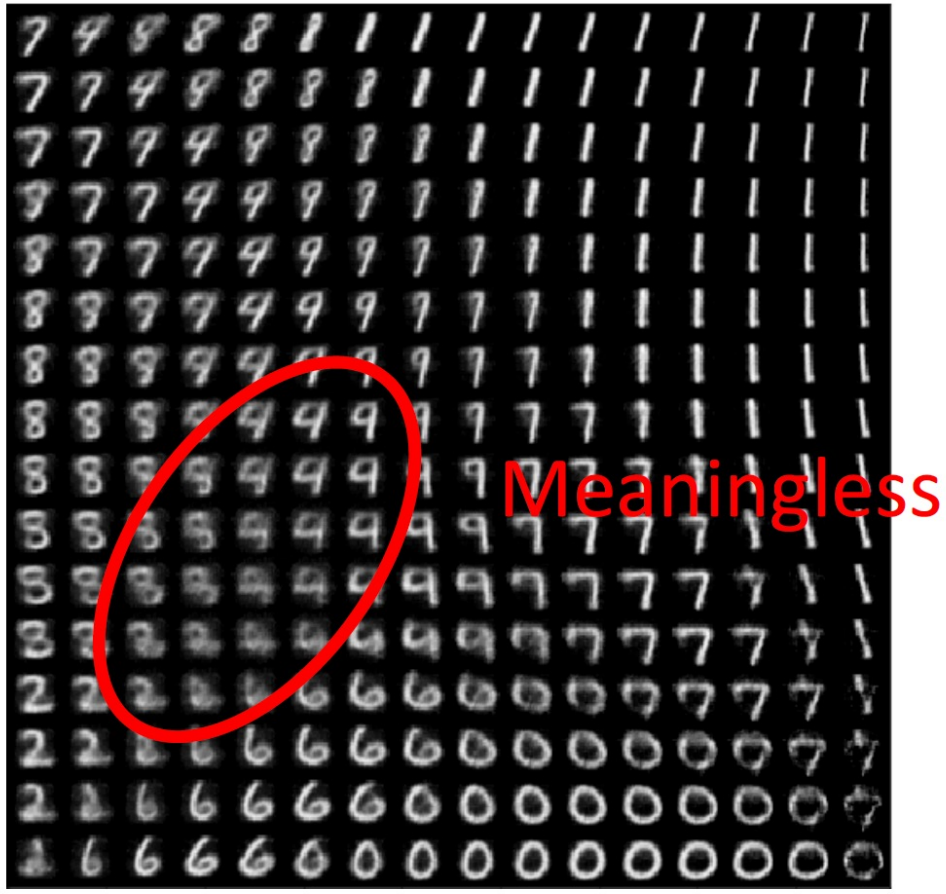
What we actually learn by minimizing `GenLoss`.

A problem with generation loss

Difficulty: By minimizing generation loss, VAE becomes standard AE.



What we hope to learn.

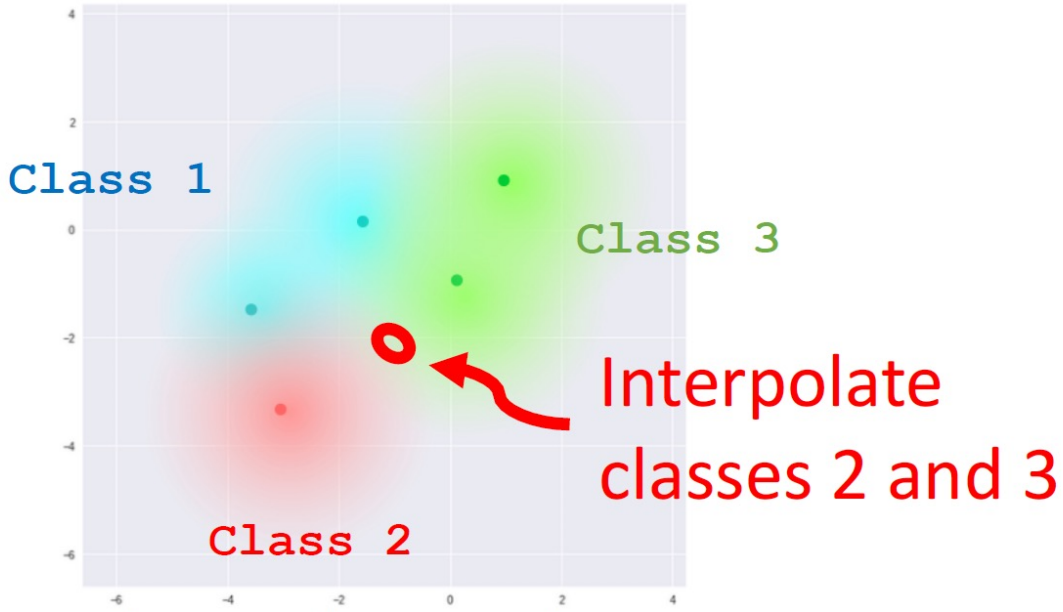


What we actually learn by minimizing `GenLoss`.

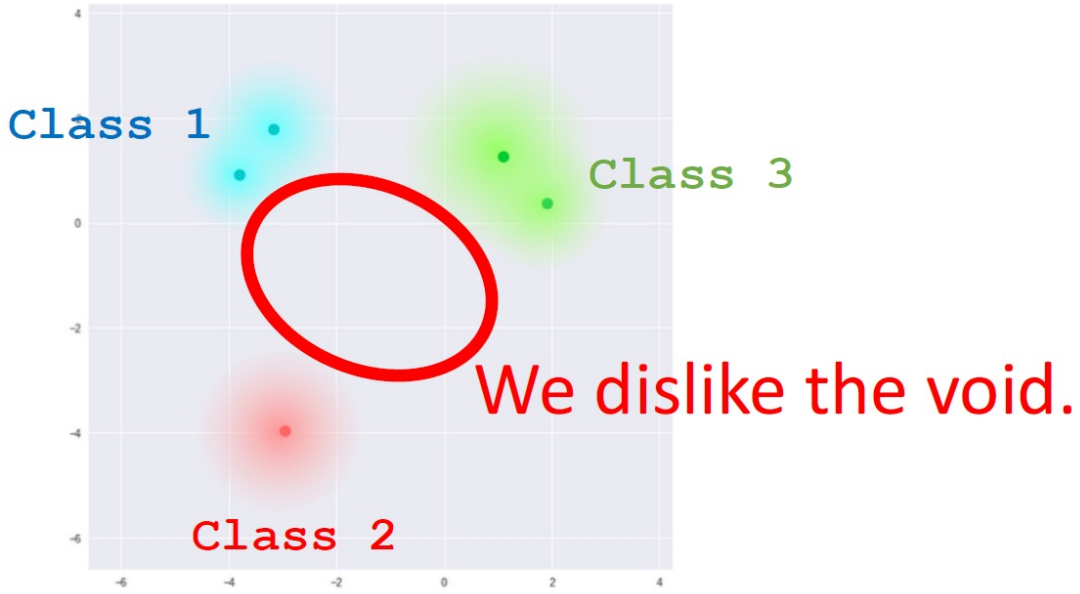
KL Loss

Idea: Encourage the distribution of **z** to be unit Gaussian.

- There will be a probability density everywhere around the origin.
- ➔ No more void.



What we hope to learn.

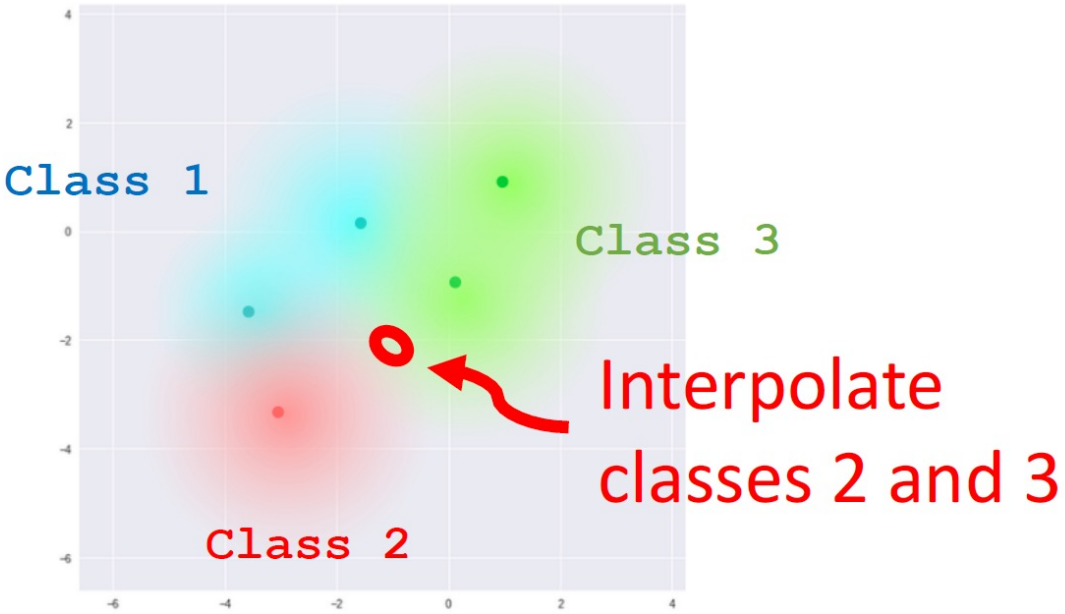


What we actually learn by minimizing `GenLoss`.

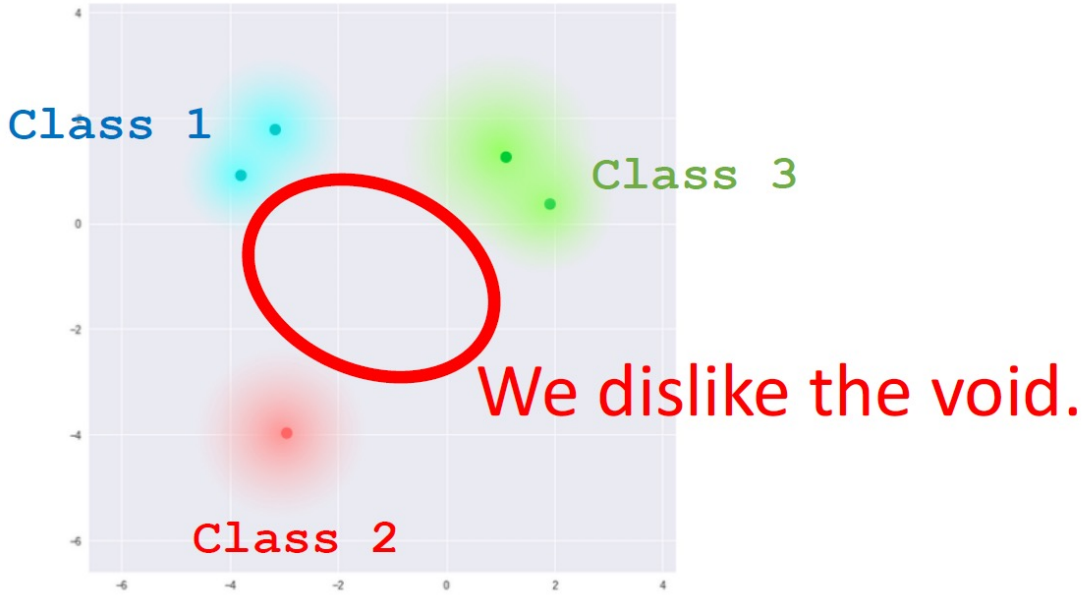
KL Loss

Idea: Encourage the distribution of $\mathbf{z}$ to be unit Gaussian.

- $\text{KLLoss} = D_{\text{KL}}\left(p(\mathbf{z}) \;||\; \mathcal{N}(0, \mathbf{I})\right)$



What we hope to learn.



What we actually learn by minimizing `GenLoss`.

KL Loss

Idea: Encourage the distribution of $\mathbf{z}$ to be unit Gaussian.

- $\text{KLLoss} = D_{\text{KL}}\left(p(\mathbf{z}) \;||\; \mathcal{N}(0, \mathbf{I})\right)$

- $\text{Loss} = \text{GenLoss} + \lambda \cdot \text{KLLoss}$

Variational Autoencoder (VAE)

- VAE = AE + probability tricks.
- The decoder network behaves like a continuous function.
- Loss = Generation Loss + $\lambda \cdot$ KL Loss.
- Application: edit images via editing code vectors.
  - Average faces.
  - Add smile.

Any Question ?