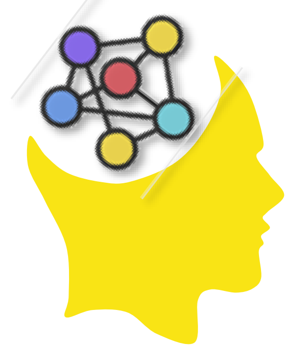


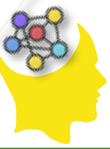
Adv ML for Gen-AI

(Variational) AutoEncoder

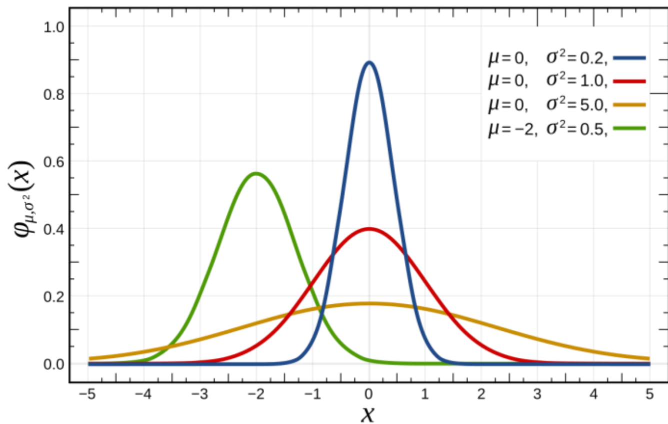
https://ml-graph.github.io/spring-2026/

Yu Wang, Ph.D.
Assistant Professor
Department of Computer Science
University of Oregon
Personal: https://yuwang0103.github.io/
Lab: https://kindlab-fly.github.io/

Summary

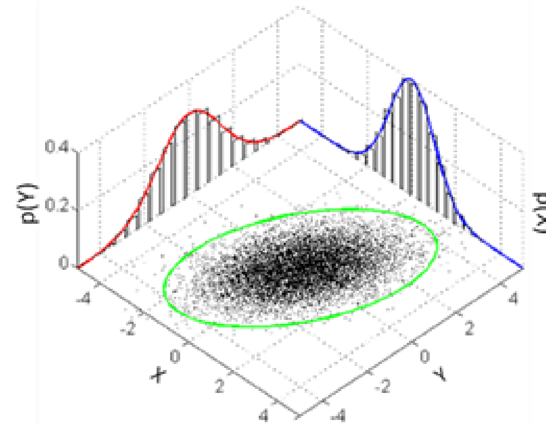


1D Gaussian Distribution

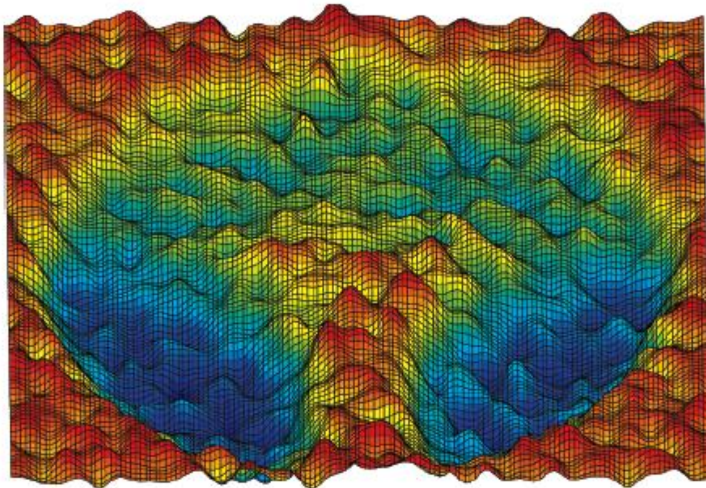


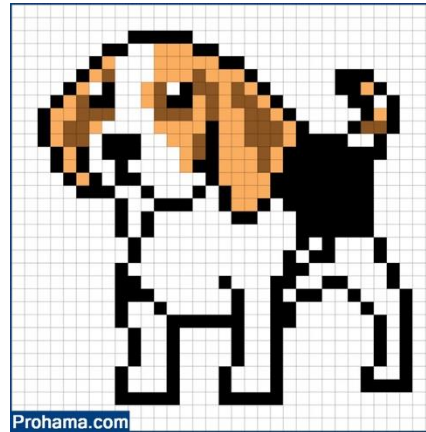
$\mathbb{R}$

2D Gaussian Distribution



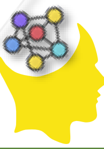
$\mathbb{R}^2$



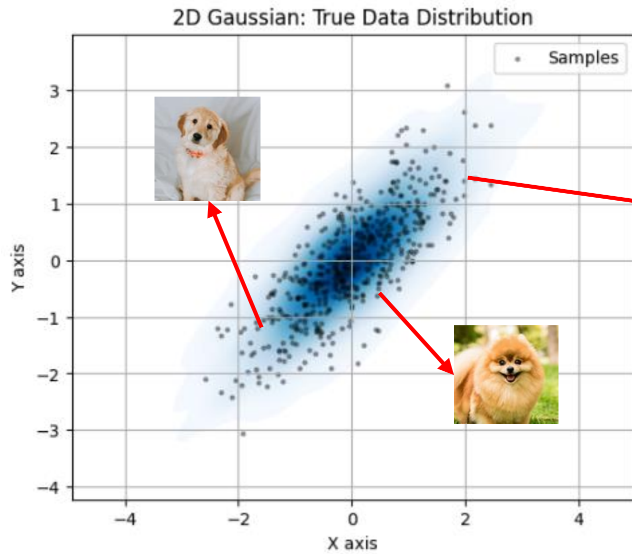


$\mathbb{R}^{256\times256}$

Summary



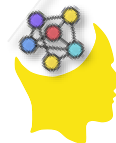
Modeling Data Distribution $\{x_i\}_{i=1}^{N} \rightarrow X$



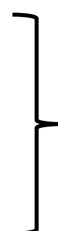
Sampling from it

$x \sim X$

Summary

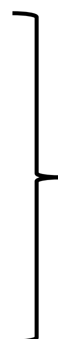


- **Statistical Methods**
  - Gaussian Estimation (1D or 2D or higher)
  - Gaussian Kernel Density Estimation
  - Gaussian Mixture Models



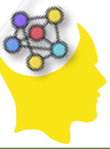
We either assume the shape
Or we use assumed shape to approximate

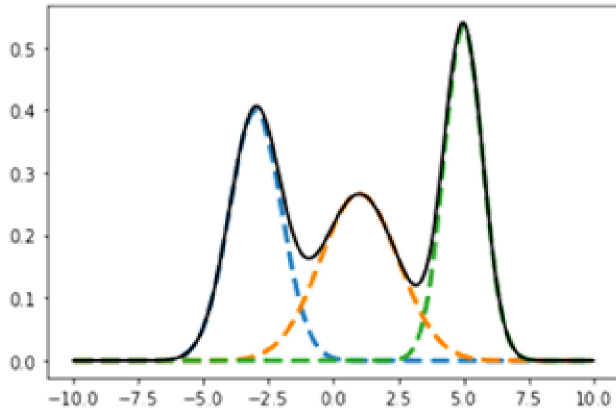
- **Machine/Deep Learning Methods**
  - Auto-Encoder (AE)
  - Variational AE
  - Generative Adversarial Network

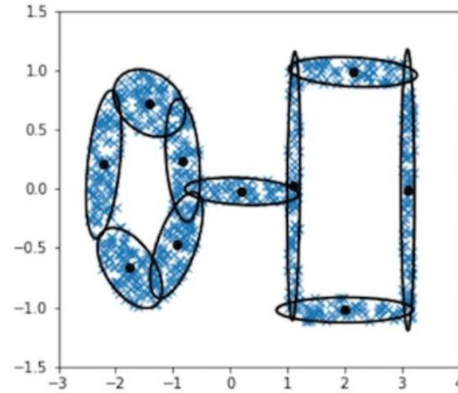


We do not assume any prior shape

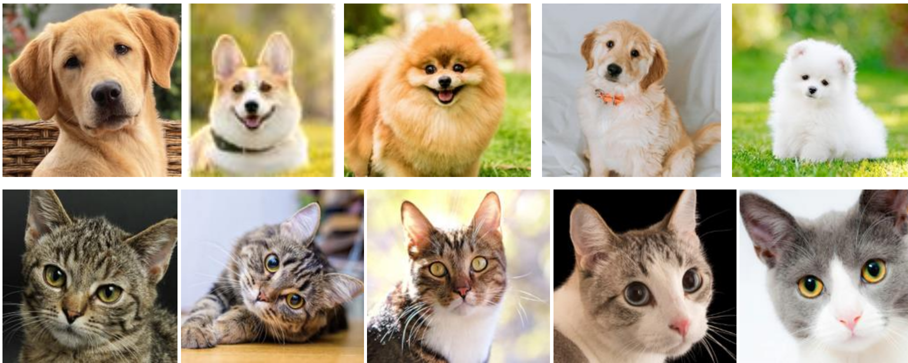
Problem so far ……







$\mathbb{R}^1, \mathbb{R}^2$

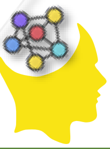


Input

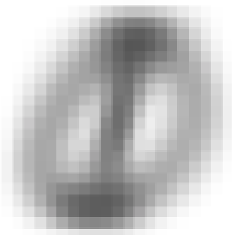
$\mathbb{R}^{256\times256}$

High-Dimensional Data

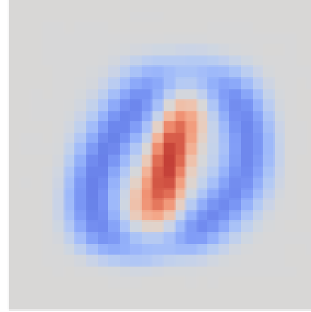
High-Dimension Data



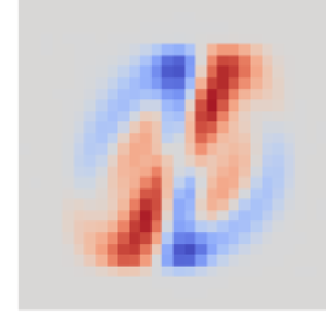
Dataset mean image

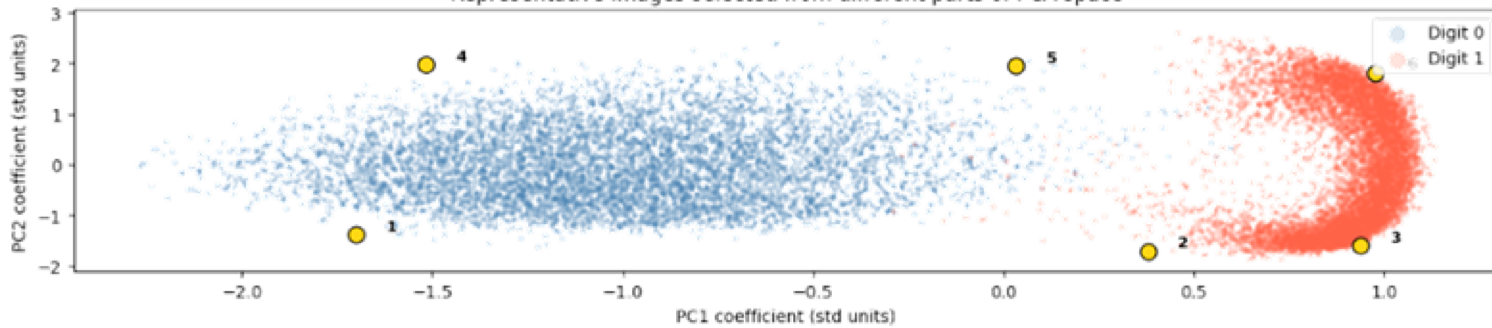


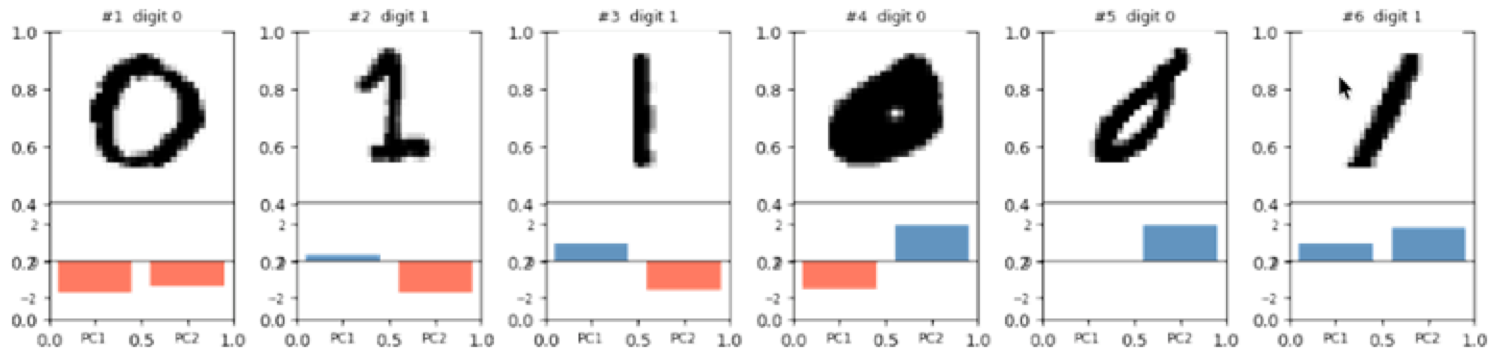
PC1 direction in pixel space



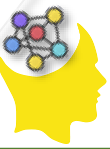
PC2 direction in pixel space







Observation



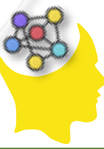
A key assumption: high-dimensional data lies on the low-dimensional manifold space

PC 2

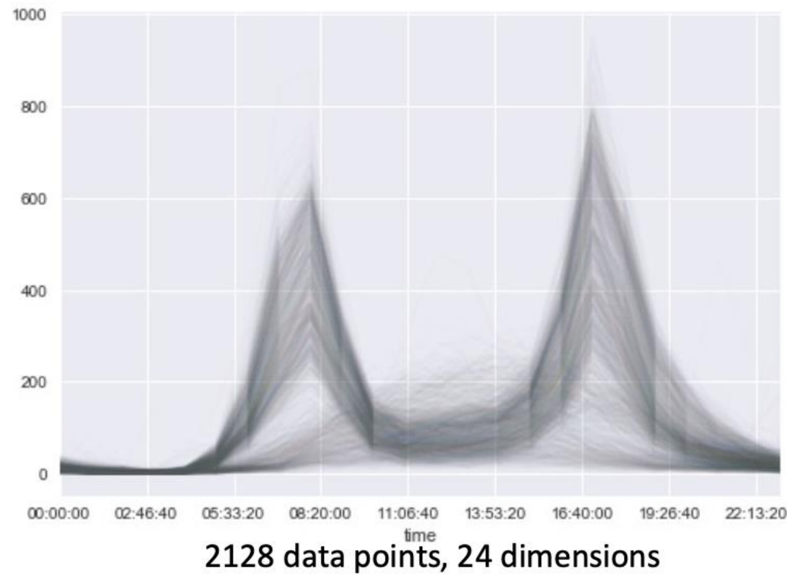
PC 1

Comp 0 (π=0.21)
Comp 1 (π=0.26)
Comp 2 (π=0.25)
Comp 3 (π=0.29)

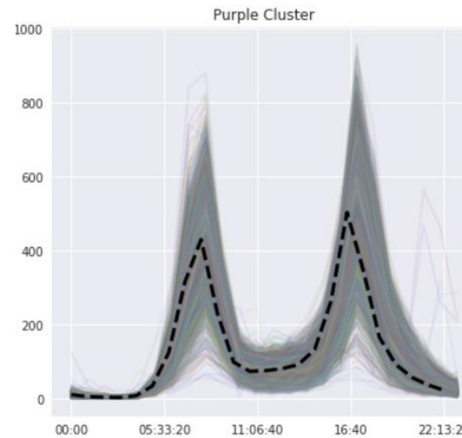
Observation

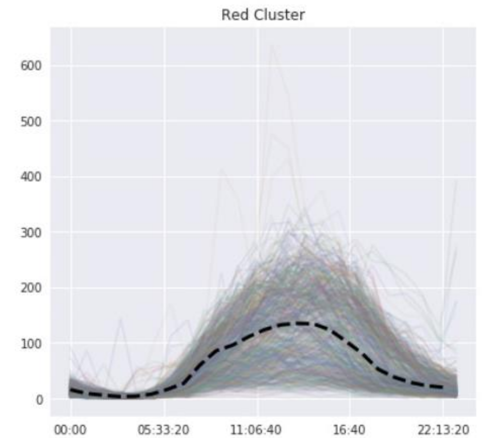


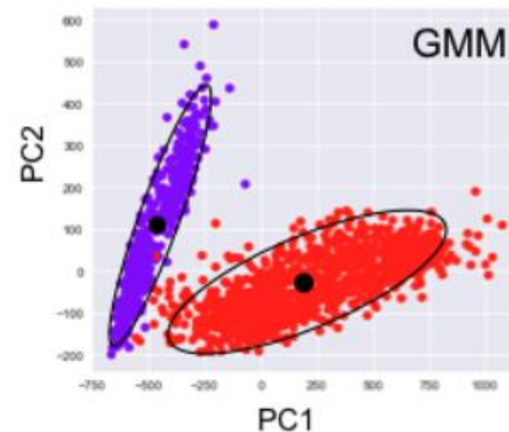
A key assumption: high-dimensional data lies on the low-dimensional manifold space



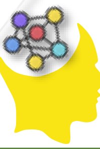
2128 data points, 24 dimensions

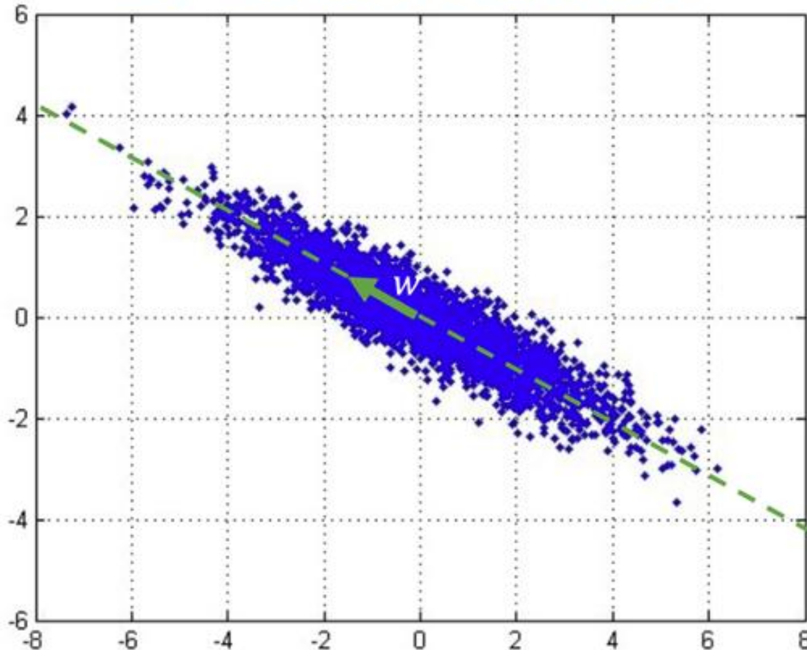






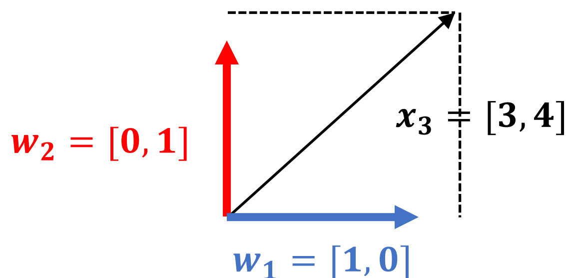
PCA - Variance Maximization





What would be a good reduction?

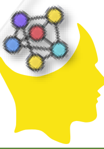
- Find w such that it maximizes the variance of the projected data
- Find w such that it minimizes the reconstruction error



$$\cos\theta = \frac{w_1^T x_3}{|w_1|_2 |x_3|_2}$$

$$\cos\theta |x_3|_2 = \frac{w_1^T x_3}{|w_1|_2} = w_1^T x_3$$

PCA - Variance Maximization

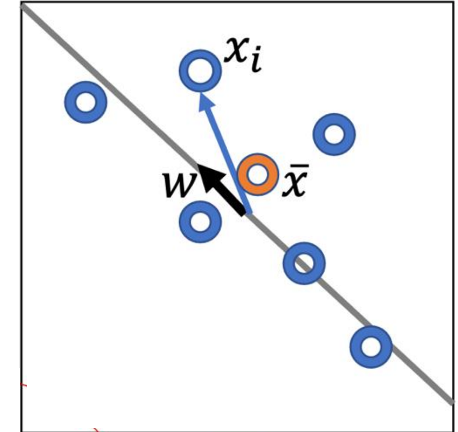


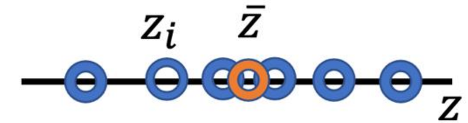
Find w such that it maximizes the variance of the projected data

$$var = \frac{1}{N}\sum_{i=1}^{N}(w^T(x_i - \bar{x}))^2 = \frac{1}{N}\sum_{i=1}^{N} w^T(x_i - \bar{x})(x_i - \bar{x})^T w$$

$$= w^T\left(\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})(x_i - \bar{x})^T\right)w$$

$$= w^T S w$$





Therefore maximizing the variance can be written as:

Constrained Optimization

$$\max_w \; w^T S w \quad s.t. \; w^T w = 1$$

KKT Condition →

$$Sw = \lambda w$$

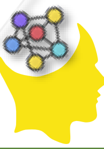
$$L(w, \alpha) = w^T S w + \alpha(w^T w - 1)$$

$$\nabla_w L(w, \alpha) = 2Sw + 2\alpha w = 0$$

$$\nabla_w L(w, \alpha) = Sw - \lambda w = 0$$

$$Sw = \lambda w$$

PCA - Variance Maximization



Some characteristics of the eigenvectors:

- $\|v_i\| = 1$
- $v_i^T v_j = 0, \quad \forall i \neq j$

Covariance matrix is a real and symmetric matrix (in fact it is PSD) therefore it can be uniquely decomposed via:

$$S = \sum_i \lambda_i v_i v_i^T$$

Multiply both sides by v_k:

$$S v_k = \lambda_k v_k$$

Therefore w should be the first eigenvector of S

$$Sw = \lambda w$$

$$\boldsymbol{\lambda_1} \geq \boldsymbol{\lambda_2} \geq \cdots \geq \boldsymbol{\lambda_n}$$

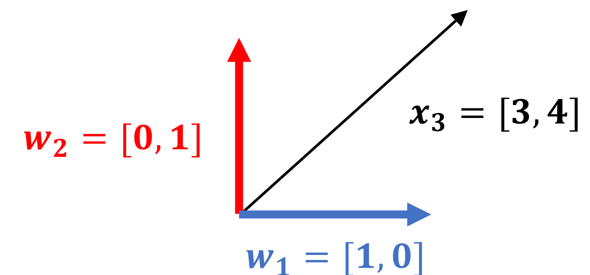
$$\boldsymbol{w_1} \geq \boldsymbol{w_2} \geq \cdots \geq \boldsymbol{w_n}$$

$$\boldsymbol{w_1^T x_1} = \boldsymbol{\sigma_1}$$

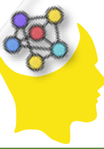
$$\boldsymbol{w_2^T x_1} = \boldsymbol{\sigma_2}$$

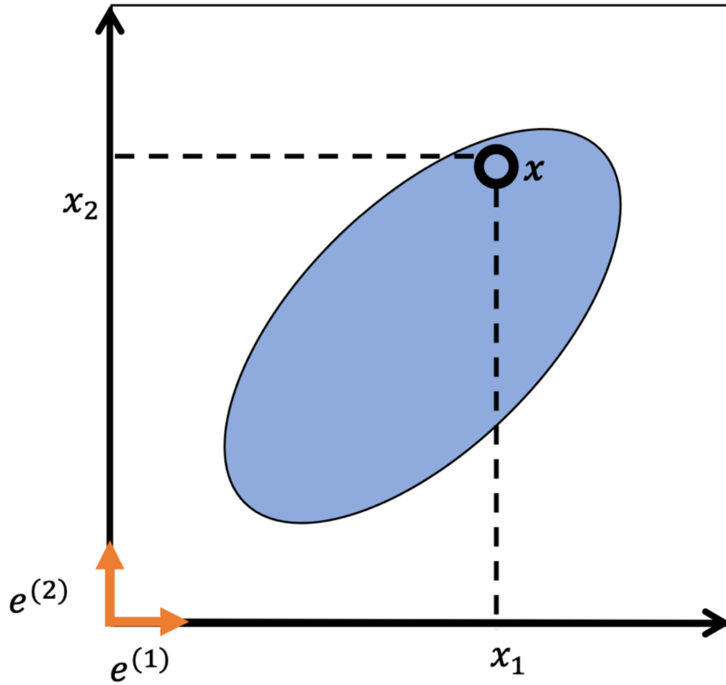
$$\vdots$$

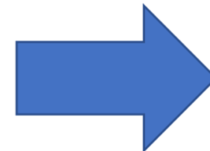
$$\boldsymbol{w_n^T x_1} = \boldsymbol{\sigma_n}$$

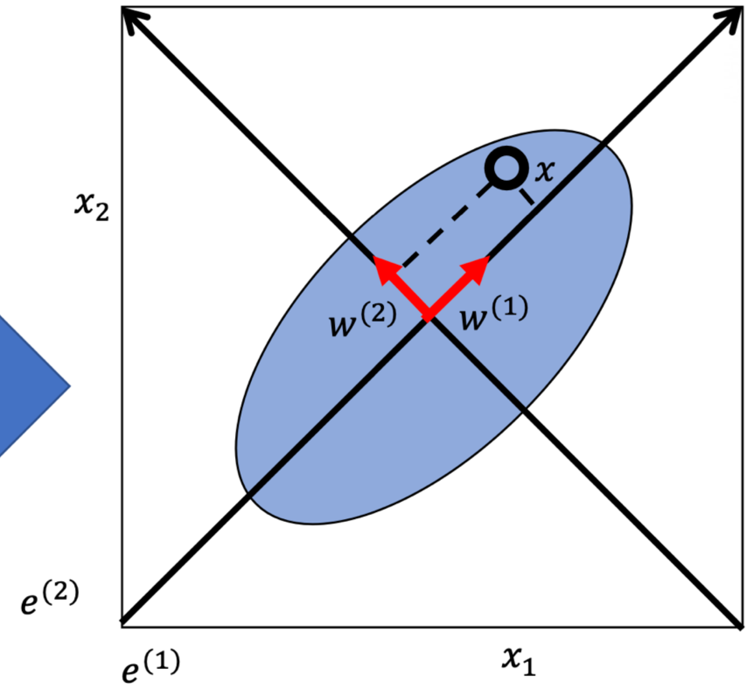


PCA - Variance Maximization





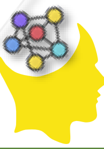




$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \Rightarrow x = x_1 \underbrace{\begin{bmatrix} 1 \\ 0 \end{bmatrix}}_{e^{(1)}} + x_2 \underbrace{\begin{bmatrix} 0 \\ 1 \end{bmatrix}}_{e^{(2)}}$$

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \Rightarrow x = \bar{x} + \left(x^T w^{(1)}\right) w^{(1)} + \left(x^T w^{(2)}\right) w^{(2)}$$

PCA - Variance Maximization



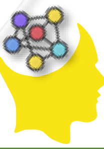
[11]:
```python
import torch
import numpy as np
import matplotlib.pyplot as plt
from torchvision import datasets, transforms

# Load MNIST dataset
transform = transforms.ToTensor()
mnist_data = datasets.MNIST(root='./data', train=True, download=True, transform=transform)
images = mnist_data.data.float()
labels = mnist_data.targets
```

[13]:
```python
# Flatten images to vectors of size 784
N, H, W = images.shape  # N=60000, H=28, W=28
X = images.view(N, H*W)  # shape: (60000, 784)

# Normalize data
mean_image = X.mean(dim=0)
X_centered = X - mean_image

# Compute covariance matrix
cov = (X_centered.T @ X_centered) / (N-1)

# Eigen-decomposition
eigenvalues, eigenvectors = torch.linalg.eigh(cov)
# Sort eigenvalues and eigenvectors in descending order
eigenvalues, indices = torch.sort(eigenvalues, descending=True)
eigenvectors = eigenvectors[:, indices]
```

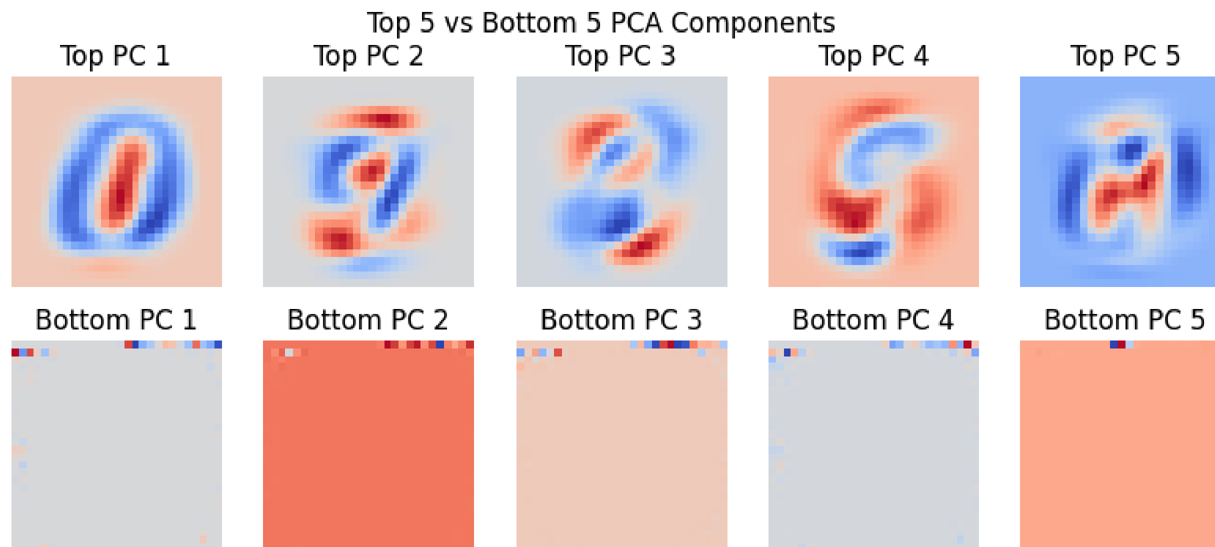
[14]:
```python
### Visualization 1: Original MNIST Dataset
fig, axes = plt.subplots(1, 10, figsize=(10, 2))
for i in range(10):
    axes[i].imshow(X[i].reshape(28, 28), cmap='gray')
    axes[i].axis('off')
plt.suptitle('Original MNIST Samples')
plt.show()
```

Original MNIST Samples

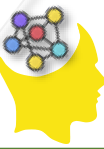
PCA - Variance Maximization



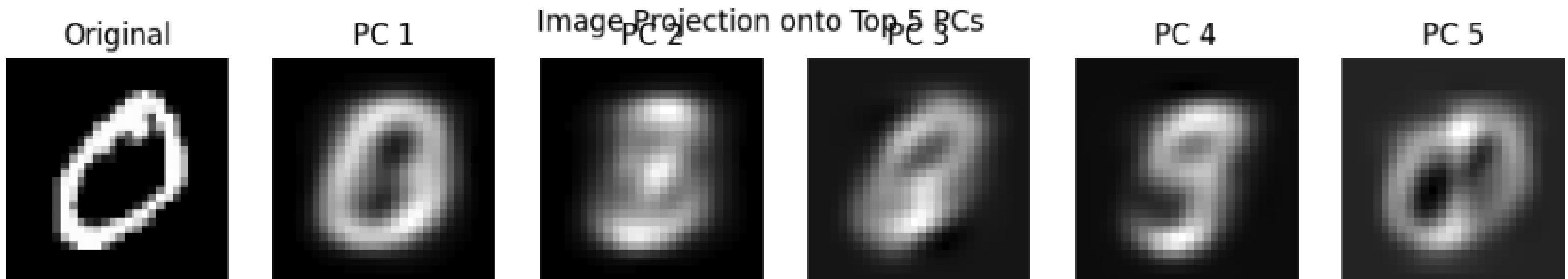
```python
### Visualization 2: Top 5 and Bottom 5 PCA Components
fig, axes = plt.subplots(2, 5, figsize=(10, 4))
for i in range(5):
    top_comp = eigenvectors[:, i].reshape(28, 28)
    bottom_comp = eigenvectors[:, -i-1].reshape(28, 28)

    axes[0, i].imshow(top_comp, cmap='coolwarm')
    axes[0, i].set_title(f'Top PC {i+1}')
    axes[0, i].axis('off')

    axes[1, i].imshow(bottom_comp, cmap='coolwarm')
    axes[1, i].set_title(f'Bottom PC {i+1}')
    axes[1, i].axis('off')
plt.suptitle('Top 5 vs Bottom 5 PCA Components')
plt.show()
```



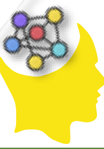
PCA - Variance Maximization



```python
### Visualization 4: Projecting a Specific Image onto PC1-5
sample_idx = 1  # Choose an image
test_image = X[sample_idx]
fig, axes = plt.subplots(1, 6, figsize=(12, 2))
axes[0].imshow(test_image.reshape(28, 28), cmap='gray')
axes[0].set_title('Original')
axes[0].axis('off')

for i in range(5):
    weight = torch.dot(test_image - mean_image, eigenvectors[:, i])
    recon = weight * eigenvectors[:, i] + mean_image
    axes[i+1].imshow(recon.reshape(28, 28), cmap='gray')
    axes[i+1].set_title(f'PC {i+1}')
    axes[i+1].axis('off')
plt.suptitle('Image Projection onto Top 5 PCs')
plt.show()
```

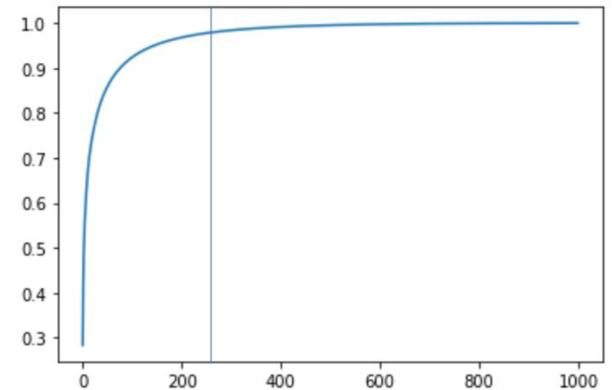


PCA - Variance Maximization

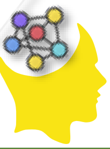


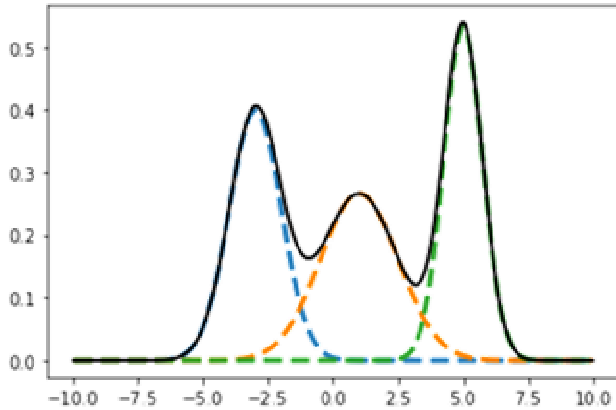
Input x = Mean $\bar{x}$ + 341.6* $w^{(1)}$ - 12.7* $w^{(2)}$ +... + 12.2* $w^{(1000)}$

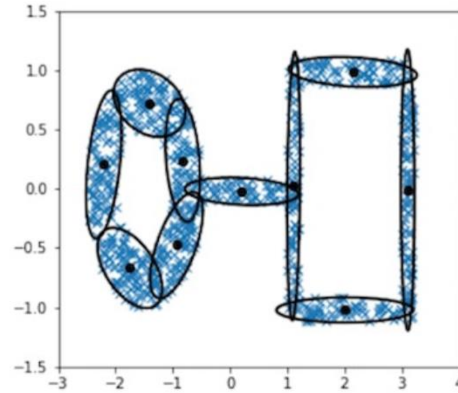
Reconstruction as a function of number of PC components



PCA to reduce dimension and then GDE/GMM





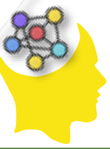


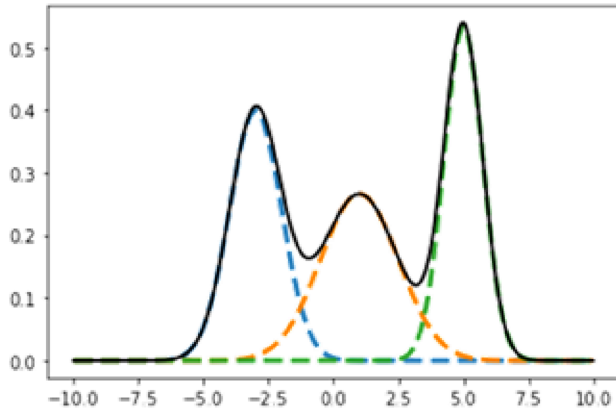
$\mathbb{R}^1, \mathbb{R}^2$

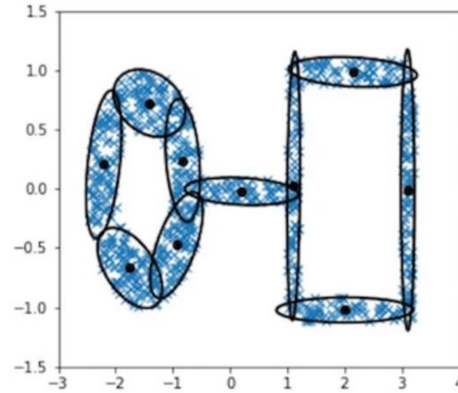
$\mathbb{R}^{256\times256}$

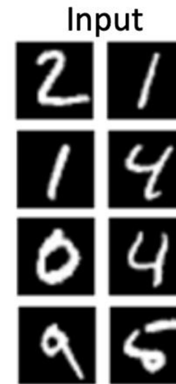
High-Dimensional Data

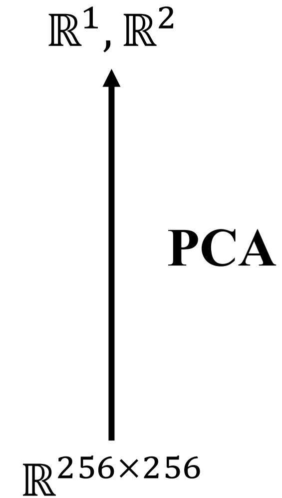
PCA to reduce dimension and then GDE/GMM





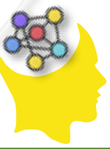


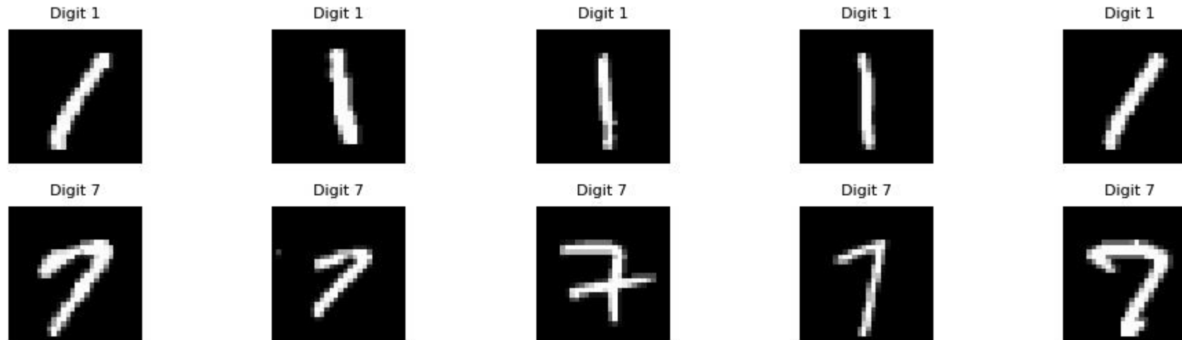


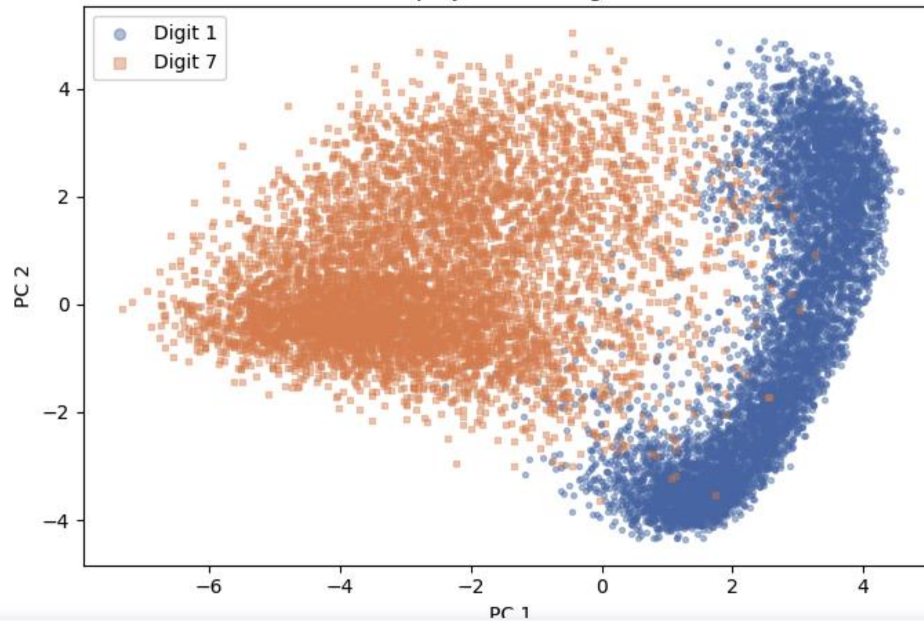


PCA + GDE/GMM

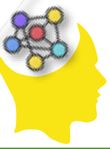
PCA to reduce dimension and then GDE/GMM







PCA to reduce dimension and then GDE/GMM

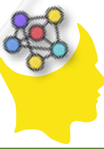


PC 2

PC 1

Comp 0 (π=0.21)
Comp 1 (π=0.26)
Comp 2 (π=0.25)
Comp 3 (π=0.29)

Connection between PCA and Reconstruction Loss



$$\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbb{R}^D$$

$$z_i = \mathbf{w}^\top \mathbf{x}_i \in \mathbb{R}, \qquad \hat{\mathbf{x}}_i = z_i \mathbf{w} = (\mathbf{w}^\top \mathbf{x}_i)\mathbf{w} \in \mathbb{R}^D$$

$$\mathbf{w} \in \mathbb{R}^D, \|\mathbf{w}\| = 1,$$

$$\max_{\mathbf{w},\, \|\mathbf{w}\|=1} \frac{1}{N} \sum_{i=1}^{N} z_i^2 = \mathbf{w}^\top \underbrace{\left(\frac{1}{N} \sum_i \mathbf{x}_i \mathbf{x}_i^\top \right)}_{\mathbf{S}} \mathbf{w} = \mathbf{w}^\top \mathbf{S} \mathbf{w}$$

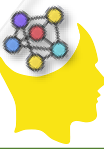
Maximize Variance

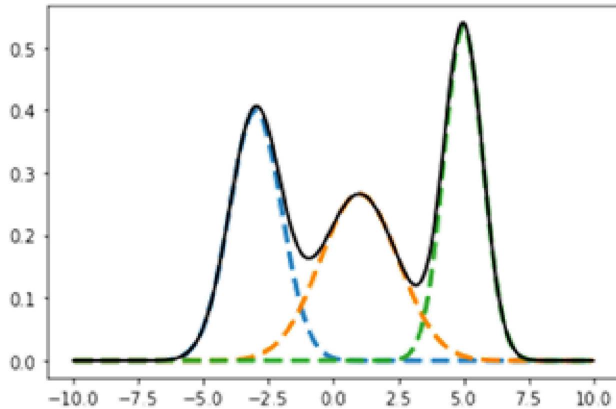
$$\min_{\mathbf{w},\, \|\mathbf{w}\|=1} \sum_{i=1}^{N} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 \qquad \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 = \mathbf{x}_i^\top \mathbf{x}_i - 2(\mathbf{w}^\top \mathbf{x}_i)(\mathbf{w}^\top \mathbf{x}_i) + (\mathbf{w}^\top \mathbf{x}_i)^2 \underbrace{\mathbf{w}^\top \mathbf{w}}_{=1} = \|\mathbf{x}_i\|^2 - (\mathbf{w}^\top \mathbf{x}_i)^2$$

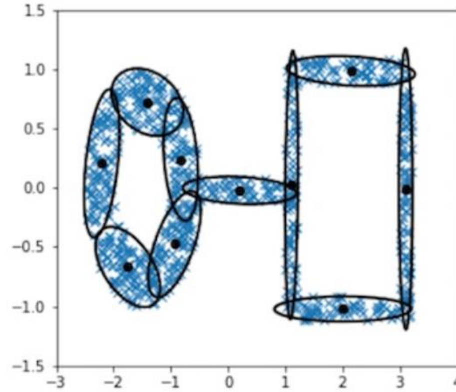
$$\sum_{i=1}^{N} \|\mathbf{x}_i - \hat{\mathbf{x}}_i\|^2 = \underbrace{\sum_{i=1}^{N} \|\mathbf{x}_i\|^2}_{\text{constant w.r.t. } \mathbf{w}} - \sum_{i=1}^{N} (\mathbf{w}^\top \mathbf{x}_i)^2 \qquad \min_{\mathbf{w}} \left[\text{const} - \sum_i (\mathbf{w}^\top \mathbf{x}_i)^2 \right] \equiv \max_{\mathbf{w}} \sum_i (\mathbf{w}^\top \mathbf{x}_i)^2 = \max_{\mathbf{w}} \mathbf{w}^\top \mathbf{S} \mathbf{w}$$

Minimize Reconstruction Loss

PCA = Dimension Reduction + Reconstruction







Input

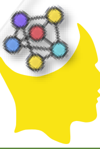
$\mathbb{R}^1, \mathbb{R}^2$

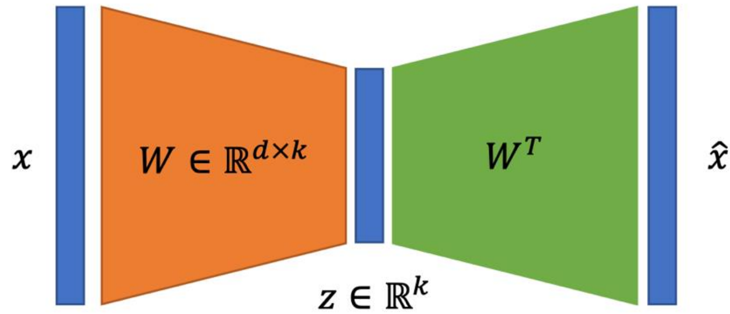
PCA

$\mathbb{R}^{256\times256}$

PCA – Reduce Dimension/Reconstruction
GDE/GMM – Latent Distribution Modeling
GenAI

From PCA to AutoEncoder

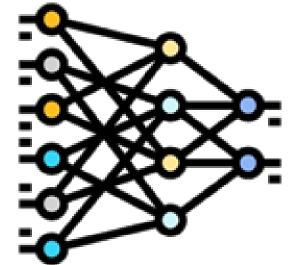


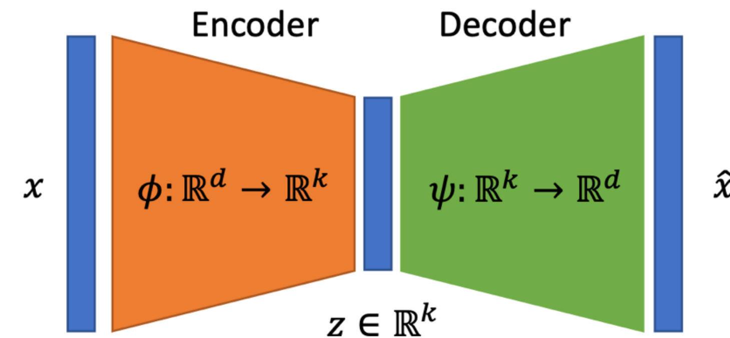


PCA:

- Forward transform: $z = W^T x$
- Inverse transform: $\hat{x} = Wz$

$$\min_W \mathbb{E}_x[\|x - \hat{x}\|^2] = \mathbb{E}_x[\| x - WW^T x\|^2]$$
$$s.t. \quad W^T W = I_{k\times k}$$



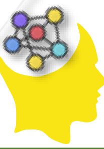


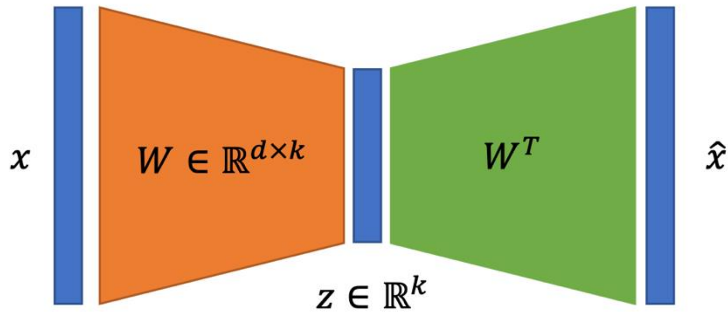
AE:

- Forward transform: $z = \phi(x)$
- Inverse transform: $\hat{x} = \psi(z)$

$$\min_{\phi,\psi} \mathbb{E}_x[\|x - \hat{x}\|^2] = \mathbb{E}_x\left[\left\| x - \psi(\phi(x))\right\|^2\right]$$

Linear Layer PCA is AutoEncoder





PCA:

- Forward transform: $z = W^T x$
- Inverse transform: $\hat{x} = Wz$

Linear dimensionality Reduction

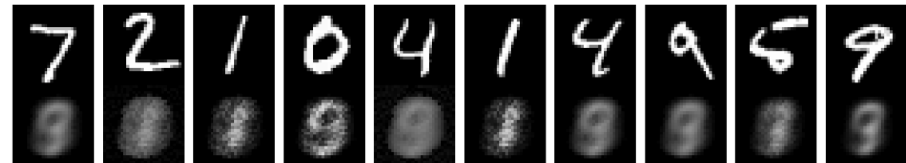
$$\min_W \mathbb{E}_x[\|x - \hat{x}\|^2] = \mathbb{E}_x[\| x - WW^T x\|^2]$$
$$s.t. \quad W^T W = I_{k \times k}$$

```
# Linear AE: encoder 784->2 (linear), decoder 2->784 (linear)
# No hidden layers, no activations  -->  equivalent to PCA
class LinearAE_MNIST(nn.Module):
    def __init__(self, d=2):
        super().__init__()
        self.encoder = nn.Sequential(nn.Flatten(), nn.Linear(784, d, bias=False))
        self.decoder = nn.Sequential(nn.Linear(d, 784, bias=False), nn.Sigmoid())
    def forward(self, x):
        z = self.encoder(x)
        return self.decoder(z).view(-1, 1, 28, 28), z
```

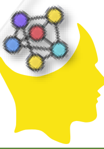
```
print("\nTraining Linear AE (= PCA via gradient descent) ...")
for epoch in range(10):
    lin_ae.train()
    total = 0
    for imgs, _ in train_dl:
        imgs = imgs.to(device)
        opt.zero_grad()
        x_hat, _ = lin_ae(imgs)
        loss = ((imgs - x_hat)**2).mean()
        loss.backward(); opt.step()
        total += loss.item()
    print("  Epoch {:2d}  loss = {:.5f}".format(epoch+1, total/len(train_dl)))
```

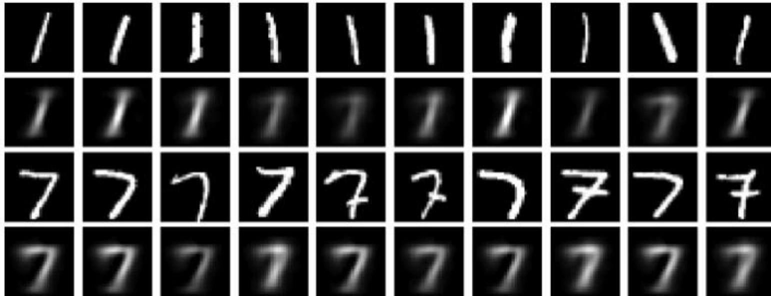
```
Training Linear AE (= PCA via gradient descent)
  Epoch  1  loss = 0.19154
  Epoch  2  loss = 0.08741
  Epoch  3  loss = 0.06887
  Epoch  4  loss = 0.06789
  Epoch  5  loss = 0.06727
  Epoch  6  loss = 0.06677
  Epoch  7  loss = 0.06626
  Epoch  8  loss = 0.06580
  Epoch  9  loss = 0.06535
  Epoch 10  loss = 0.06488
```

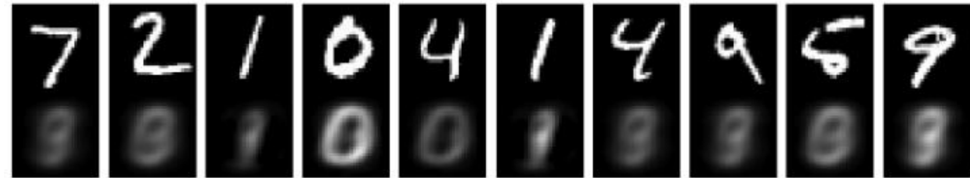
Linear AE on MNIST (d=2) — reconstructions are blurry and washed out
Reason: a 2D linear subspace cannot capture the curved digit manifold

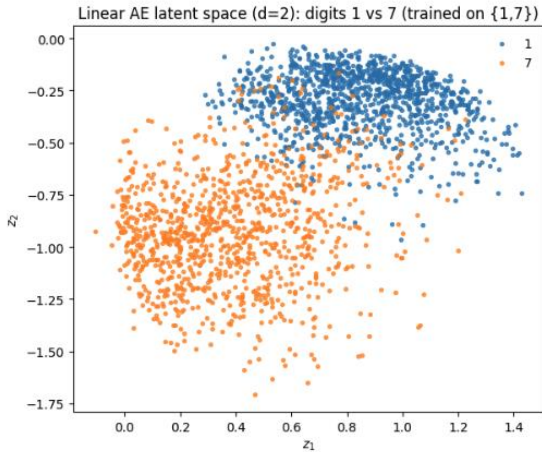


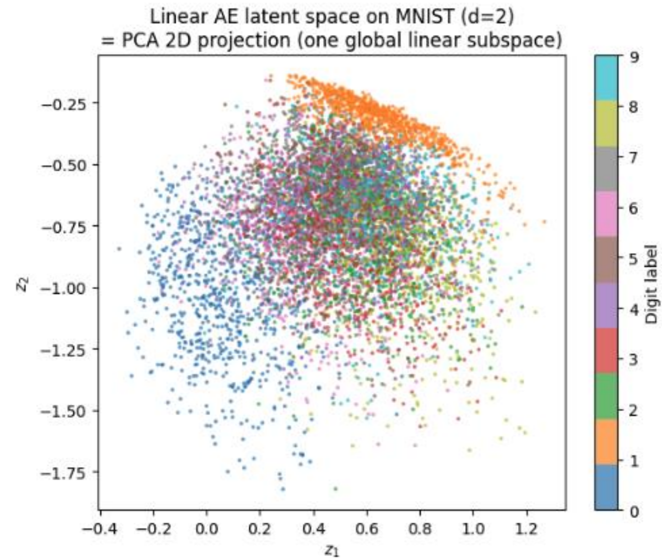
Linear Layer Autoencoder/PCA is limited







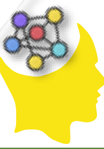


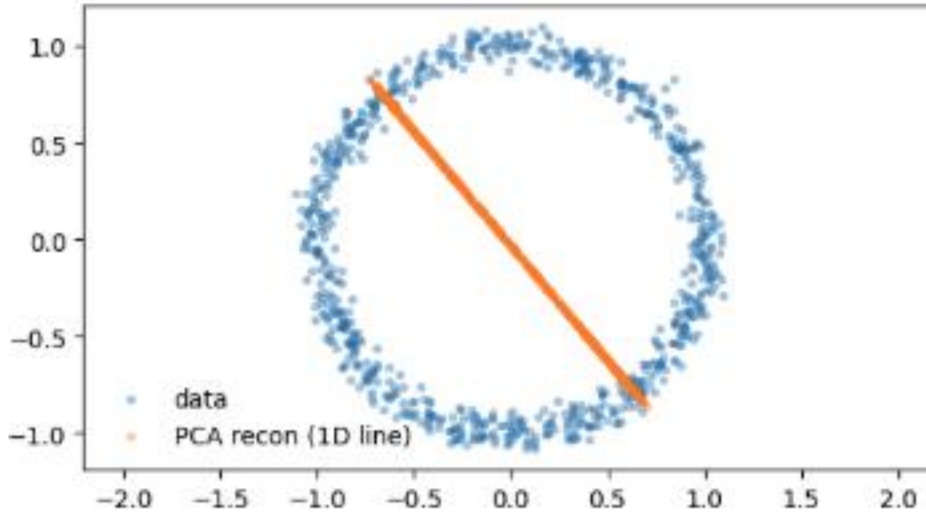


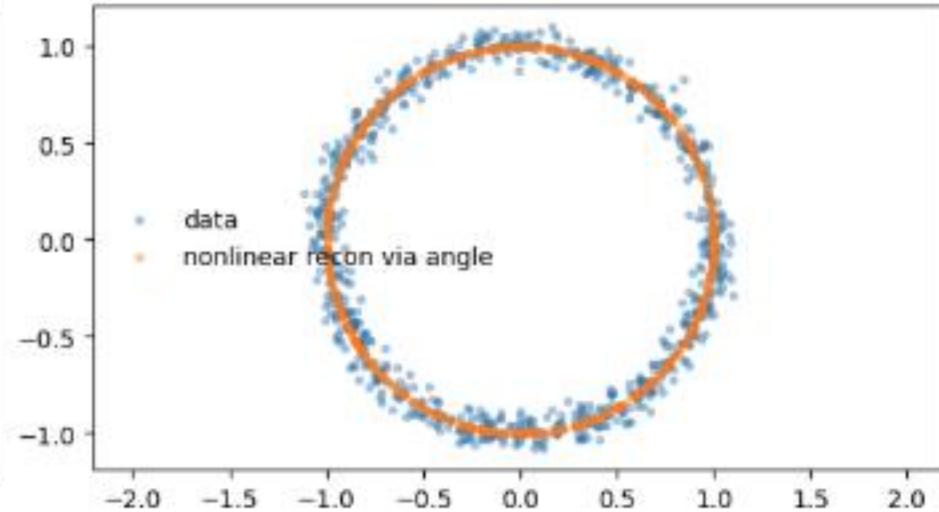
Linear Layer cannot separate

Linear Layer cannot separate

Linear Layer Autoencoder/PCA is limited



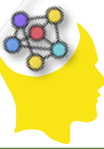


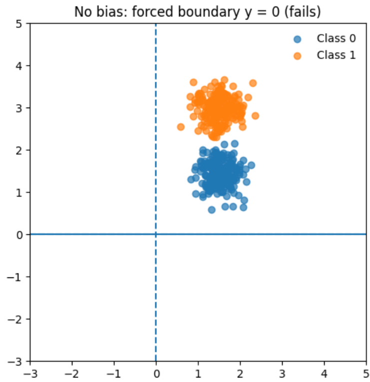


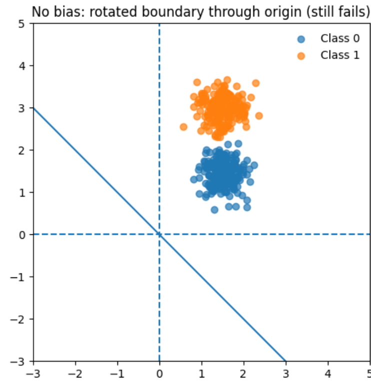
PCA Reconstruction
Linear Layer

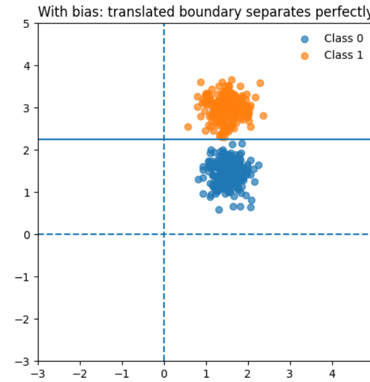
Nonlinear Manifold

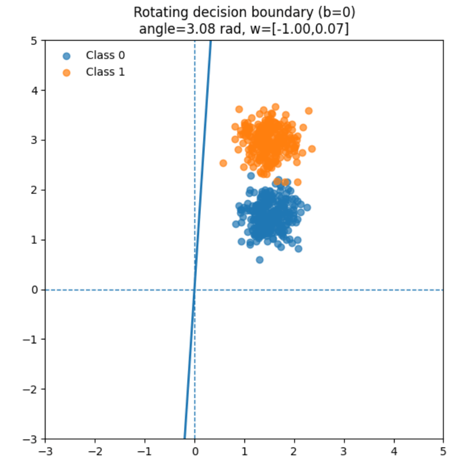
Nonlinear Layer can bend space



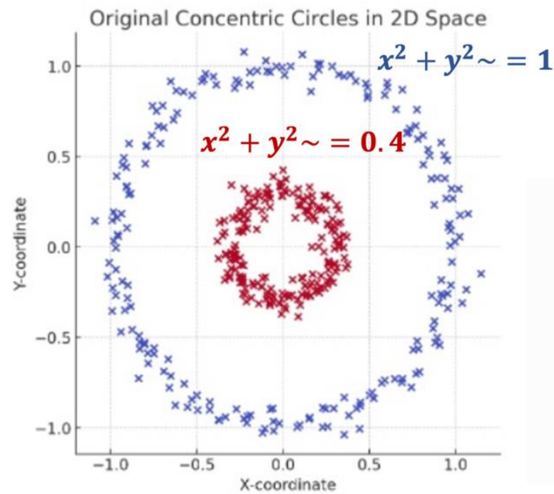


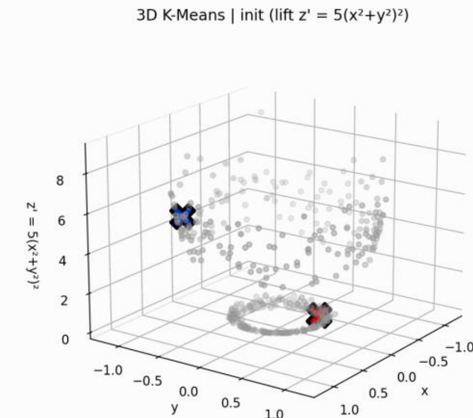




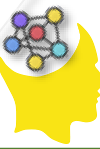


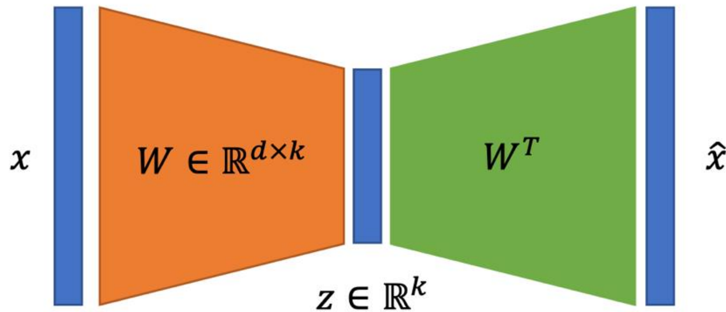
$$y = \sigma(w^T x + \mathbf{b})$$





From PCA to AutoEncoder





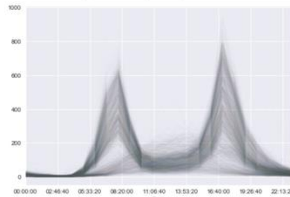
PCA:

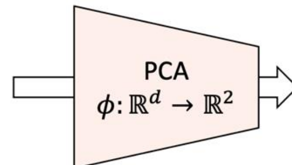
- Forward transform: $z = W^T x$
- Inverse transform: $\hat{x} = Wz$

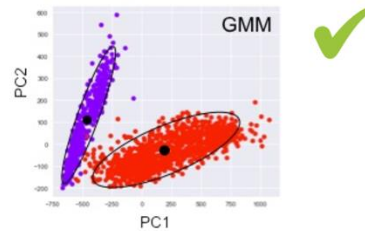
Linear dimensionality Reduction

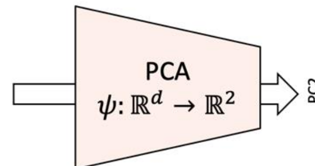
$$\min_W \mathbb{E}_x[\|x - \hat{x}\|^2] = \mathbb{E}_x[\| x - WW^T x\|^2]$$
$$s.t. \quad W^T W = I_{k \times k}$$

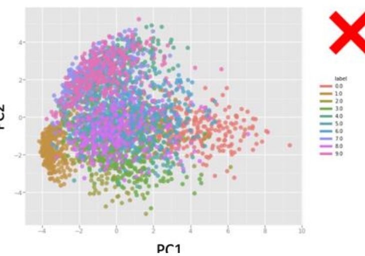
High-dimensional data often lives on non-linear manifolds that cannot be captured by linear models such as PCA







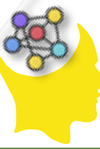


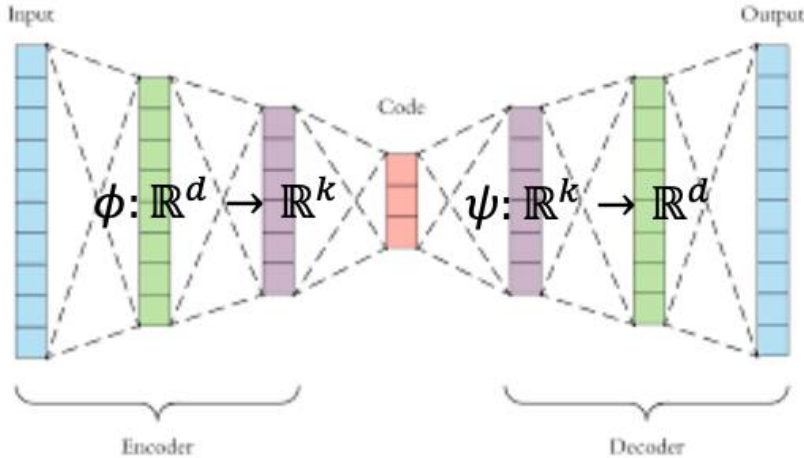


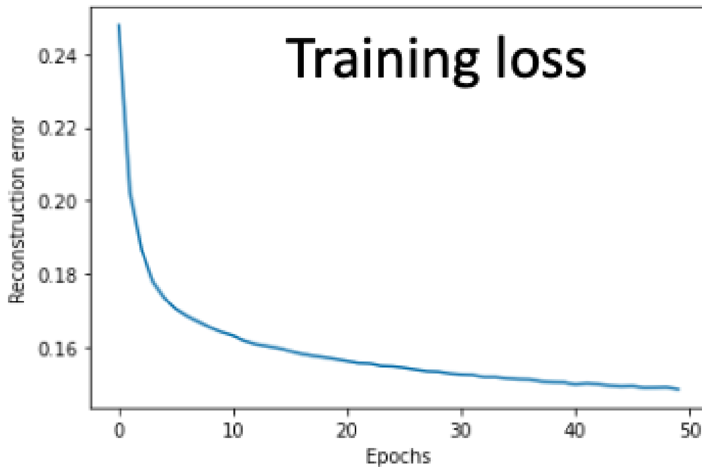
Can we add nonlinearity?

Yes, then it becomes neural network!

AutoEncoder

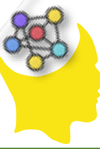




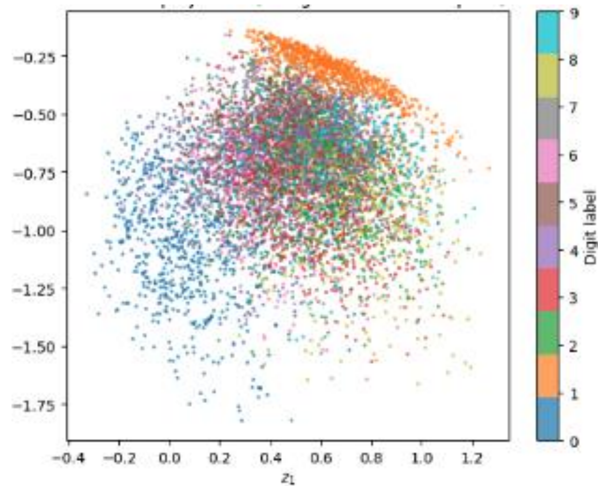


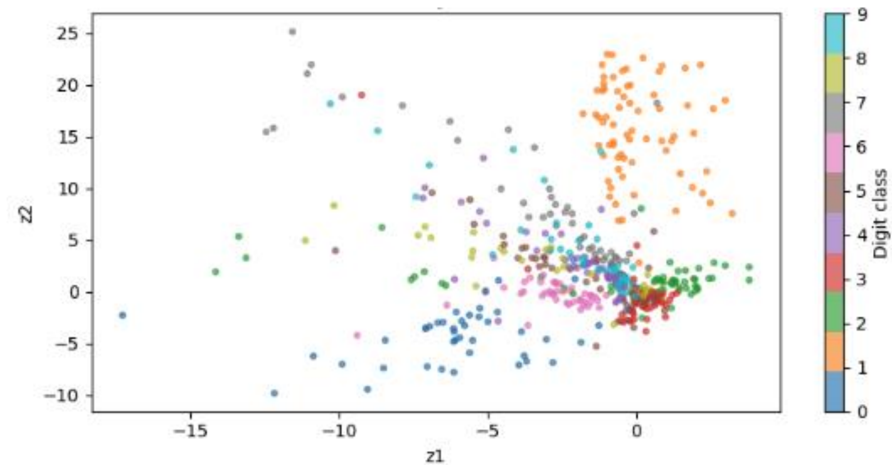

```python
class MLP_AE(nn.Module):
  def __init__(self,architecture=[784,128,64,2],activation='LeakyReLU'):
        super(MLP_AE, self).__init__()
        self.architecture=architecture
        if activation=='LeakyReLU':
            self.activation=nn.LeakyReLU()
        elif activation=='ReLU':
            self.activation=nn.ReLU()
        elif activation=='Sigmoid':
            self.activation=nn.Sigmoid()
        else:
            print('Activation not defined, reverting to default!')
            self.activation=nn.LeakyReLU()
        # Defining $\phi$
        arch=[]
        for i in range(1,len(architecture)):
            arch.append(nn.Linear(architecture[i-1],architecture[i]))
            if i!=len(architecture)-1:
                arch.append(self.activation)
        self.encoder=nn.Sequential(*arch)
        # Defining $\psi$
        arch=[]
        for i in range(len(architecture)-1,0,-1):
            arch.append(nn.Linear(architecture[i],architecture[i-1]))
            if i!=1:
                arch.append(self.activation)
        self.decoder=nn.Sequential(*arch)

  def encode(self,f):
        assert f.shape[1]==self.architecture[0]
        return self.encoder(f)

  def decode(self,fhat):
        assert fhat.shape[1]==self.architecture[-1]
        return self.decoder(fhat)

  def forward(self,x):
    return self.decode(self.encode(x))
```

AutoEncoder

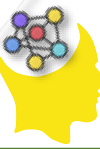


Original vs Linear vs Nonlinear





AutoEncoder + Supervised Classification



Input

Latent representation

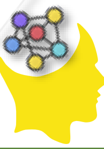
Output

Encoder
$\phi(\cdot;\theta_\phi): \mathbb{R}^d \to \mathbb{R}^k$
e.g. NN

Decoder
$\psi(\cdot,\theta_\psi): \mathbb{R}^k \to \mathbb{R}^d$
e.g. NN

Need to estimate the latent distribution post-hoc!

AutoEncoder + Supervised Classification



$$\mathcal{L}_{\text{total}} = \underbrace{\mathcal{L}_{\text{recon}}}_{\text{reconstruct } x} + \lambda \underbrace{\mathcal{L}_{\text{cls}}}_{\text{predict label}}$$

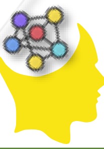
The encoder is now pulled in **two directions**:

Loss	What it forces z to encode
Reconstruction	pixel-level detail — shape, stroke, texture
Classification	class identity — which digit it is

```python
# AE + classification head
class AE_Clf(nn.Module):
    """Same encoder/decoder as AE, plus a linear classification head on z."""
    def __init__(self, latent_dim=2, n_classes=10):
        super().__init__()
        self.encoder  = Encoder(latent_dim)          # reuse from Part 2
        self.decoder  = Decoder(latent_dim)          # reuse from Part 2
        self.clf_head = nn.Linear(latent_dim, n_classes)
    def forward(self, x):
        z      = self.encoder(x)
        x_hat  = self.decoder(z)
        logits = self.clf_head(z)
        return x_hat, z, logits
```

```python
print("Training AE + prediction loss  (recon + {:.0f} x cross-entropy)".format(lam))
for epoch in range(10):
    ae_clf.train()
    t_recon = t_cls = 0
    for imgs, labels in train_dl:
        imgs, labels = imgs.to(device), labels.to(device)
        opt_clf.zero_grad()
        x_hat, z, logits = ae_clf(imgs)
        loss_r = ((imgs - x_hat)**2).mean()
        loss_c = ce_loss(logits, labels)
        loss   = loss_r + lam * loss_c
        loss.backward(); opt_clf.step()
        t_recon += loss_r.item(); t_cls += loss_c.item()
```

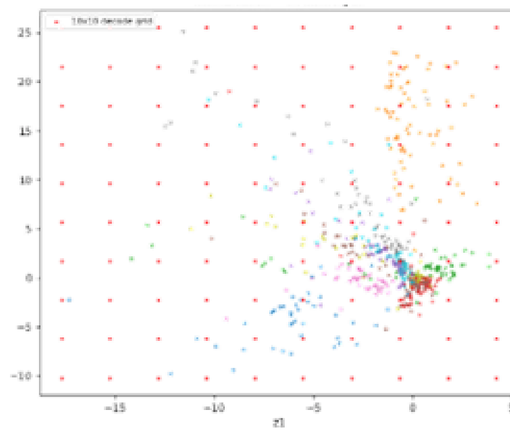
AutoEncoder + Supervised Classification

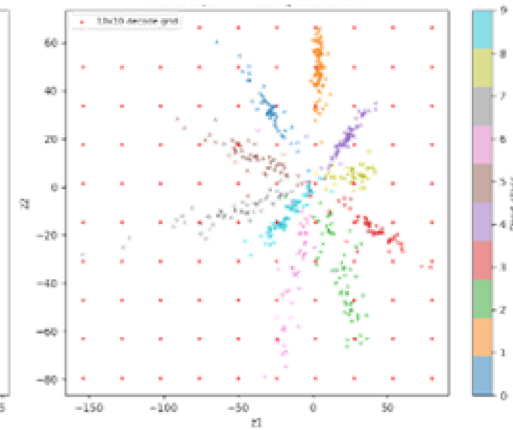


$$\mathcal{L}_{\text{total}} = \underbrace{\mathcal{L}_{\text{recon}}}_{\text{reconstruct } x} + \lambda \underbrace{\mathcal{L}_{\text{cls}}}_{\text{predict label}}$$

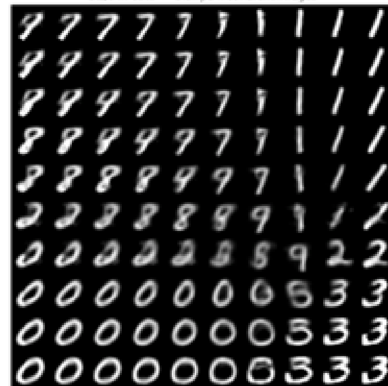
The encoder is now pulled in **two directions**:

Loss	What it forces z to encode
Reconstruction	pixel-level detail — shape, stroke, texture
Classification	class identity — which digit it is



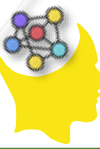


Decoded images at red grid points (pure AE)
Gaps in data -> blurry / incoherent images



Decoded images at red grid points (AE + clf)
Grid spans class clusters -> more recognisable digits

Problem of AutoEncoder



Input

Latent representation

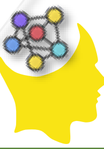
Output

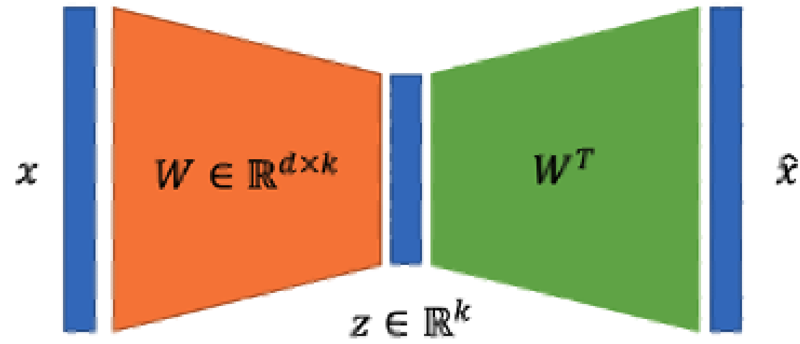
Encoder
$\phi(\cdot;\theta_\phi): \mathbb{R}^d \rightarrow \mathbb{R}^k$
e.g. NN

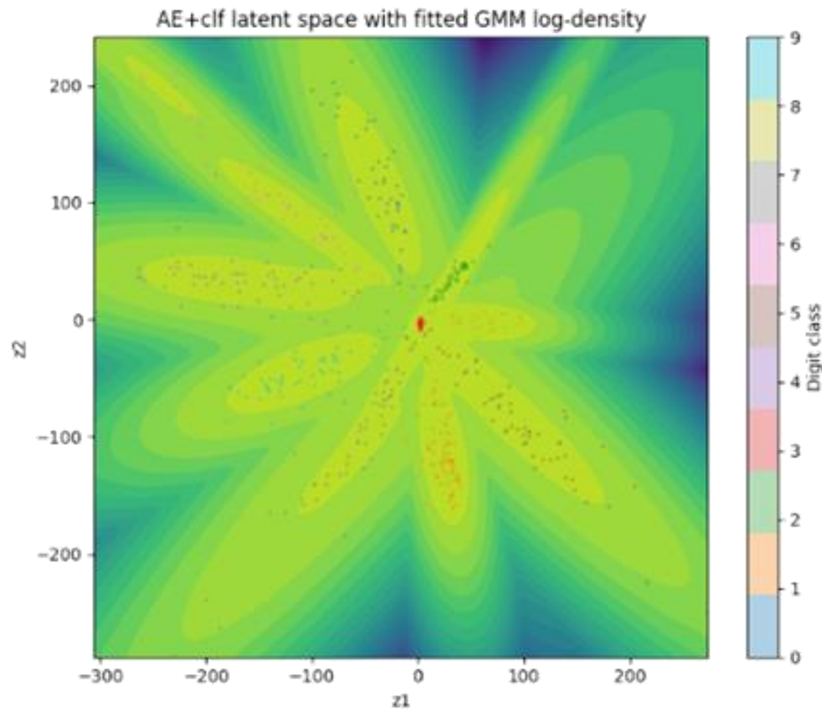
Decoder
$\psi(\cdot,\theta_\psi): \mathbb{R}^k \rightarrow \mathbb{R}^d$
e.g. NN

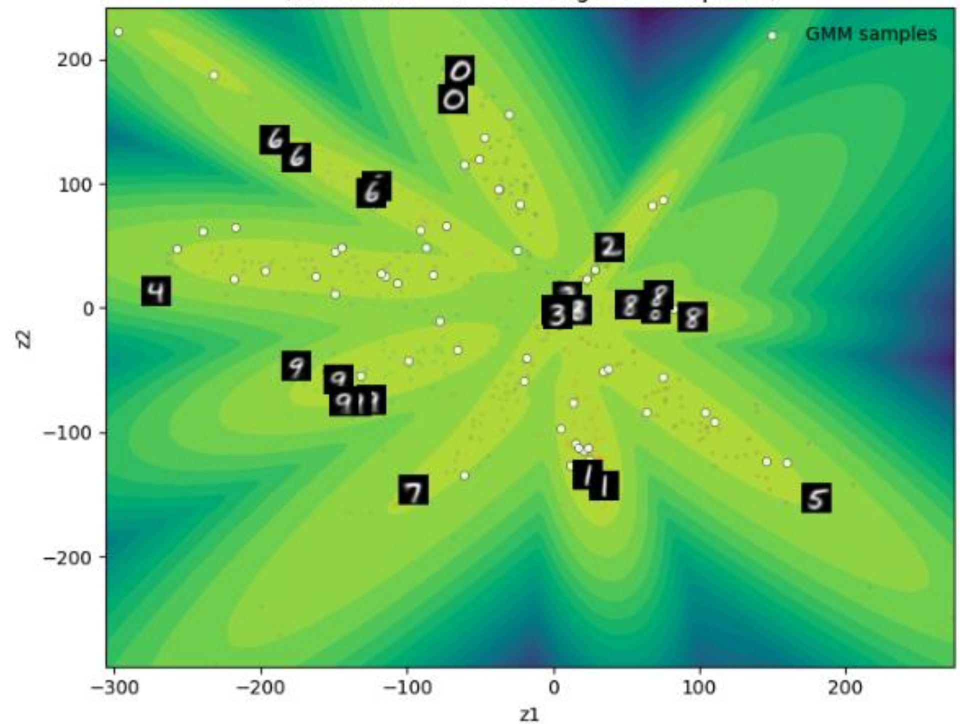
How to use AE to do Generation? **You might need to estimate latent distribution**

Problem of AutoEncoder

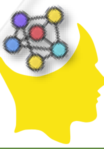








Motivation of Variational AutoEncoder



encoder

Smile: 0.99

Skin tone: 0.85

Gender: -0.73

Beard: 0.85

Glasses: 0.002

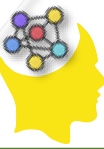
Hair color: 0.68

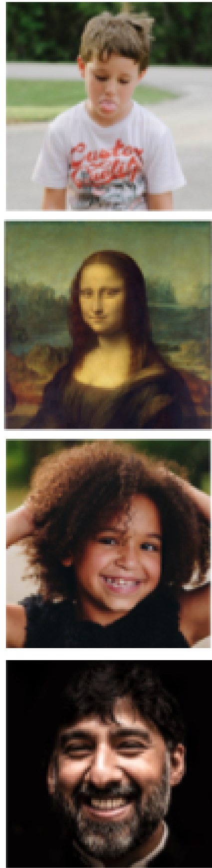
decoder

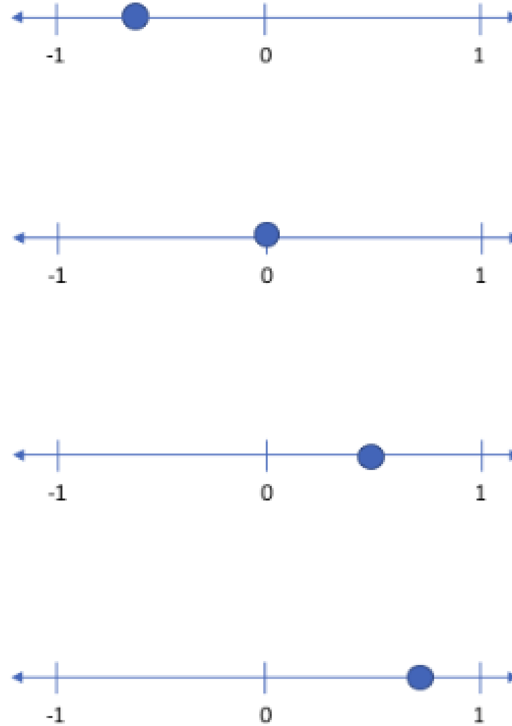
Latent attributes

AE model latent space point-to-point

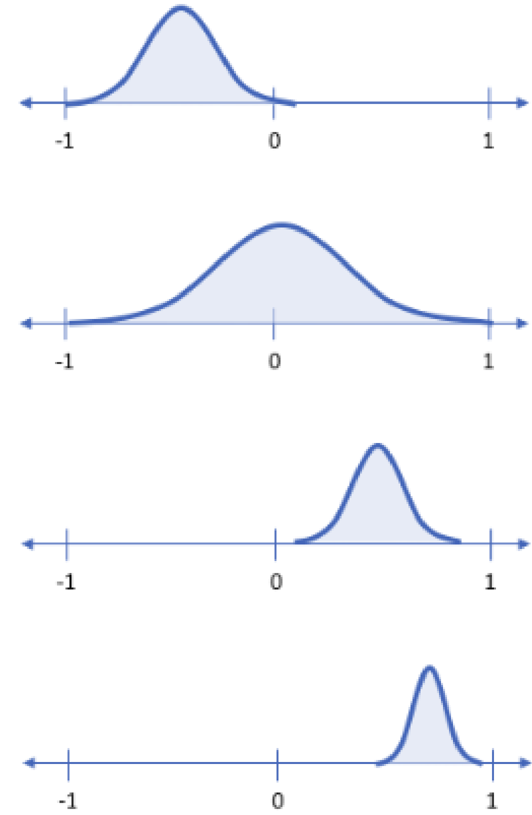
Motivation of Variational AutoEncoder





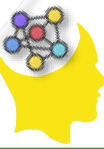


vs.



How about we modeling latent space point-to-distribution?

Motivation of Variational AutoEncoder



encoder

Smile:
Skin tone:
Gender:
Beard:
Glasses:
Hair color:

Latent distributions

sample

sample

Smile: 0.23
Skin tone: 0.02
Gender: -0.18
Beard: 0.71
Glasses: -0.19
Hair color: 0.33

Smile: 0.17
Skin tone: 0.28
Gender: -0.11
Beard: 0.66
Glasses: -0.14
Hair color: 0.26

Sampled latent attributes

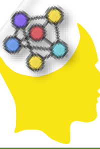
decoder

decoder

We expect an accurate reconstruction for any sample from the latent state distributions

And then we can sample from the latent distribution and regenerate the iimage

Variational AutoEncoder



encoder

Smile: -1 0 1

Skin tone: -1 0 1

Gender: -1 0 1

Beard: -1 0 1

Glasses: -1 0 1

Hair color: -1 0 1

Latent distributions

sample → Smile: 0.23, Skin tone: 0.02, Gender: -0.18, Beard: 0.71, Glasses: -0.19, Hair color: 0.33 → decoder

sample → Smile: 0.17, Skin tone: 0.28, Gender: -0.11, Beard: 0.66, Glasses: -0.14, Hair color: 0.26 → decoder

Sampled latent attributes

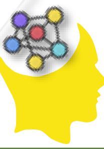
We expect an accurate reconstruction for any sample from the latent state distributions

- $\mu_\phi(x) \in \mathbb{R}^d$ — the mean (where the code should be)
- $\log \sigma^2_\phi(x) \in \mathbb{R}^d$ — the log-variance (how uncertain we are)

Modeling a Distribution

For Encoder $z = f_\theta(x) \longrightarrow q_\phi(z \mid x) = \mathcal{N}\left(\mu_\phi(x),\ \sigma^2_\phi(x)\right)$

Variational AutoEncoder



encoder

Smile: Skin tone: Gender: Beard: Glasses: Hair color:

Latent distributions

sample → Smile: 0.23, Skin tone: 0.02, Gender: -0.18, Beard: 0.71, Glasses: -0.19, Hair color: 0.33 → decoder

sample → Smile: 0.17, Skin tone: 0.28, Gender: -0.11, Beard: 0.66, Glasses: -0.14, Hair color: 0.26 → decoder

Sampled latent attributes

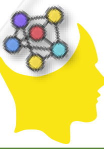
We expect an accurate reconstruction for any sample from the latent state distributions

Sampling from Latent Distribution

For Sampling $q_\phi(z \mid x) = \mathcal{N}\left(\mu_\phi(x),\ \sigma^2_\phi(x)\right) \longrightarrow z \sim q_\phi(z|x)$

$$z = \mu_\phi(x) + \sigma_\phi(x) \odot \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I)$$

Variational AutoEncoder



encoder

Smile: Skin tone: Gender: Beard: Glasses: Hair color:

-1 0 1

Latent distributions

sample

sample

Smile: 0.23
Skin tone: 0.02
Gender: -0.18
Beard: 0.71
Glasses: -0.19
Hair color: 0.33

Smile: 0.17
Skin tone: 0.28
Gender: -0.11
Beard: 0.66
Glasses: -0.14
Hair color: 0.26

Sampled latent attributes

decoder

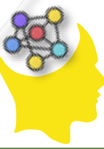
decoder

We expect an accurate reconstruction for any sample from the latent state distributions

Decoding

$$z \sim q_\phi(z|x) \xrightarrow{\textbf{Decode}} p_\theta(x|z)$$

Variational AutoEncoder



encoder

Smile: Skin tone: Gender: Beard: Glasses: Hair color:

Latent distributions

sample → Smile: 0.23, Skin tone: 0.02, Gender: -0.18, Beard: 0.71, Glasses: -0.19, Hair color: 0.33 → decoder

sample → Smile: 0.17, Skin tone: 0.28, Gender: -0.11, Beard: 0.66, Glasses: -0.14, Hair color: 0.26 → decoder

Sampled latent attributes

We expect an accurate reconstruction for any sample from the latent state distributions

$$q_\phi(z \mid x) = \mathcal{N}\left(\mu_\phi(x),\ \sigma_\phi^2(x)\right)$$

$$\downarrow$$

$$z \sim q_\phi(z|x)$$

$$\downarrow$$

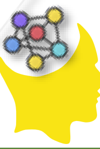
$$p_\theta(x|z)$$

Training Stage

Reconstruction Well

Latent Space can support sampling

Variational AutoEncoder



encoder

Latent distributions: Smile, Skin tone, Gender, Beard, Glasses, Hair color

sample → Smile: 0.23, Skin tone: 0.02, Gender: -0.18, Beard: 0.71, Glasses: -0.19, Hair color: 0.33 → decoder

sample → Smile: 0.17, Skin tone: 0.28, Gender: -0.11, Beard: 0.66, Glasses: -0.14, Hair color: 0.26 → decoder

Latent distributions

Sampled latent attributes

We expect an accurate reconstruction for any sample from the latent state distributions

Training Stage

$$q_\phi(z \mid x) = \mathcal{N}\left(\mu_\phi(x),\ \sigma^2_\phi(x)\right)$$

$$\downarrow$$

$$z \sim q_\phi(z|x)$$

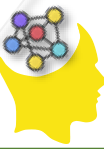
$$\downarrow$$

$$p_\theta(x|z)$$

Reconstruction Well

$$\underbrace{\mathbb{E}_{q_\phi}[\log p_\theta(x|z)]}_{\text{reconstruction}}$$

Variational AutoEncoder



encoder

Smile: Skin tone: Gender: Beard: Glasses: Hair color:

Latent distributions

sample → Smile: 0.23, Skin tone: 0.02, Gender: -0.18, Beard: 0.71, Glasses: -0.19, Hair color: 0.33 → decoder

sample → Smile: 0.17, Skin tone: 0.28, Gender: -0.11, Beard: 0.66, Glasses: -0.14, Hair color: 0.26 → decoder

Sampled latent attributes

We expect an accurate reconstruction for any sample from the latent state distributions

Training Stage

$$q_\phi(z \mid x) = \mathcal{N}\left(\mu_\phi(x),\ \sigma^2_\phi(x)\right)$$

$$\downarrow$$

$$z \sim q_\phi(z|x)$$

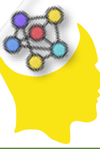
$$\downarrow$$

$$p_\theta(x|z)$$

Latent Space Support Sampling

$$D_{\mathrm{KL}}(q_\phi(z|x) \,\|\, \mathcal{N}(0, I))$$

Variational AutoEncoder



encoder

Smile:
Skin tone:
Gender:
Beard:
Glasses:
Hair color:

-1 0 1

Latent distributions

sample

Smile: 0.23
Skin tone: 0.02
Gender: -0.18
Beard: 0.71
Glasses: -0.19
Hair color: 0.33

sample

Smile: 0.17
Skin tone: 0.28
Gender: -0.11
Beard: 0.66
Glasses: -0.14
Hair color: 0.26

Sampled latent attributes

decoder

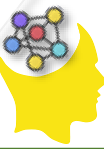
decoder

We expect an accurate reconstruction for any sample from the latent state distributions

Training Stage

$$\underbrace{\mathbb{E}_{q_\phi}[\log p_\theta(x|z)]}_{\text{reconstruction}} - \underbrace{D_{\mathrm{KL}}(q_\phi(z|x) \,\|\, p(z))}_{\text{regularise toward prior}}$$

Variational AutoEncoder



encoder

Latent distributions
Smile:
Skin tone:
Gender:
Beard:
Glasses:
Hair color:

sample → Sampled latent attributes: Smile: 0.23, Skin tone: 0.02, Gender: -0.18, Beard: 0.71, Glasses: -0.19, Hair color: 0.33 → decoder

sample → Sampled latent attributes: Smile: 0.17, Skin tone: 0.28, Gender: -0.11, Beard: 0.66, Glasses: -0.14, Hair color: 0.26 → decoder

Latent distributions

Sampled latent attributes

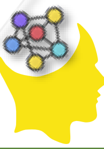
We expect an accurate reconstruction for any sample from the latent state distributions

Training Stage

$$\underbrace{\mathbb{E}_{q_\phi}[\log p_\theta(x|z)]}_{\text{reconstruction}} - \underbrace{D_{\mathrm{KL}}(q_\phi(z|x) \,\|\, p(z))}_{\text{regularise toward prior}}$$

$$\log p_\theta(x \mid z) = \underbrace{-\frac{D}{2}\log(2\pi\sigma^2)}_{\text{constant w.r.t. } \theta,\phi} - \frac{1}{2\sigma^2}\|x - \hat{x}\|^2 \longrightarrow \max_{\theta,\phi} \mathbb{E}_{q_\phi}[\log p_\theta(x|z)] \equiv \min_{\theta,\phi} \frac{1}{2\sigma^2}\mathbb{E}_{q_\phi}\left[\|x - \hat{x}\|^2\right] \equiv \min_{\theta,\phi} \|x - \hat{x}\|^2$$

Variational AutoEncoder



encoder

Smile: ... Skin tone: ... Gender: ... Beard: ... Glasses: ... Hair color:

Latent distributions

sample → Smile: 0.23, Skin tone: 0.02, Gender: -0.18, Beard: 0.71, Glasses: -0.19, Hair color: 0.33 → decoder

sample → Smile: 0.17, Skin tone: 0.28, Gender: -0.11, Beard: 0.66, Glasses: -0.14, Hair color: 0.26 → decoder

Sampled latent attributes

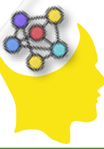
We expect an accurate reconstruction for any sample from the latent state distributions

Training Stage

$$\underbrace{\mathbb{E}_{q_\phi}[\log p_\theta(x|z)]}_{\text{reconstruction}} - \underbrace{D_{\mathrm{KL}}(q_\phi(z|x) \,\|\, p(z))}_{\text{regularise toward prior}}$$

$$\mathcal{L}_{\mathrm{KL}} = D_{\mathrm{KL}}(q_\phi(z|x) \,\|\, \mathcal{N}(0, I)) \longrightarrow \mathcal{L}_{\mathrm{KL}} = -\frac{1}{2}\sum_{j=1}^{d}(1 + \log\sigma_j^2 - \mu_j^2 - \sigma_j^2)$$

Variational AutoEncoder



encoder

Smile: -1 0 1
Skin tone: -1 0 1
Gender: -1 0 1
Beard: -1 0 1
Glasses: -1 0 1
Hair color: -1 0 1

Latent distributions

sample

Smile: 0.23
Skin tone: 0.02
Gender: -0.18
Beard: 0.71
Glasses: -0.19
Hair color: 0.33

sample

Smile: 0.17
Skin tone: 0.28
Gender: -0.11
Beard: 0.66
Glasses: -0.14
Hair color: 0.26

Sampled latent attributes

decoder

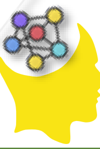
decoder

We expect an accurate reconstruction for any sample from the latent state distributions

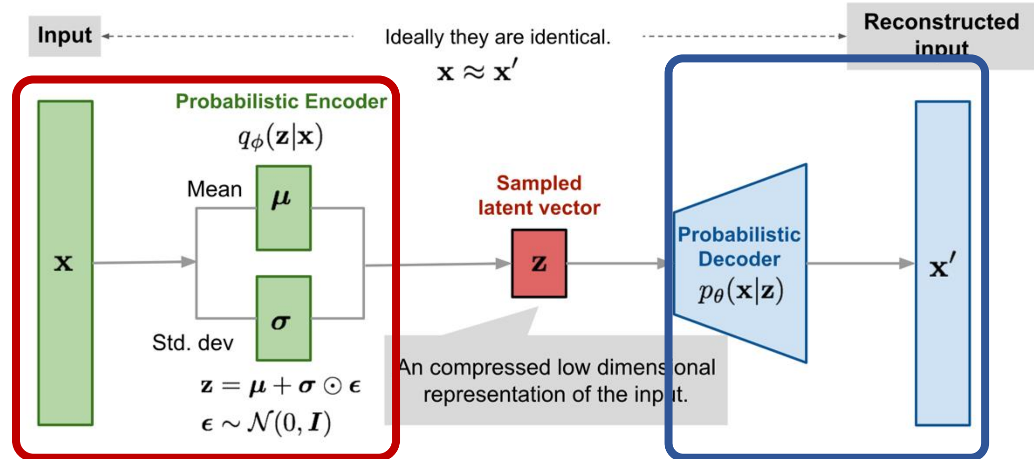
Training Stage

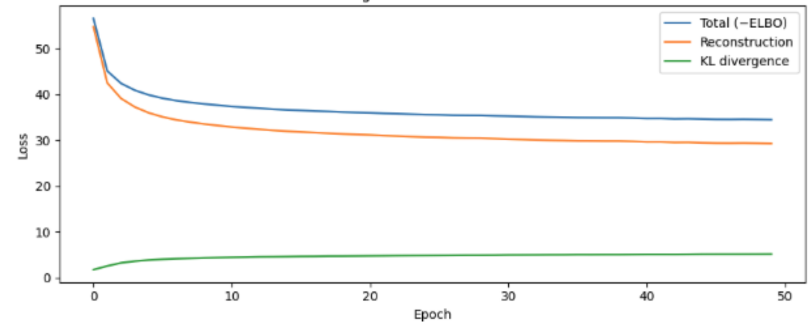
$$\mathcal{L}_{\text{VAE}} = \underbrace{\|x - \hat{x}\|^2}_{\text{Gaussian decoder} \Rightarrow \text{MSE}} + \underbrace{D_{\text{KL}}(q_\phi(z|x) \,\|\, \mathcal{N}(0, I))}_{\text{KL regulariser}}$$

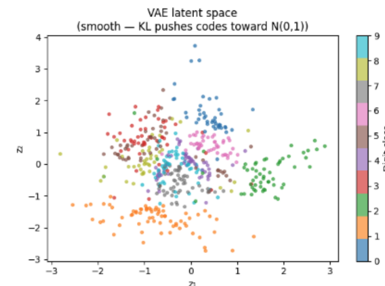
Variational AutoEncoder

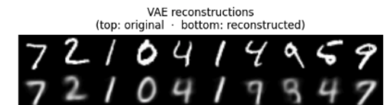


```python
# — VAE (uses train_dl, device, Decoder from Part 2) ————
class VAEEncoder(nn.Module):
    def __init__(self, latent_dim):
        super().__init__()
        self.shared    = nn.Sequential(
            nn.Flatten(),
            nn.Linear(784, 256), nn.ReLU(),
            nn.Linear(256, 64),  nn.ReLU(),
        )
        self.fc_mu     = nn.Linear(64, latent_dim)
        self.fc_logvar = nn.Linear(64, latent_dim)
    def forward(self, x):
        h = self.shared(x)
        return self.fc_mu(h), self.fc_logvar(h)

class VAE(nn.Module):
    def __init__(self, latent_dim=2):
        super().__init__()
        self.encoder = VAEEncoder(latent_dim)
        self.decoder = Decoder(latent_dim)    # reuse from Part 2

    def reparameterise(self, mu, logvar):
        sigma = torch.exp(0.5 * logvar)       # σ = exp(log σ)
        eps   = torch.randn_like(sigma)       # ε ~ N(0,I), outside graph
        return mu + sigma * eps

    def forward(self, x):
        mu, logvar = self.encoder(x)
        z          = self.reparameterise(mu, logvar)
        return self.decoder(z), mu, logvar

def elbo_loss(x, x_hat, mu, logvar):
    recon = ((x - x_hat)**2).sum(dim=[1,2,3]).mean()
    kl    = -0.5 * (1 + logvar - mu**2 - logvar.exp()).sum(dim=1).mean()
    return recon + kl, recon, kl
```

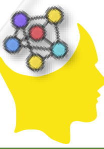


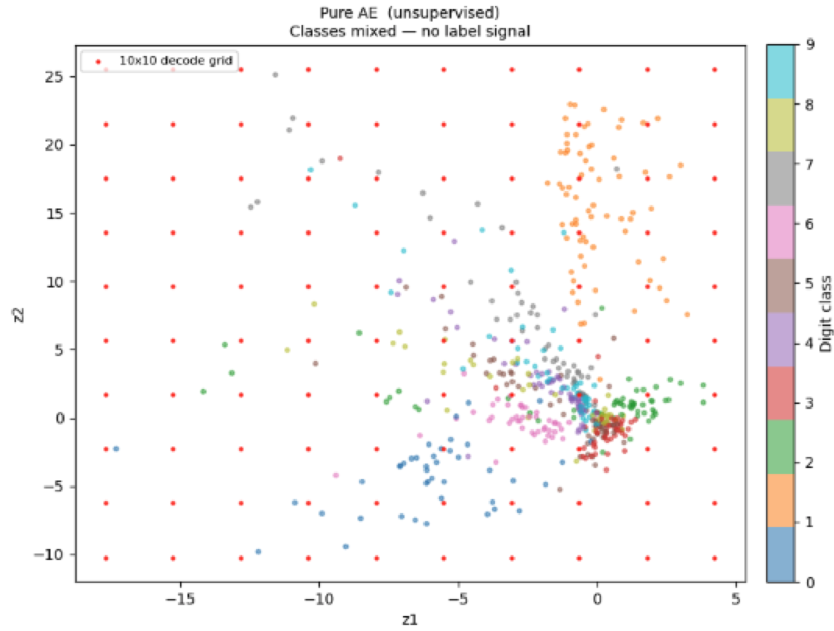


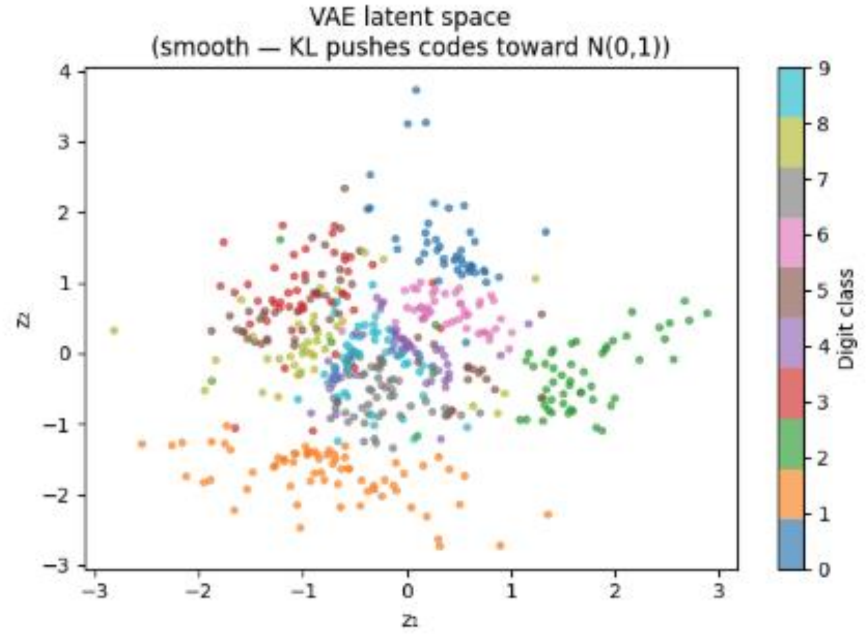


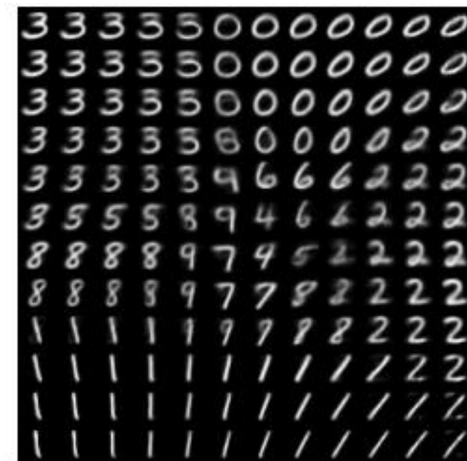


Variational AutoEncoder vs AE









QA