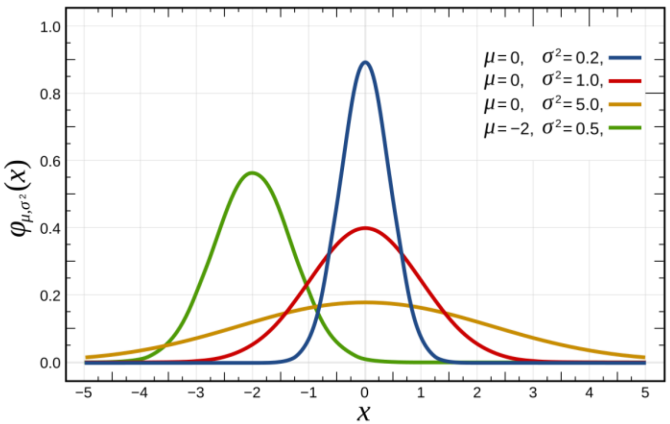
Advanced Machine Learning
Generative Model

Yu Wang
Assistant Professor
Department of Computer Science
University of Oregon

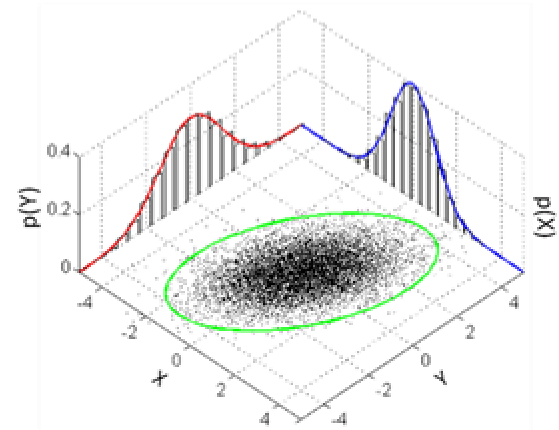
Summary

1D Gaussian Distribution

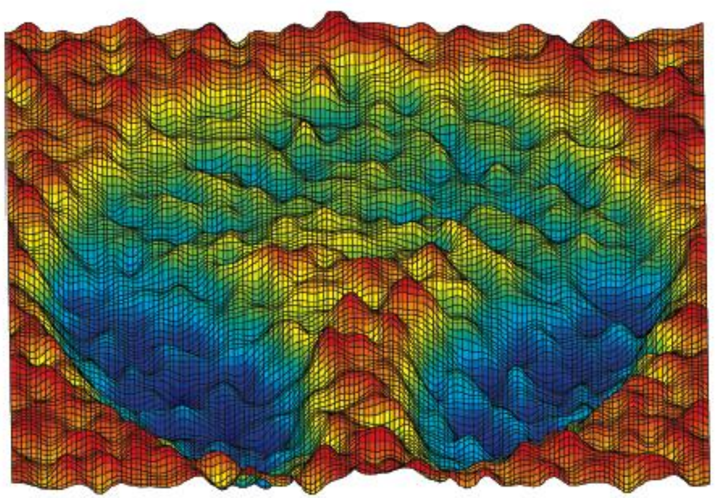


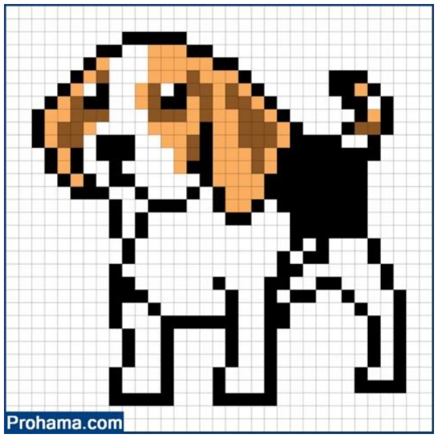
$\mathbb{R}$

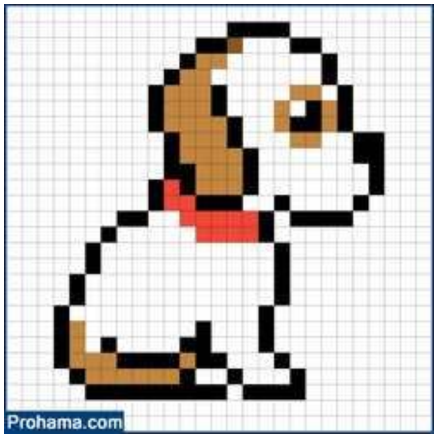
2D Gaussian Distribution



$\mathbb{R}^2$







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

Summary

Probability distribution of the **objective** based on the **observed data**

$$\{x_i\}_{i=1}^{N} \xrightarrow{\text{Good Model}} P(x) \xrightarrow{\text{Good Data}} x$$

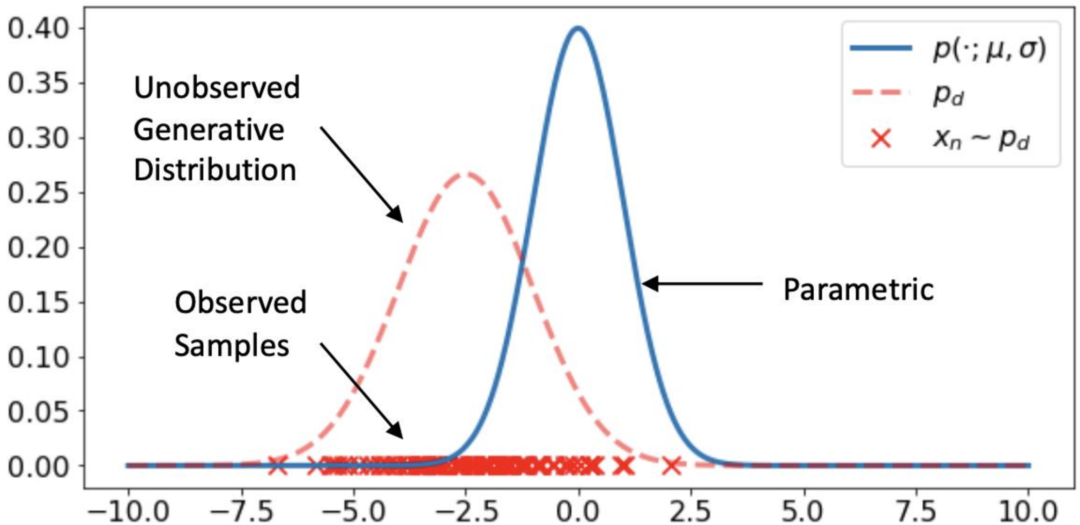
- **Machine Learning Methods**
  - Gaussian Kernel Density Estimation
  - Gaussian Mixture Models

Using existing function to estimate what you do not know that can best fit your observation

- **Deep Learning Methods**
  - Auto-Encoder (AE)
  - Variational AE (LLM is actually a VAE)
  - Generative Adversarial Network
  - Diffusion Model

Using learnable function to estimate what you do not know that can best fit your observation

Summary – Gaussian Kernel Density Estimation



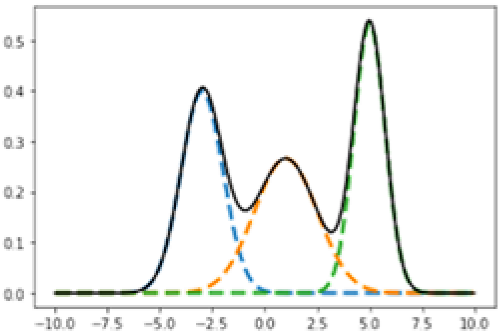
$$\{x_i | x_i \sim P_d\}_{i=1}^{N}$$

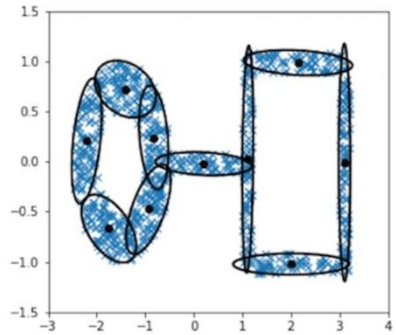
3- Minimizing Negative Log-Likelihood:

$$\underbrace{\underset{\mu,\sigma}{\operatorname{argmin}} \sum_{n=1}^{N} \frac{\log(2\pi\sigma^2)}{2} + \frac{(x_n-\mu)^2}{2\sigma^2}}_{\text{L}} \longrightarrow \begin{cases} \dfrac{\partial L}{\partial \mu} = 0 \Rightarrow \mu_* = \dfrac{1}{N}\displaystyle\sum_{n=1}^{N} x_n \\ \dfrac{\partial L}{\partial \sigma} = 0 \Rightarrow \sigma_*^2 = \dfrac{1}{N}\displaystyle\sum_{n=1}^{N} (x_n-\mu^*)^2 \end{cases}$$

Code-Demo

Summary – Gaussian Mixture Models





- **E-step: Estimate the Latent Variable given the current Gaussian Parameter**

$$\alpha_1 = \sum_{i=1}^{N} \frac{r_i^1}{\lambda} = \sum_{i=1}^{N} \frac{r_i^1}{N} \qquad \alpha_2 = \sum_{i=1}^{N} \frac{r_i^2}{\lambda} = \sum_{i=1}^{N} \frac{r_i^2}{N} \qquad p(z_k|x_i) = \frac{p(x_i|z_k)p(z_k)}{p(x_i)} = \frac{p(x_i|z_k)p(z_k)}{\sum_{k=1}^{3} p(x_i|z_k)p(z_k)} = r_i^k$$

$$\alpha_3 = \sum_{i=1}^{N} \frac{r_i^3}{\lambda} = \sum_{i=1}^{N} \frac{r_i^3}{N}$$

- **M**aximization Step: for fixed r_n^k solve the maximum log-likelihood to obtain optimal parameters:

  - Means: $\mu_k = \frac{1}{N_k} \sum_n r_n^k x_n$

  - Variances: $\sigma_k^2 = \frac{1}{N_k} \sum_n r_n^k (x_n - \mu_k)^2$

Code-Demo

Summary – Gaussian Mixture Models

- **Both of these Gaussian Kernel Density Estimation and GMM are solved theoretically**

- **We can also do it computationally using gradient descent**

$$p(x; [(\alpha_k, \mu_k, \sigma_k)]_{k=1}^{K}) = \sum_{k=1}^{K} \alpha_k N(x; \mu_k, \sigma_k) = \sum_{k=1}^{K} \frac{\alpha_k}{\sqrt{2\pi\sigma_k^2}} e^{-\frac{(x-\mu_k)^2}{2\sigma_k^2}}$$

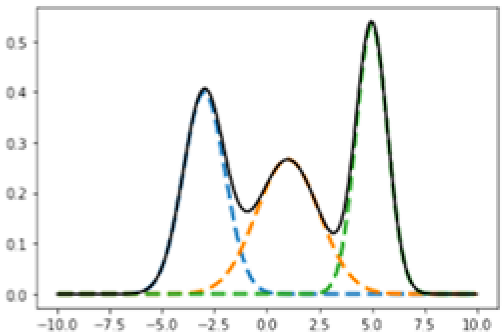
Where $\alpha_k \geq 0$, $\sum_{k=1}^{K} \alpha_k = 1$, and $\sigma_k \geq 0$.

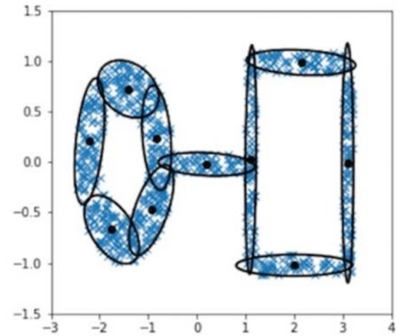
```
var = softplus(log_var) + 1e-6  # ensure numerical stability
pi = torch.softmax(logits, dim=0)

# Compute log-likelihood
data_exp = data.unsqueeze(1)  # (N, 1, D)
mu_exp = mu.unsqueeze(0)      # (1, K, D)
var_exp = var.unsqueeze(0)    # (1, K, D)

diff = data_exp - mu_exp                    # (N, K, D)
log_prob = -0.5 * torch.sum((diff ** 2) / var_exp + torch.log(2 * math.pi * var_exp), dim=2)  # (N, K)
weighted_log_prob = log_prob + torch.log(pi)  # (N, K)
log_sum_exp = torch.logsumexp(weighted_log_prob, dim=1)  # (N,)
neg_log_likelihood = -log_sum_exp.mean()

# Backward and step
neg_log_likelihood.backward()
optimizer.step()
```

Code-Demo

Problem?

Using existing function to estimate what you do not know that can best fit your observation





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

$\downarrow$

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

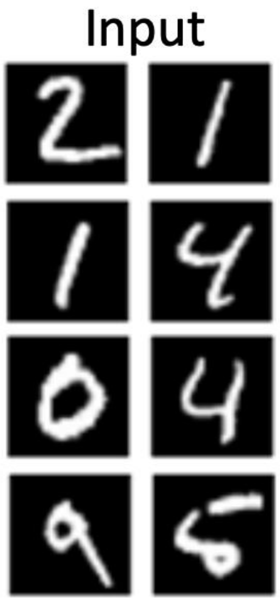
What you have is some low-dimensional data
But what you want to model is some high-dimensional data, how it could be?

Problem?

What we want: model any data distribution

How to transform any data distribution to low dimensional data?

What we have: kernel density estimation to estimate low dimensional PDF

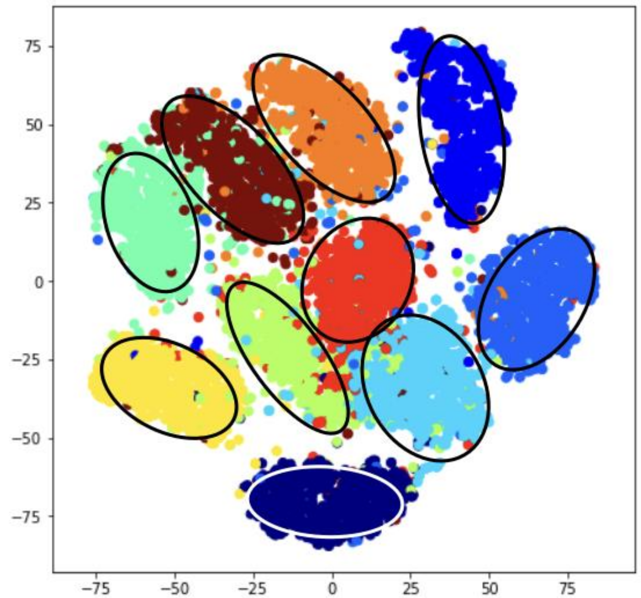


Someway to transform

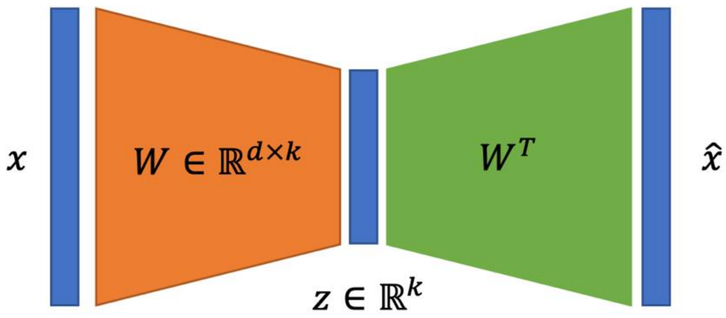
PCA

Transform back

Kernel Density Estimation



From PCA to Auto-Encoder



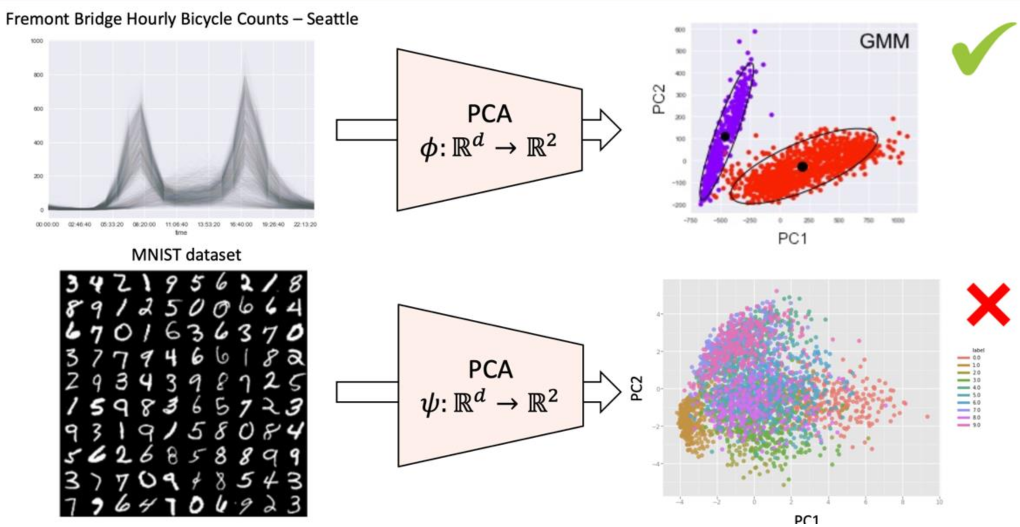
PCA:

- 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

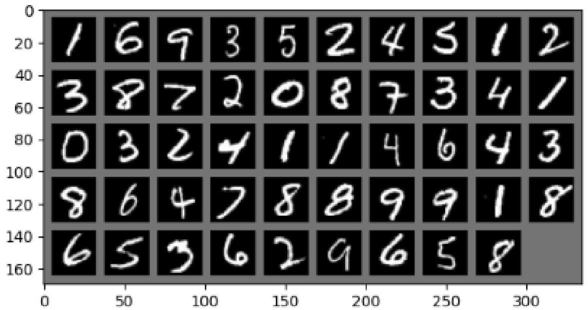


Can we add nonlinearity?

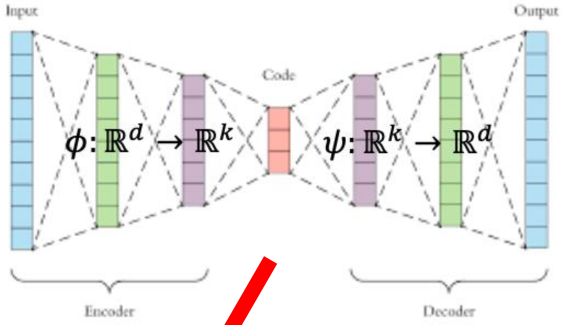
Yes, then it becomes neural network!

Auto-Encoder

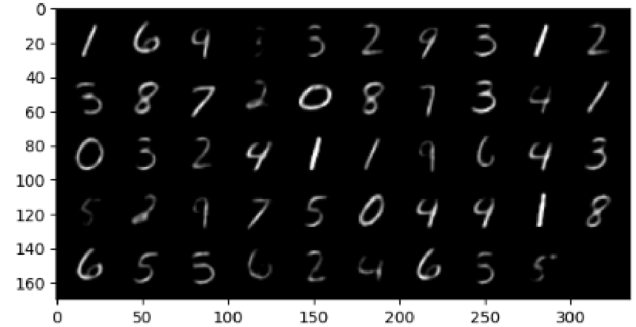
Input

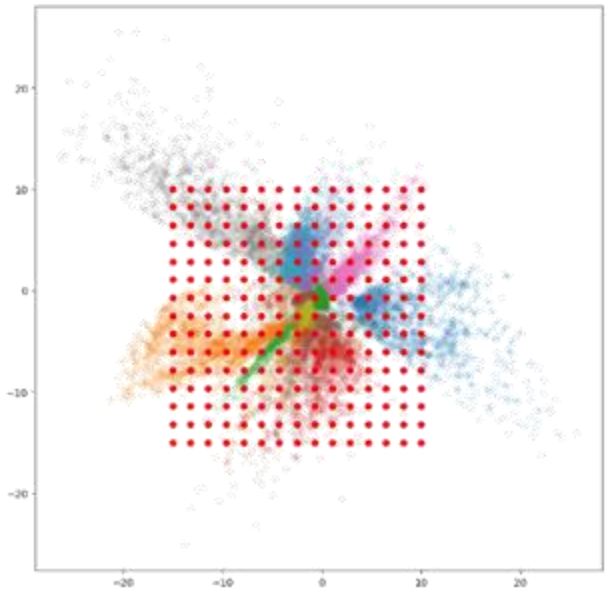


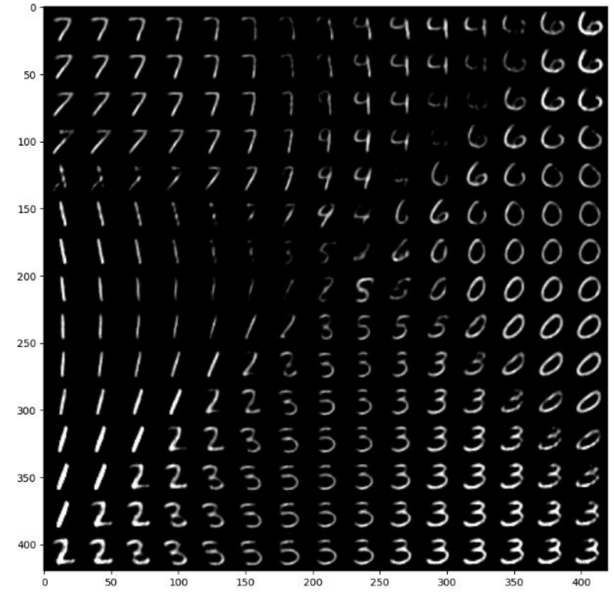
AE



Output

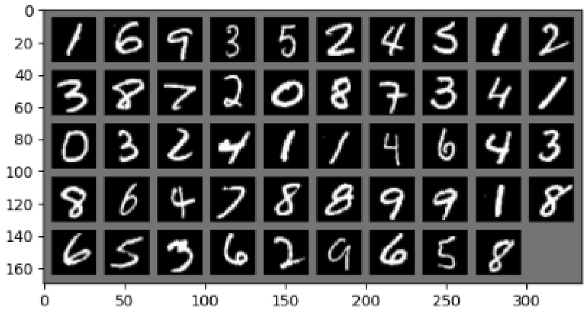




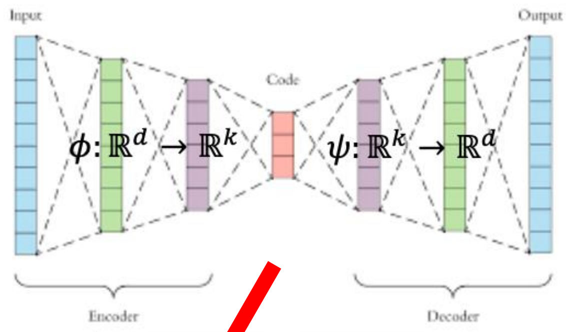


Class-supervised Auto-Encoder

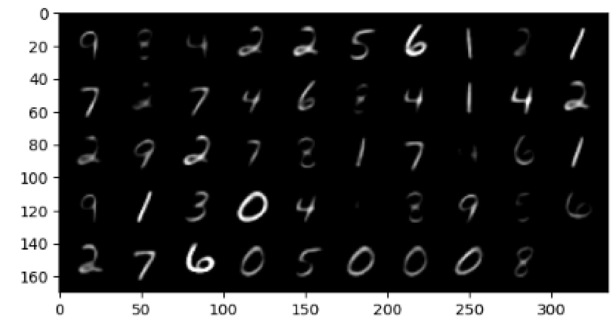
Input

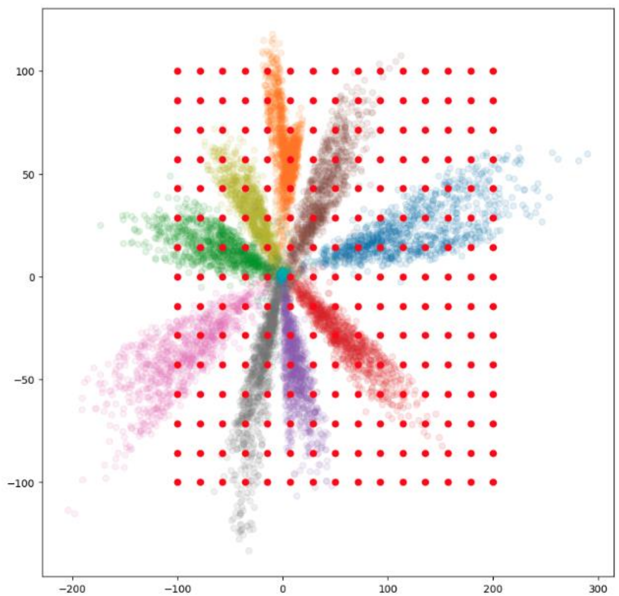


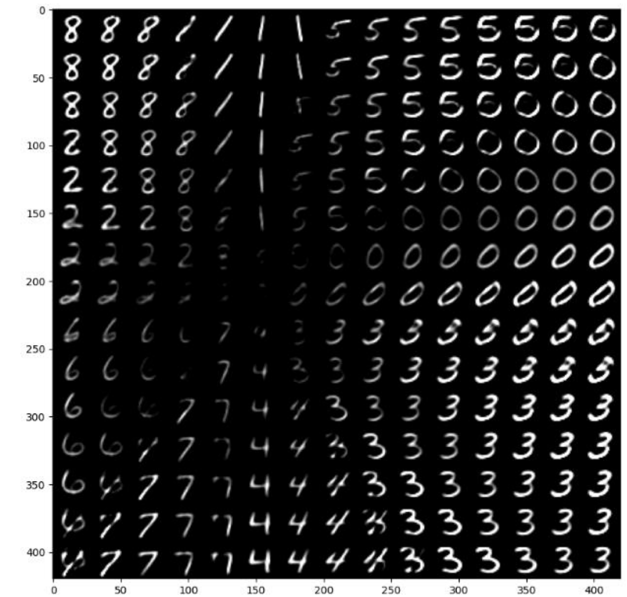
AE



Output







Auto-Encoder and MLE

🧠 **Goal of Autoencoder**

An autoencoder learns:

- An encoder: $z = f_\phi(x)$
- A decoder: $\hat{x} = g_\theta(z) = g_\theta(f_\phi(x))$

It is trained by **minimizing the reconstruction loss**:

$$\mathcal{L}_{\text{AE}} = \|x - \hat{x}\|^2 = \|x - g_\theta(f_\phi(x))\|^2$$

🎯 **MLE View: Reconstruction as Likelihood**

Assume that the decoder defines a conditional probability distribution:

$$p_\theta(x|z) = \mathcal{N}(x; g_\theta(z), \sigma^2 I)$$

This says: given a latent z, the output x is normally distributed around the decoder output $g_\theta(z)$ with variance σ^2.

Then the **log-likelihood** of x under this model is:

$$\log p_\theta(x|z) = -\frac{1}{2\sigma^2}\|x - g_\theta(z)\|^2 + \text{const}$$

So, maximizing the log-likelihood is equivalent to **minimizing**:

$$\mathcal{L}_{\text{MLE}} = \|x - g_\theta(f_\phi(x))\|^2$$

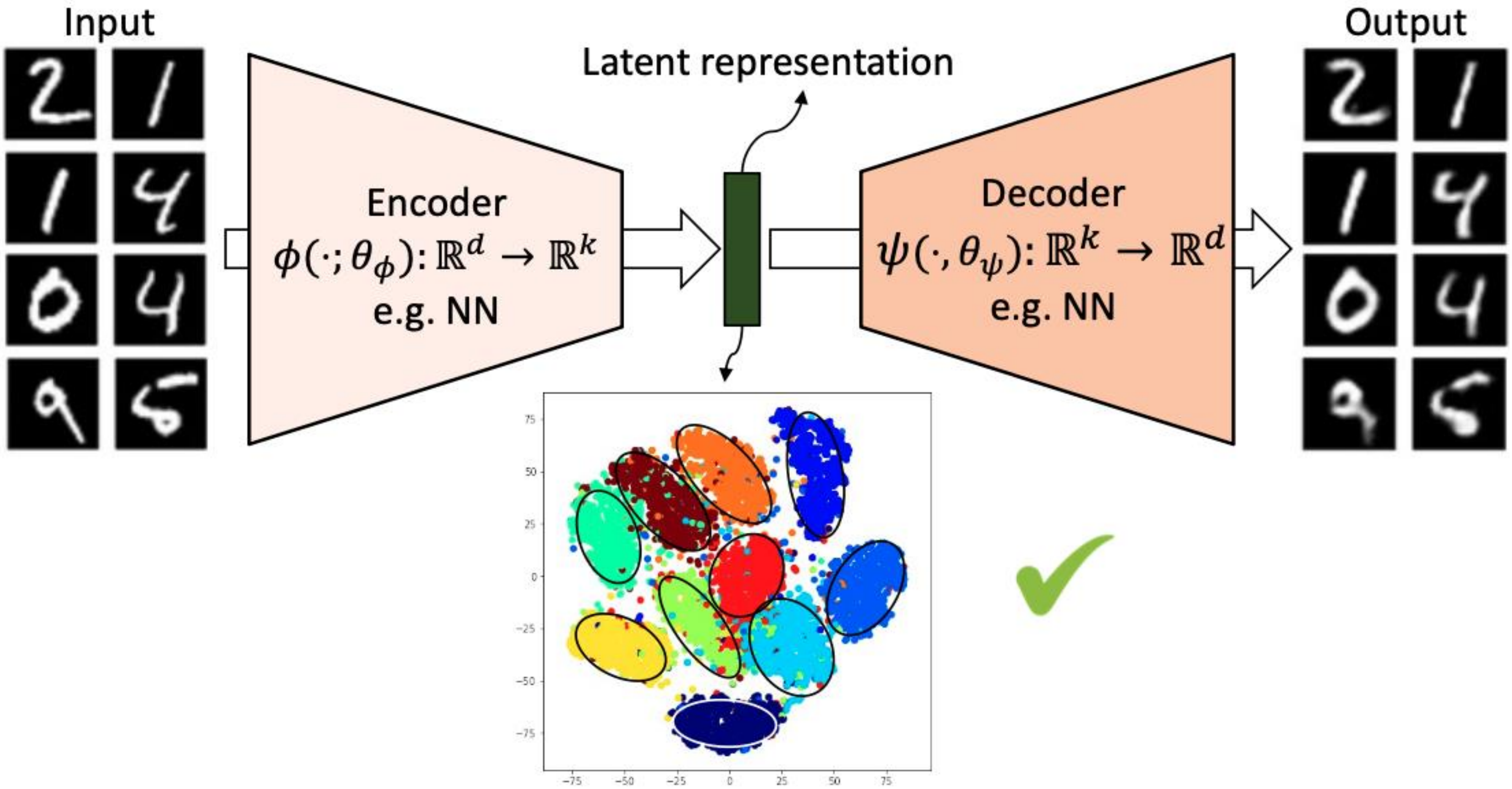
which is **exactly** the autoencoder's reconstruction loss (up to a constant factor).

Credited by ChatGPT

But theoretically the decoder of AE is fixed Dirac Delta Function

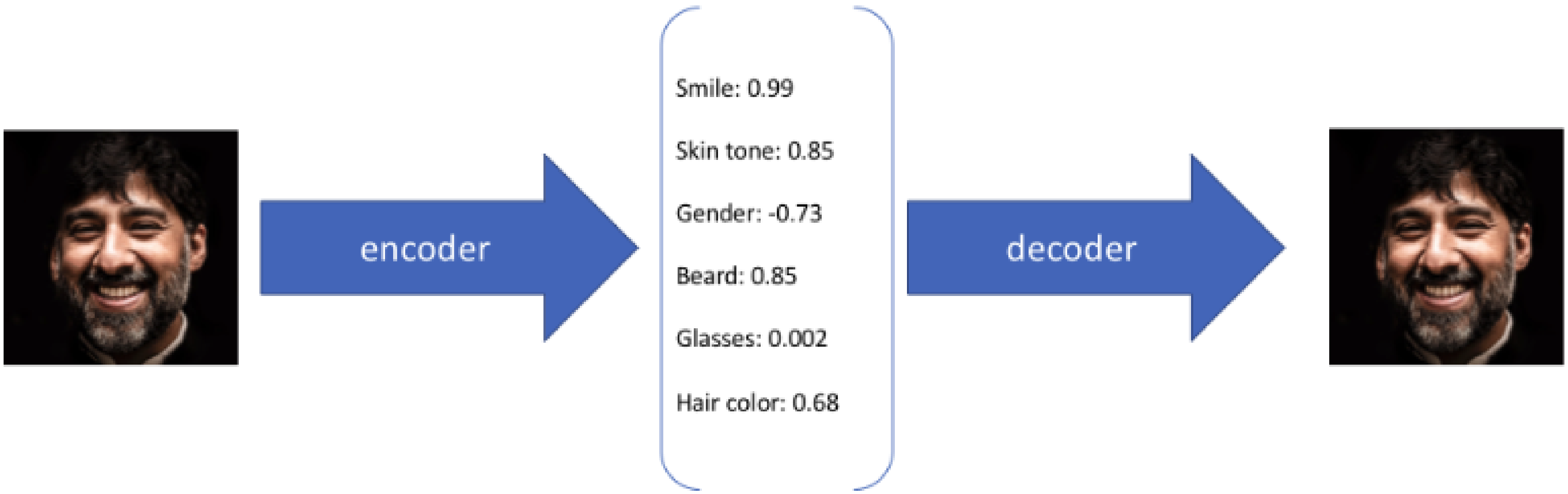
But we can assume there is some noise such as randomness (dropout) in neural network

Problem with AE



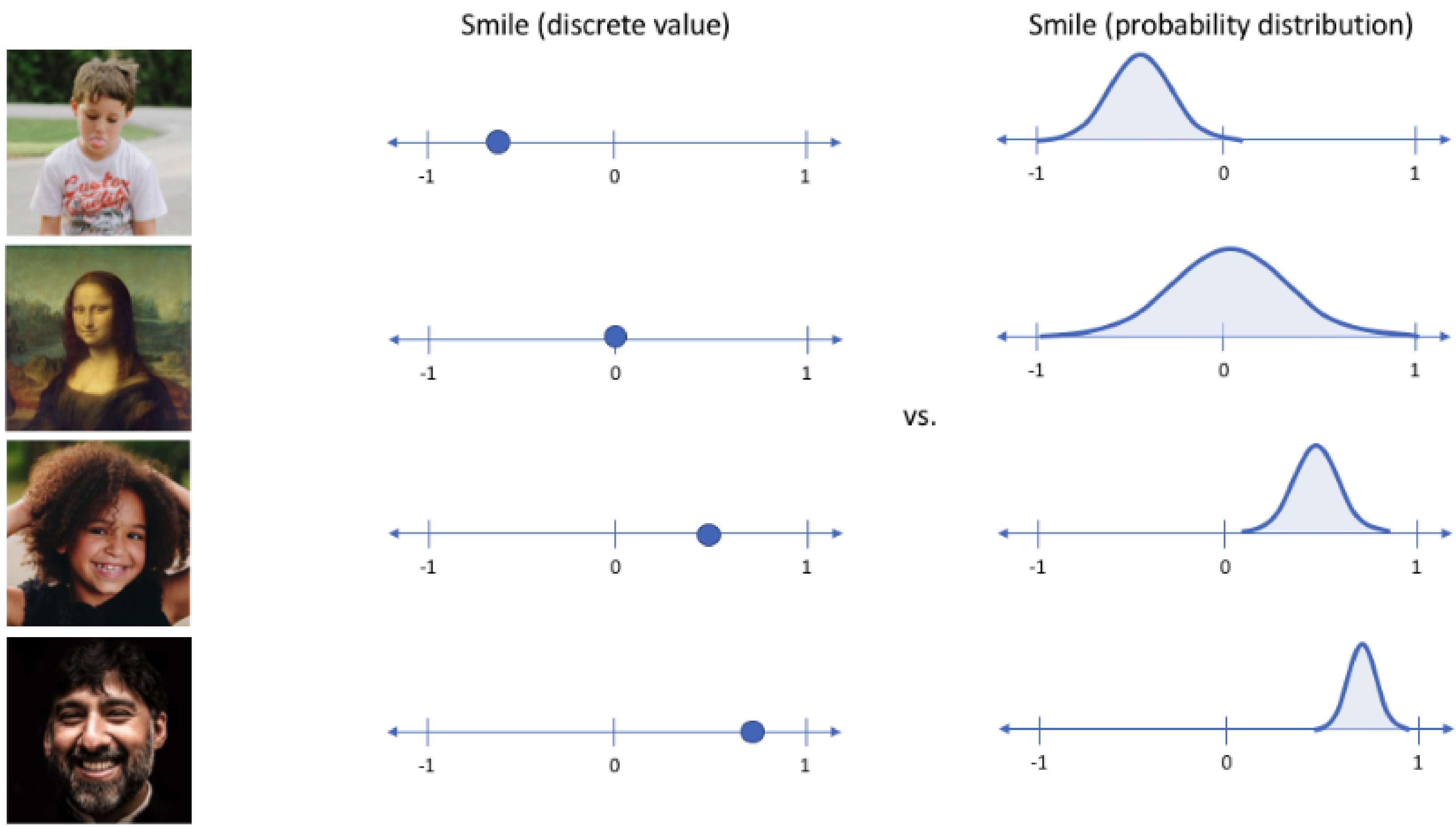
Need to estimate the latent distribution post-hoc!

Problem with AE

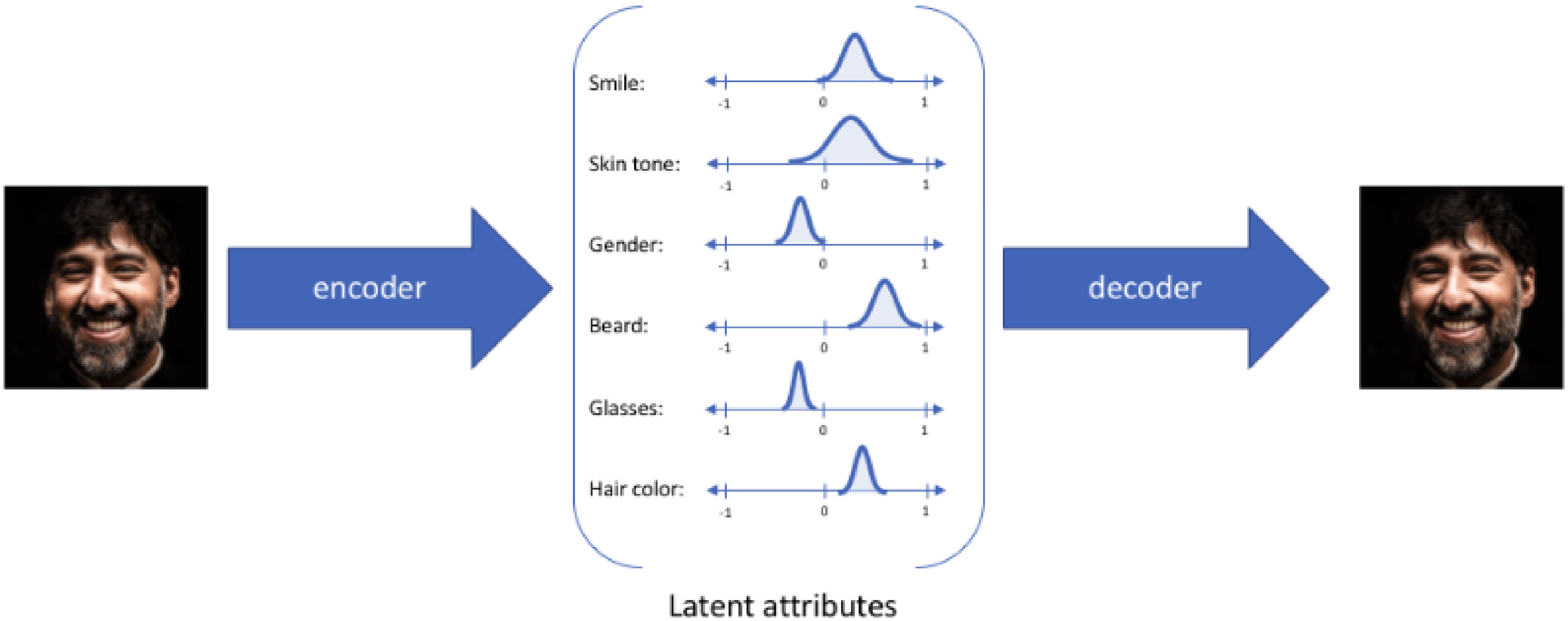


Latent attributes

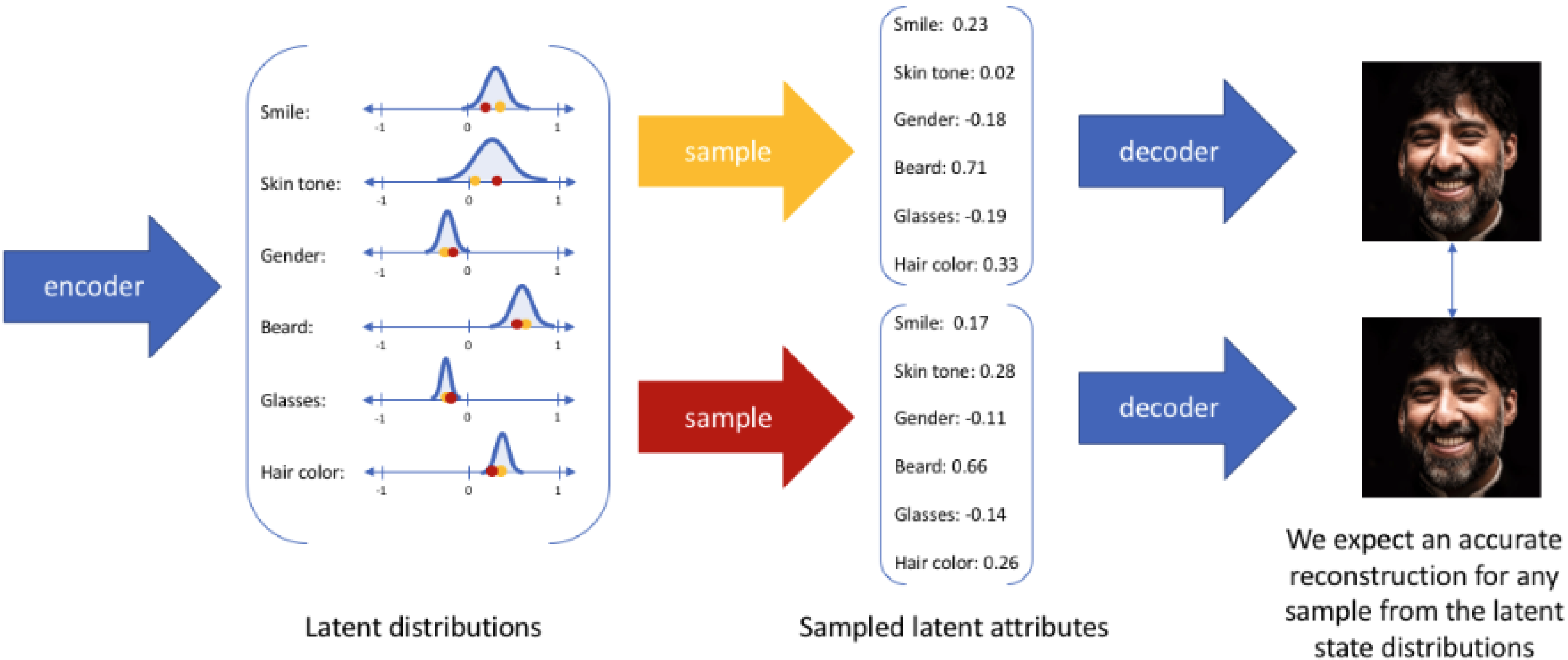
Problem with AE



Problem with AE



Problem with AE



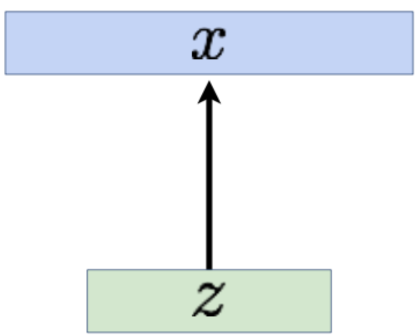
Idea – Latent Variable

Probabilistic spin on autoencoder - will let us sample from the model to generate

Assume training data is generated from underlying unobserved (latent) representation z

Sample from true **conditional**

$p_{\theta^*}(x \mid z^{(i)})$



Sample from true **prior**

$p_{\theta^*}(z)$

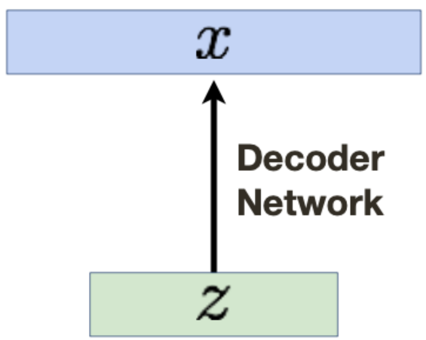
Intuition: $\boldsymbol{x}$ is an image, $\boldsymbol{z}$ is latent factors used to generate $\boldsymbol{x}$ (e.g., attributes, orientation, *etc.*)

Idea – Latent Variable

We want to **estimate the true parameters** θ^* of this generative model

Sample from
true **conditional**

$$p_{\theta^*}(x \mid z^{(i)})$$



Sample from
true **prior**

$$p_{\theta^*}(z)$$

How do we **represent** this model?

Choose prior $p(z)$ to be simple, e.g., Gaussian
Reasonable for latent attributes, e.g., pose, amount of smile

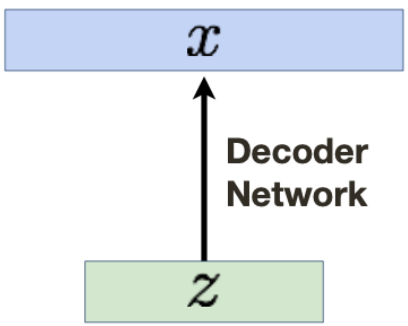
Conditional $p(x|z)$ is complex (generates image)
Represent with Neural Network

Idea – Latent Variable

We want to **estimate the true parameters** θ^* of this generative model

Sample from true **conditional**

$p_{\theta^*}(x \mid z^{(i)})$



Sample from true **prior**

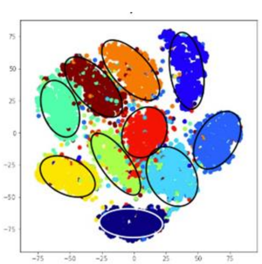
$p_{\theta^*}(z)$

How do we **train** this model?

Remember the strategy from earlier — learn model parameters to maximize likelihood of training data

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

(now with latent z that we need to marginalize)



$p_\theta(z|x)$

Intractable!

Which x corresponding to which z?

$q_\phi(z|x) \longrightarrow p_\theta(z|x)$

Idea – Latent Variable

Now equipped with **encoder** and **decoder** networks, let's see (log) data likelihood:

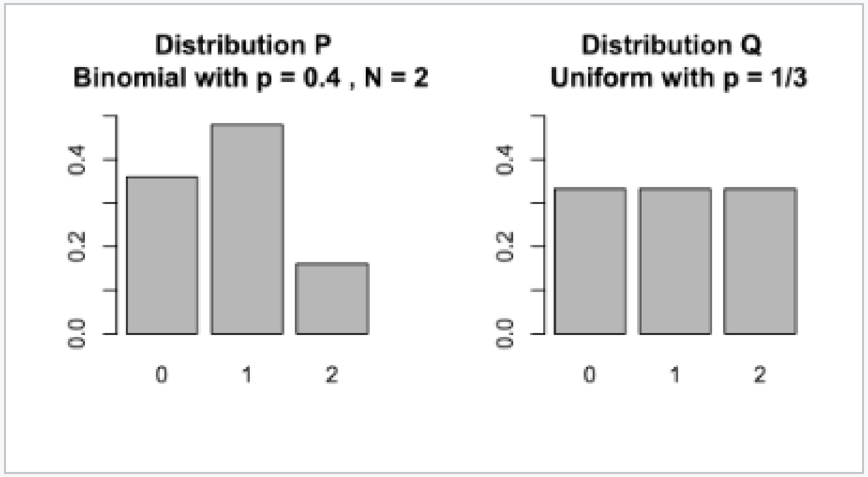
$$
\begin{aligned}
\log p_\theta(x^{(i)}) &= \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \qquad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \qquad (\text{Bayes' Rule}) \\
&= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})} \frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \qquad (\text{Multiply by constant}) \\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z)}\right] + \mathbf{E}_z\left[\log \frac{q_\phi(z \mid x^{(i)})}{p_\theta(z \mid x^{(i)})}\right] \qquad (\text{Logarithms}) \\
&= \mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z)) + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))
\end{aligned}
$$

Expectation with respect to z
(using encoder network) leads to nice KL terms

$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x\in\mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right).$$

[Kullback–Leibler divergence]

Idea – Latent Variable



Two distributions to illustrate relative entropy

Distribution \ x	0	1	2
$P(x)$	$\frac{9}{25}$	$\frac{12}{25}$	$\frac{4}{25}$
$Q(x)$	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$

$$\begin{aligned} D_{\text{KL}}(P \parallel Q) &= \sum_{x \in \mathcal{X}} P(x) \ln\left(\frac{P(x)}{Q(x)}\right) \\ &= \frac{9}{25} \ln\left(\frac{9/25}{1/3}\right) + \frac{12}{25} \ln\left(\frac{12/25}{1/3}\right) + \frac{4}{25} \ln\left(\frac{4/25}{1/3}\right) \\ &= \frac{1}{25}\left(32 \ln(2) + 55 \ln(3) - 50 \ln(5)\right) \approx 0.0852996, \end{aligned}$$

$$\begin{aligned} D_{\text{KL}}(Q \parallel P) &= \sum_{x \in \mathcal{X}} Q(x) \ln\left(\frac{Q(x)}{P(x)}\right) \\ &= \frac{1}{3} \ln\left(\frac{1/3}{9/25}\right) + \frac{1}{3} \ln\left(\frac{1/3}{12/25}\right) + \frac{1}{3} \ln\left(\frac{1/3}{4/25}\right) \\ &= \frac{1}{3}\left(-4 \ln(2) - 6 \ln(3) + 6 \ln(5)\right) \approx 0.097455. \end{aligned}$$

KL divergence is bigger than 0

Idea – Latent Variable

Now equipped with **encoder** and **decoder** networks, let's see (log) data likelihood:

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z\sim q_\phi(z|x^{(i)})}\left[\log p_\theta(x^{(i)})\right] \qquad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\right] \qquad \text{(Bayes' Rule)}$$

$$= \mathbf{E}_z\left[\log \frac{p_\theta(x^{(i)} \mid z)p_\theta(z)}{p_\theta(z \mid x^{(i)})}\frac{q_\phi(z \mid x^{(i)})}{q_\phi(z \mid x^{(i)})}\right] \qquad \text{(Multiply by constant)}$$

Reconstruct Input Data

Make approximate posterior close to the prior

$$= \underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)} + D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z \mid x^{(i)}))$$

$$\log p_\theta(x^{(i)}) \geq \mathcal{L}(x^{(i)},\theta,\phi)$$

Variational lower bound ("**ELBO**")

Training: Maximize lower bound

$$\theta^*,\phi^* = \arg\max_{\theta,\phi}\sum_{i=1}^{N}\mathcal{L}(x^{(i)},\theta,\phi)$$

Idea – Latent Variable

$$\underbrace{\boxed{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right]} - D_{KL}(q_\phi(z \mid x^{(i)}) \,\|\, \boxed{p_\theta(z)})}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Gaussian

(1) Reconstruction loss: given z – decoder – x and setup the reconstruction loss

(2) KL divergence: how to optimize the KL divergence between two distributions?

Idea – Latent Variable

Avoid Post-hoc Density Estimation

- Variational Auto-Encoders (Very Popular)
- Sliced Wasserstein Auto-Encoders (Simpler and practical, yet not as popular! ☹)

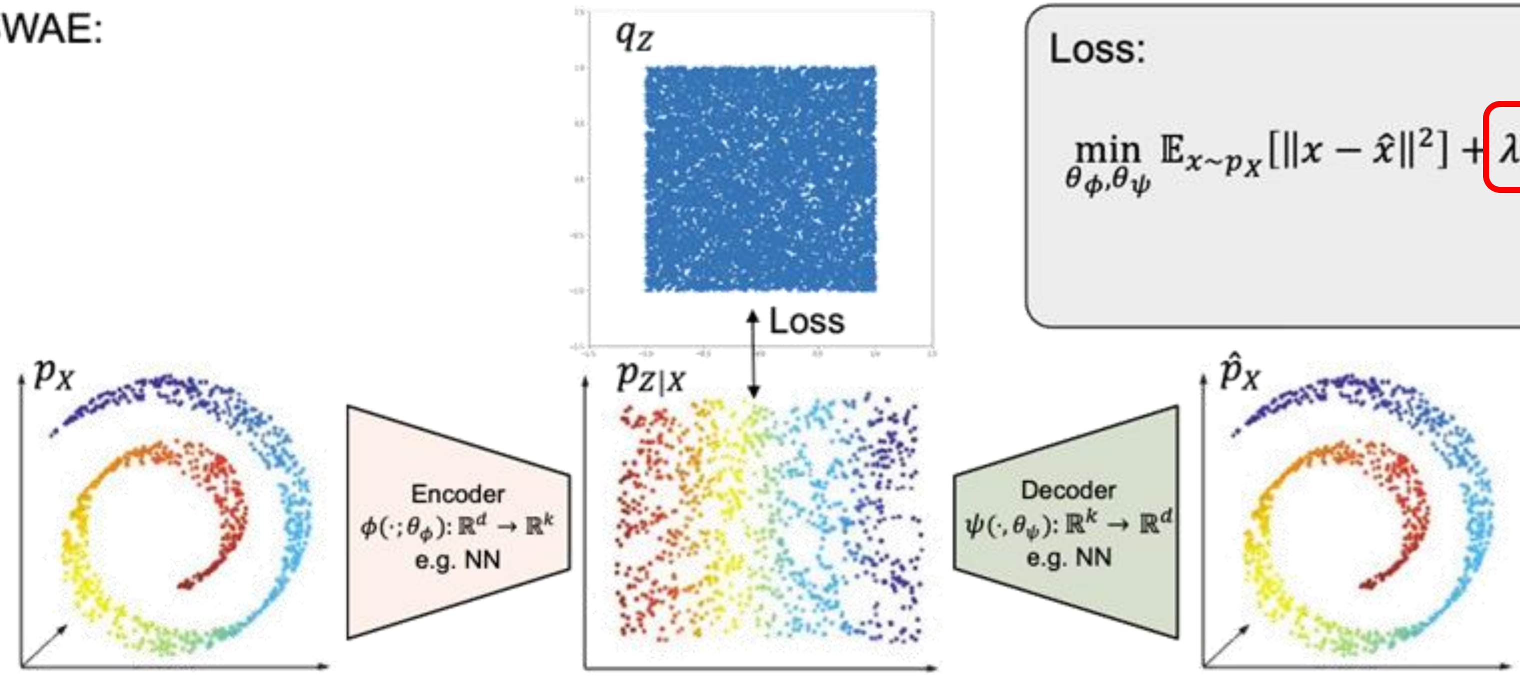
"Overall, given the conceptual and training simplicity of SWAEs, I personally found them to be a lucrative alternative to VAEs and WAEs. Completely deterministic encoder and decoders, and not requiring a discriminator during learning are two factors going in favor of SWAEs over VAEs and WAEs."

https://bigredt.github.io/2019/05/13/genae/

Solution 1 – Sliced Wasserstein AE

Sliced Wasserstein Distance between two distributions!

SWAE:



Loss:

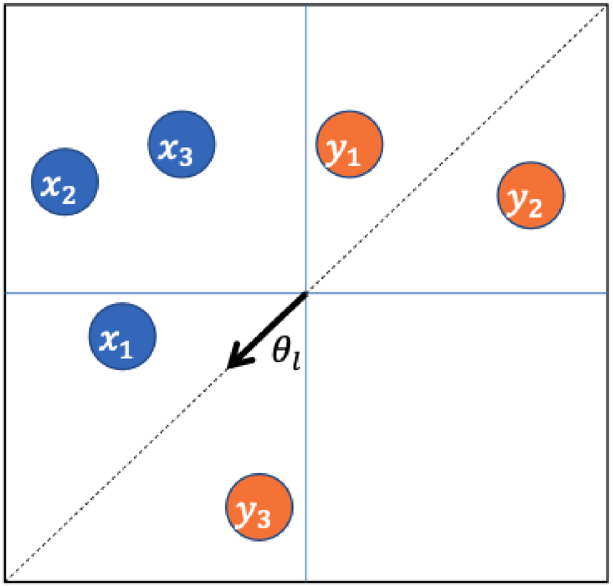
$$\min_{\theta_\phi,\theta_\psi} \mathbb{E}_{x\sim p_X}[\|x-\hat{x}\|^2] + \boxed{\lambda\, SW(p_{Z|X}, q_Z)}$$

Solution 1 – Sliced Wasserstein AE

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$

$$\min_{\theta_\phi,\theta_\psi} \mathbb{E}_{x\sim p_X}[||x-\hat{x}||^2] + \lambda\, SW\left(p_{Z|X}, q_Z\right)$$

Solution1 – Sliced Wasserstein AE



SW2=0

For l in range(L):

- Generate a random unit vector, θ_l
- Calculate $\{\theta_l^T x_n\}_{n=1}^N$ and sort them
- Calculate $\{\theta_l^T y_n\}_{n=1}^N$ and sort them
- $SW2 = SW2 + \frac{1}{L}\sum_{n=1}^N \left(\theta_l^T x_{\pi_x[n]} - \theta_l^T y_{\pi_y[n]}\right)^2$

Approximate the distance between two distributions

Solution 1 – Sliced Wasserstein AE

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$

$$\mathbb{E}_{z\sim q_\phi(z|x)}\left[\log p_\theta(x|z)\right]$$

$$p_\theta(x|z) = \mathcal{N}(x;\mu_\theta(z),\sigma^2 I) \Rightarrow \log p_\theta(x|z) = -\frac{1}{2\sigma^2}\|x-\mu_\theta(z)\|^2 + \text{const}$$

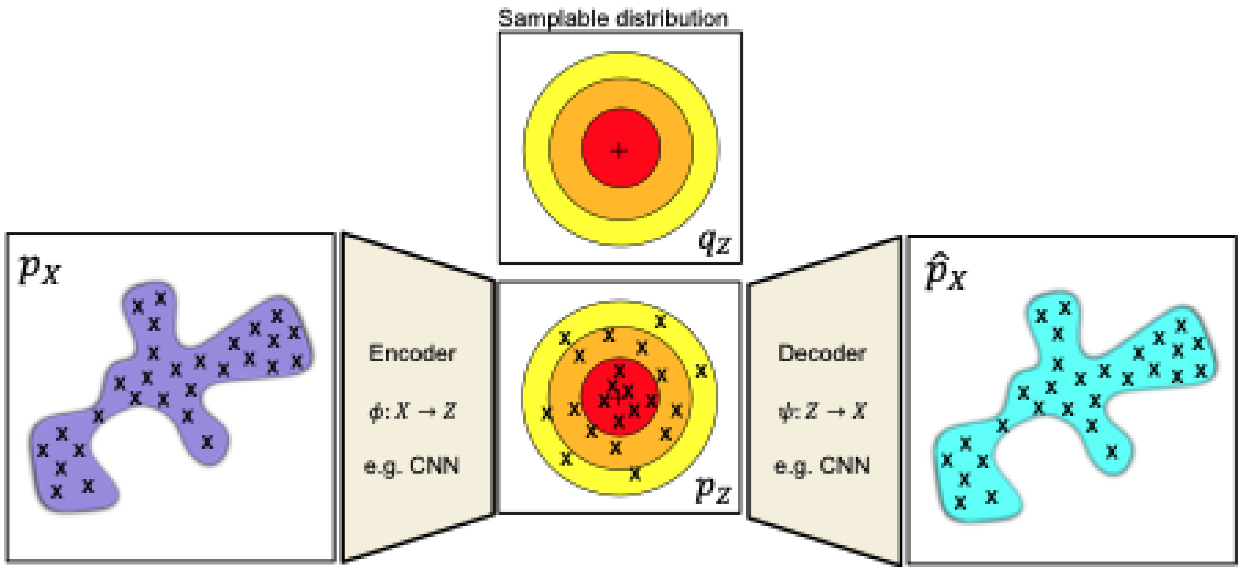
$$\log p_\theta(x|z) = -\frac{1}{2\sigma^2}\|x-\mu_\theta(z)\|^2 + \text{const} \Rightarrow \mathbb{E}_{q_\phi(z|x)}[\log p_\theta(x|z)] = \text{const} - \frac{1}{2\sigma^2}\mathbb{E}_{q_\phi(z|x)}[\|x-\mu_\theta(z)\|^2]$$

$$\min_{\theta_\phi,\theta_\psi} \mathbb{E}_{x\sim p_X}[\|x-\hat{x}\|^2] + \lambda\, SW(p_{Z|X}, q_Z)$$

Solution1 – Sliced Wasserstein AE

Algorithm 1 Sliced-Wasserstein Auto-Encoder (SWAE)

Require: Regularization coefficient λ, and number of random projections, L.
  Initialize the parameters of the encoder, ϕ, and decoder, ψ
  while ϕ and ψ have not converged **do**
    Sample $\{x_1, ..., x_M\}$ from training set (i.e. p_X)
    Sample $\{\tilde{z}_1, ..., \tilde{z}_M\}$ from q_Z
    Sample $\{\theta_1, ..., \theta_L\}$ from $\mathbb{S}^{K-1}$
    Sort $\theta_l \cdot \tilde{z}_M$ such that $\theta_l \cdot \tilde{z}_{i[m]} \leq \theta_l \cdot \tilde{z}_{i[m+1]}$
    Sort $\theta_l \cdot \phi(x_m)$ such that $\theta_l \cdot \phi(x_{j[m]}) \leq \theta_l \cdot \phi(x_{j[m+1]})$
    Update ϕ and ψ by descending: $\sum_{m=1}^{M} c(x_m, \psi(\phi(x_m))) + \lambda \sum_{l=1}^{L} \sum_{m=1}^{M} c(\theta_l \cdot \tilde{z}_{i[m]}, \theta_l \cdot \phi(x_{j[m]}))$
  end while



Solution1 – Sliced Wasserstein AE

Codebook

```python
# Define Encoder and Decoder neural network architectures
class Encoder(nn.Module):
    def __init__(self):
        super(Encoder, self).__init__()
        self.fc1 = nn.Linear(28*28, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, latent_dim)
    def forward(self, x):
        x = x.view(x.size(0), -1)   # flatten 28x28 images
        x = F.relu(self.fc1(x))
        z = self.fc2(x)
        return z

class Decoder(nn.Module):
    def __init__(self):
        super(Decoder, self).__init__()
        self.fc1 = nn.Linear(latent_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, 28*28)
    def forward(self, z):
        z = F.relu(self.fc1(z))
        x_recon = torch.sigmoid(self.fc2(z))  # output in [0,1] for BCE
        return x_recon

# Initialize models and optimizer
encoder = Encoder().to(device)
decoder = Decoder().to(device)
optimizer = torch.optim.Adam(list(encoder.parameters()) + list(decoder.parameters()), lr=learning_rate)
```

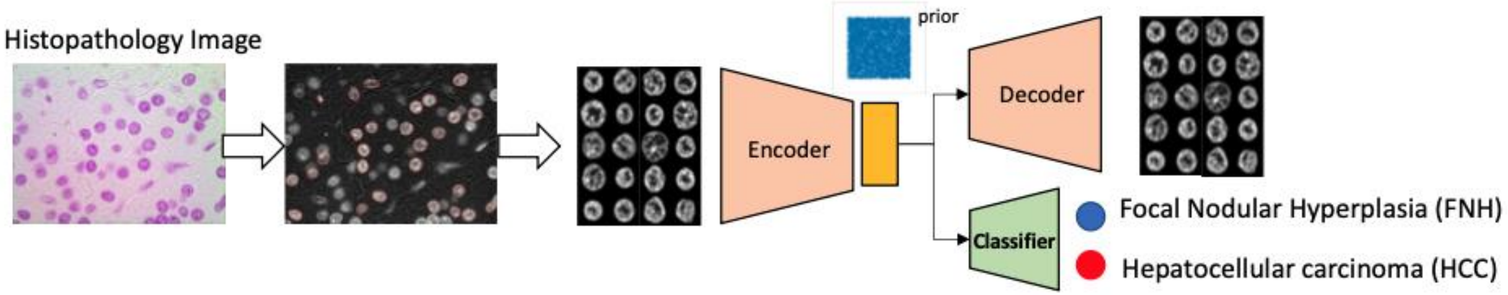
```python
        # Compute reconstruction loss (binary cross-entropy)
        recon_loss = F.binary_cross_entropy(recon_images, images.view(images.size(0), -1), reduction='sum') / images.size(0)
        # Sample from prior (standard normal) and compute sliced Wasserstein loss
        z_prior = torch.randn_like(z)
        sw_loss = sliced_wasserstein_distance(z, z_prior, n_projections=num_projections)
        # Total loss = reconstruction + lambda * SWD
        loss = recon_loss + lambda_reg * sw_loss
```

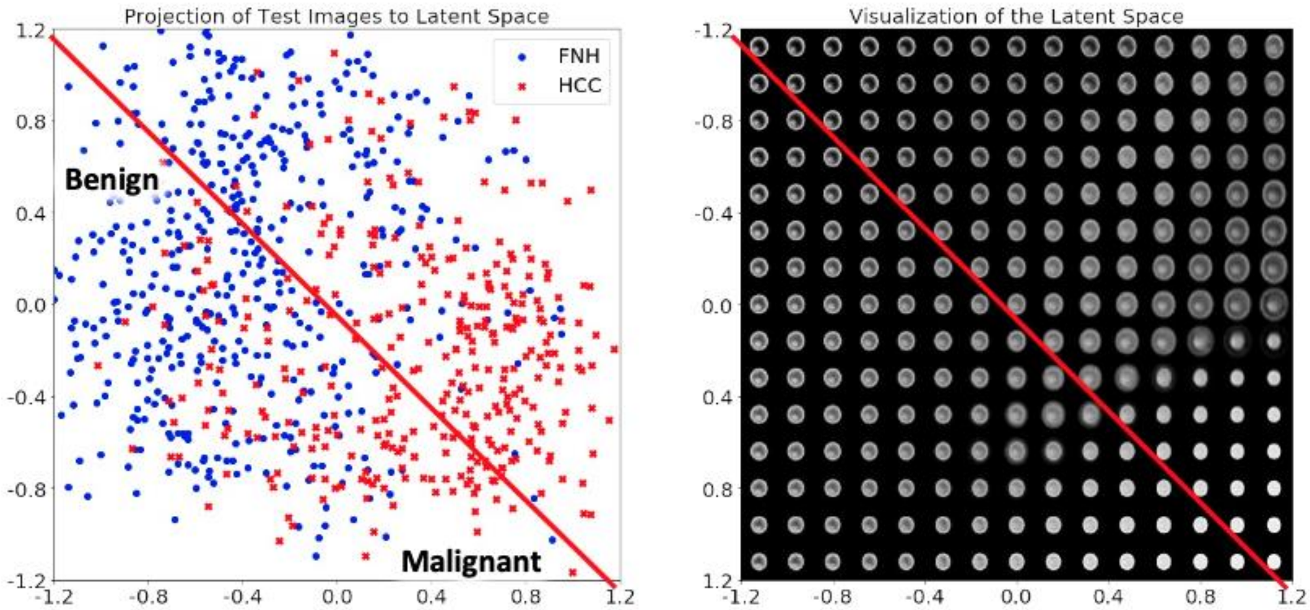
```python
# Sliced Wasserstein distance (SWD) function
def sliced_wasserstein_distance(z_real, z_prior, n_projections=50):
    # 1. Sample random projection directions from the unit sphere
    d = z_real.shape[1]
    directions = torch.randn(d, n_projections, device=z_real.device)  # random directions
    directions = directions / torch.sqrt(torch.sum(directions**2, dim=0, keepdim=True))  # normalize

    # 2. Project real and prior samples onto the random directions
    proj_real = z_real @ directions  # shape: [batch_size, n_projections]
    proj_prior = z_prior @ directions  # shape: [batch_size, n_projections]

    # 3. Sort the projections along each direction
    proj_real_sorted, _ = torch.sort(proj_real, dim=0)
    proj_prior_sorted, _ = torch.sort(proj_prior, dim=0)

    # 4. Compute the average L2 distance between sorted projections (1D Wasserstein distance for each projec
    diffs = proj_real_sorted - proj_prior_sorted
    dist = torch.sqrt(torch.mean(diffs**2, dim=0))  # L2 distance for each projection

    # 5. Return the average distance over all projections
    return torch.mean(dist)
```

Solution1 – Sliced Wasserstein AE

Sliced Wasserstein Auto-Encoder (SWAE): Semi-Supervised Learning

Sliced-Wasserstein Auto-Encoders (SWAE) for Semi Supervised Representation Learning:





Solution2 – VAE

$$\underbrace{\boxed{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right]} - D_{KL}(q_\phi(z \mid x^{(i)}) \,\|\, \boxed{p_\theta(z)})}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$

Gaussian

(1) Reconstruction loss: given z – decoder – x and setup the reconstruction loss

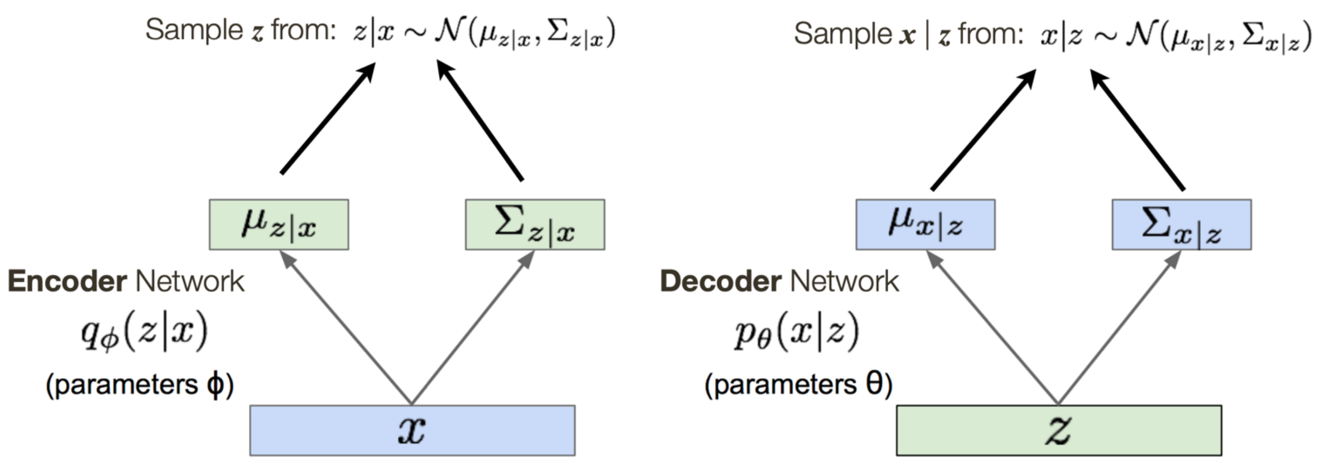
(2) KL divergence: how to optimize the KL divergence between two distributions?

Solution2 – VAE

$$\underbrace{\boxed{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right]} - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

$$\mathbb{E}_{x\sim p_X}[||x-\hat{x}||^2]$$

(1) Reconstruction loss: given z – decoder – x and setup the reconstruction loss

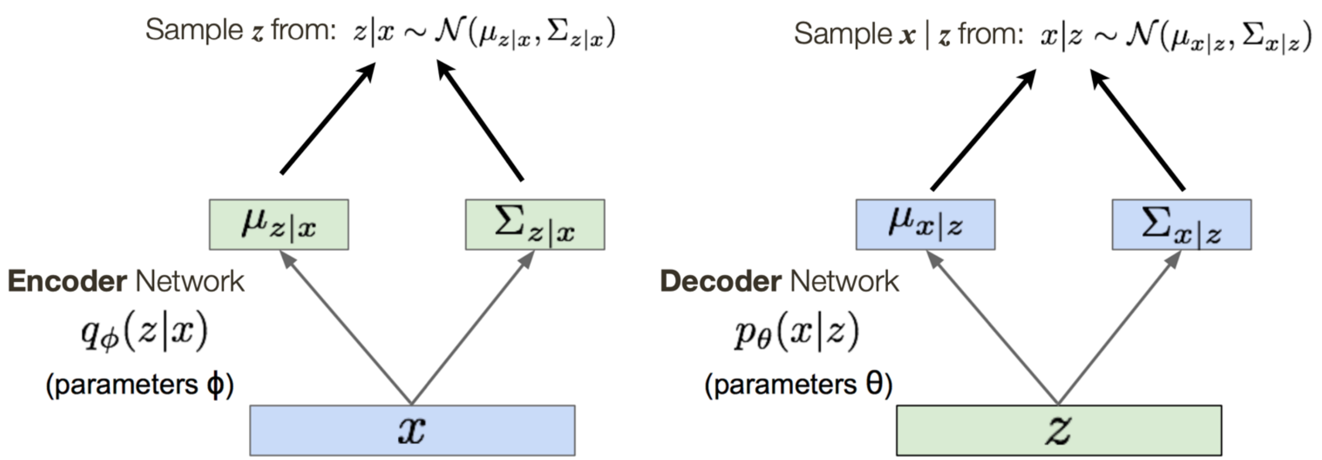


Solution2 – VAE

$$\underbrace{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right] - \boxed{D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, p_\theta(z))}}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

$$D_{\mathrm{KL}}(q_\phi(z|x) \,||\, p(z)) = \frac{1}{2}\sum_{j=1}^{d}\left[\sigma_j^2 + \mu_j^2 - 1 - \log \sigma_j^2\right]$$

(2) KL divergence: how to optimize the KL divergence between two gaussian distributions?



Codebook

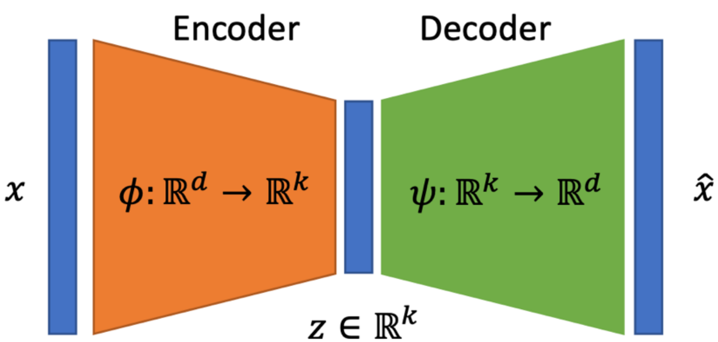
```python
# 2. Define VAE Model
class VAE(nn.Module):
    def __init__(self, latent_dim=LATENT_DIM):
        super(VAE, self).__init__()
        # Encoder layers (784 -> 400 -> latent)
        self.fc1 = nn.Linear(784, 400)
        self.fc_mu = nn.Linear(400, latent_dim)
        self.fc_logvar = nn.Linear(400, latent_dim)
        # Decoder layers (latent -> 400 -> 784)
        self.fc3 = nn.Linear(latent_dim, 400)
        self.fc4 = nn.Linear(400, 784)
    def encode(self, x):
        x = x.view(x.size(0), -1)  # flatten input
        h = torch.relu(self.fc1(x))
        mu = self.fc_mu(h)
        logvar = self.fc_logvar(h)
        return mu, logvar
    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)
        return mu + eps * std
    def decode(self, z):
        h = torch.relu(self.fc3(z))
        out = torch.sigmoid(self.fc4(h))
        return out
    def forward(self, x):
        mu, logvar = self.encode(x)
        z = self.reparameterize(mu, logvar)
        return self.decode(z), mu, logvar

model = VAE(latent_dim=LATENT_DIM).to(device)
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)
```

```python
        recon, mu, logvar = model(images)
        # Compute reconstruction and KL divergence losses
        recon_loss = F.binary_cross_entropy(recon, images.view(-1, 784), reduction='sum')
        kl_loss = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
```

Solution2 – VAE

$$\underbrace{\boxed{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right]} - D_{KL}(q_\phi(z \mid x^{(i)}) \| \boxed{p_\theta(z)})}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$

Decoder **Encoder** **Gaussian**



AE:

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

Nonlinear dimensionality Reduction

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

Solution2 – VAE

$$\underbrace{\boxed{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right]} - D_{KL}(q_\phi(z \mid x^{(i)}) \,||\, \boxed{p_\theta(z)})}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$

Decoder **Encoder** **Gaussian**

📌 **Suppose you have two Gaussians:**

- $q(z) = \mathcal{N}(\mu_q, \sigma_q^2)$
- $p(z) = \mathcal{N}(\mu_p, \sigma_p^2)$

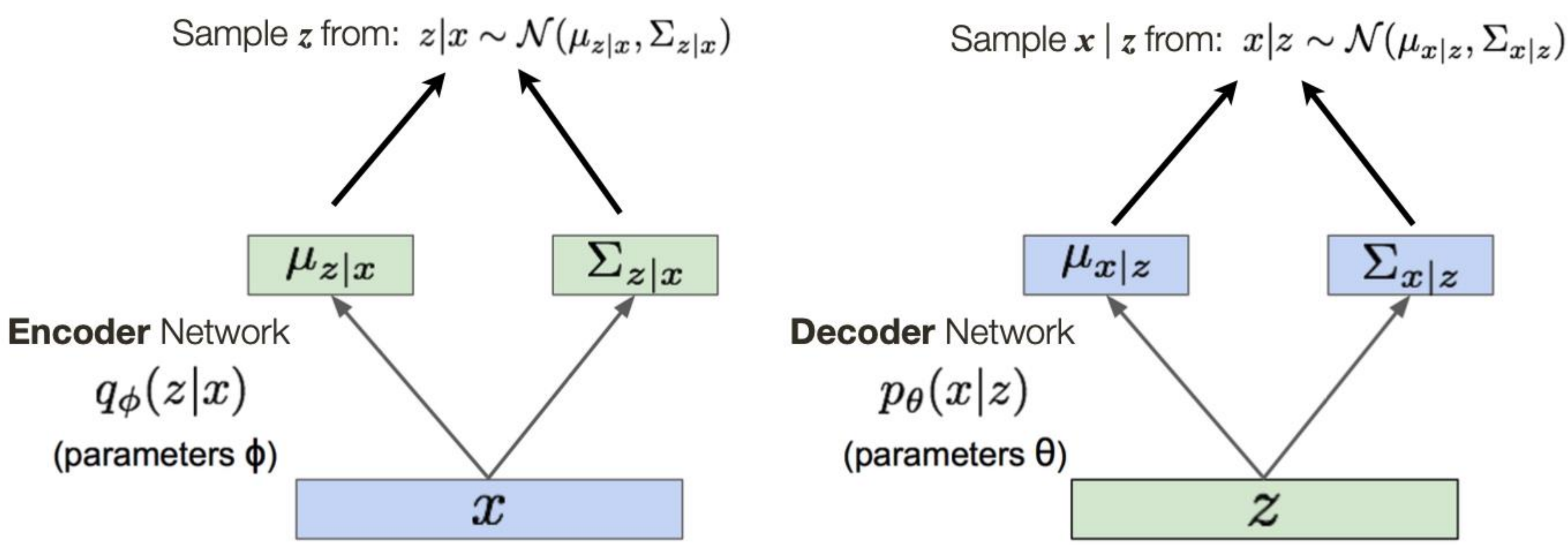
Then the **KL divergence** $\mathrm{KL}(q\,||\,p)$ is:

$$\mathrm{KL}(q\,||\,p) = \log\left(\frac{\sigma_p}{\sigma_q}\right) + \frac{\sigma_q^2 + (\mu_q - \mu_p)^2}{2\sigma_p^2} - \frac{1}{2}$$

Solution2 – VAE

$$\underbrace{\boxed{\mathbf{E}_z\left[\log p_\theta(x^{(i)} \mid z)\right]} - D_{KL}(q_\phi(z \mid x^{(i)}) \,\|\, \boxed{p_\theta(z)})}_{\mathcal{L}(x^{(i)},\theta,\phi)}$$

Decoder **Encoder** **Gaussian**



Solution2 – VAE

Codebook