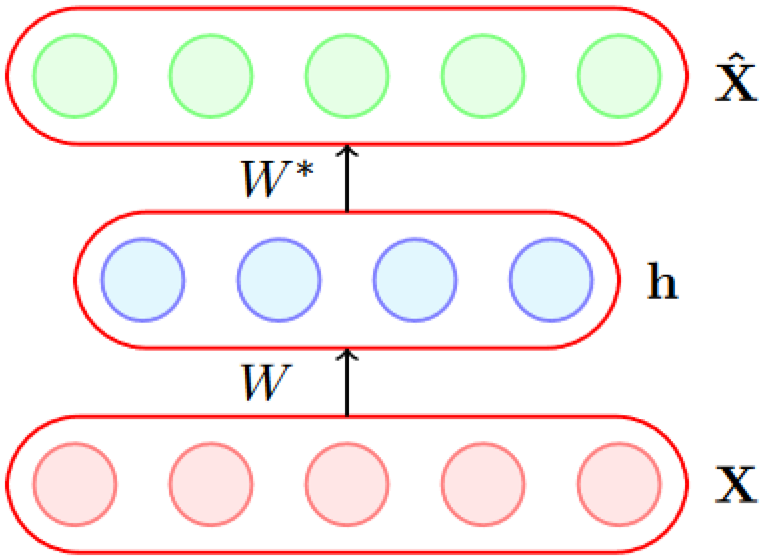
Variational Auto encoder

Biplab Banerjee

GNR 638

Auto-encoder re-visited



$$\mathbf{h} = g(W\mathbf{X} + \mathbf{b})$$

$$\hat{\mathbf{X}} = f(W^*\mathbf{h} + \mathbf{c})$$

- It contains two parts:
  - ✓ Encoder
  - ✓ Decoder
- Encoder is used for feature abstraction

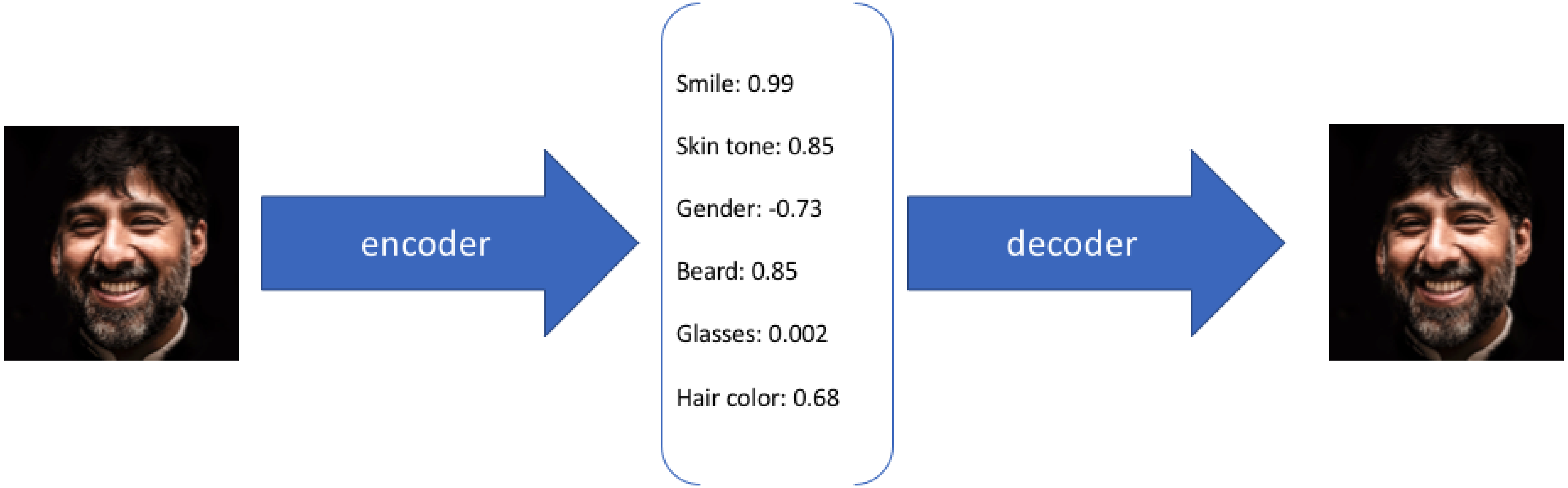
- Can this be used as a generative model?
  - ✓ Given *h*, can we generate meaningful data?

Entangled vs disentangled latent space

- In an *entangled* latent space, each dimension (or dimension cluster) of the latent representation typically encodes multiple factors of variation simultaneously. There is *no single axis* or small subset of axes that corresponds to a *single*, interpretable factor.

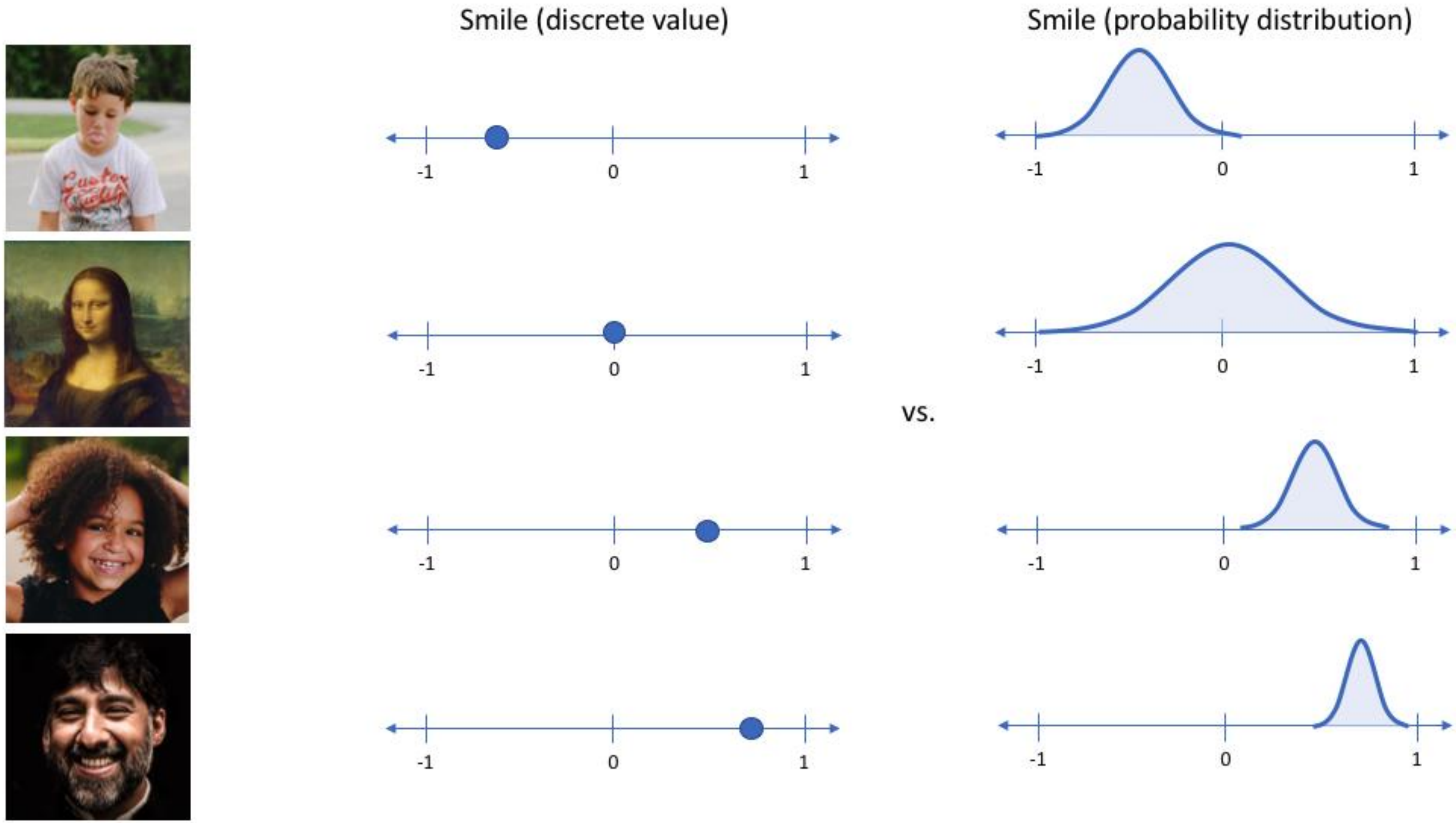
- In a *disentangled* latent space, each dimension (or small group of dimensions) is responsible for capturing *one specific factor of variation* in the data. For example, in a disentangled representation of faces, one latent dimension might correspond to hair color, another to face shape, another to lighting, etc.

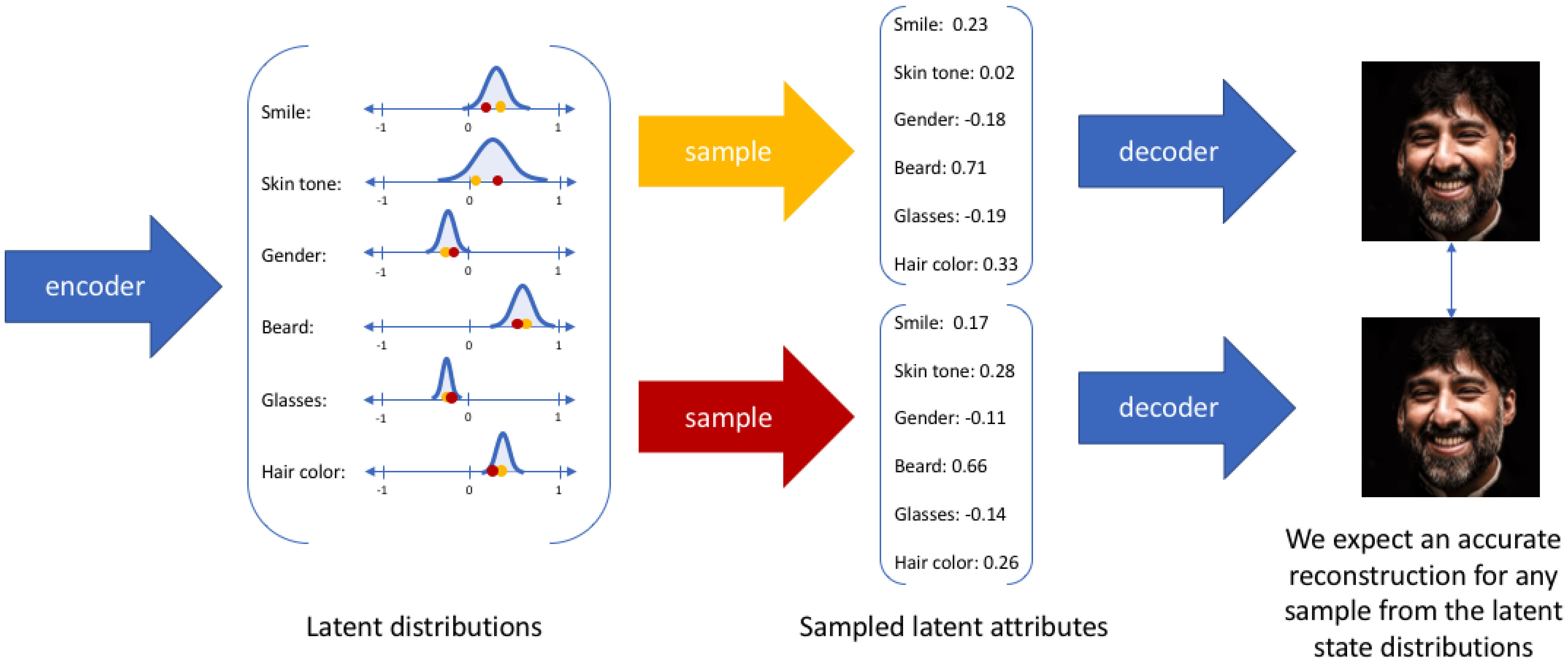
Example of disentangled latent space



Latent attributes

simple autoencoders	input **x**	encoding →	latent representation **z = e(x)**	decoding →	input reconstruction **d(z)**			
variational autoencoders	input **x**	encoding →	latent distribution **p(z|x)**	sampling →	sampled latent representation **z ~ p(z|x)**	decoding →	input reconstruction **d(z)**


Smile (discrete value)
Smile (probability distribution)
-1
0
1
vs.



encoder
Smile:
Skin tone:
Gender:
Beard:
Glasses:
Hair color:
-1
0
1
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
decoder
decoder
Latent distributions
Sampled latent attributes
We expect an accurate reconstruction for any sample from the latent state distributions

Variational inference

A **Variational Autoencoder (VAE)** is a type of **latent variable model**. In a latent variable model, we assume:

- We have observed data $\mathbf{x}$. (For simplicity, assume $\mathbf{x}$ is a single data point; you can extend to a dataset by summing or averaging across data points.)
- We introduce a latent (unobserved) variable $\mathbf{z}$ which "explains" or "generates" the data.

The joint distribution of the observed $\mathbf{x}$ and latent $\mathbf{z}$ is typically written as:

$$p(\mathbf{x}, \mathbf{z}) \;=\; p_\theta(\mathbf{x} \mid \mathbf{z})\, p(\mathbf{z}),$$

where:

- $p(\mathbf{z})$ is the *prior* on the latent variable (often chosen to be a standard Normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{I})$).
- $p_\theta(\mathbf{x} \mid \mathbf{z})$ is the *likelihood* of data $\mathbf{x}$ given the latent $\mathbf{z}$. This is governed by parameters θ, which will usually be learned (e.g., via a neural network decoder).

We want to model $\mathbf{x}$ (the data) by marginalizing out $\mathbf{z}$:

$$p_\theta(\mathbf{x}) \;=\; \int p_\theta(\mathbf{x} \mid \mathbf{z})\, p(\mathbf{z})\, d\mathbf{z}.$$

The posterior – explain the hidden structure of the data

$$p_\theta(\mathbf{z} \mid \mathbf{x}) = \frac{p_\theta(\mathbf{x}, \mathbf{z})}{p_\theta(\mathbf{x})} = \frac{p_\theta(\mathbf{x} \mid \mathbf{z})\, p(\mathbf{z})}{\int p_\theta(\mathbf{x} \mid \mathbf{z}')\, p(\mathbf{z}')\, d\mathbf{z}'}.$$

- Computing $\int p_\theta(\mathbf{x} \mid \mathbf{z}')\, p(\mathbf{z}')\, d\mathbf{z}'$ exactly is often intractable.
- Hence, computing $p_\theta(\mathbf{z} \mid \mathbf{x})$ exactly is also difficult.

Variational Inference (VI) tackles this challenge by introducing a *variational distribution* $q_\phi(\mathbf{z} \mid \mathbf{x})$—an approximation to the true posterior $p_\theta(\mathbf{z} \mid \mathbf{x})$. We choose a functional form for q_ϕ (often a Gaussian whose mean and variance are given by neural networks), and then we optimize the parameters ϕ so that $q_\phi(\mathbf{z} \mid \mathbf{x})$ is as "close" as possible to the true posterior $p_\theta(\mathbf{z} \mid \mathbf{x})$.

The evidence lower bound calculation

1. Start with the log-likelihood of the data:

$$\log p_\theta(\mathbf{x}) \;=\; \log \int p_\theta(\mathbf{x} \mid \mathbf{z})\, p(\mathbf{z})\, d\mathbf{z}.$$

2. Introduce $q_\phi(\mathbf{z} \mid \mathbf{x})$ inside the integral:

$$\log p_\theta(\mathbf{x}) = \log \int p_\theta(\mathbf{x} \mid \mathbf{z})\, p(\mathbf{z}) \frac{q_\phi(\mathbf{z} \mid \mathbf{x})}{q_\phi(\mathbf{z} \mid \mathbf{x})}\, d\mathbf{z}.$$

$$\log p_\theta(\mathbf{x}) \;=\; \log \int q_\phi(\mathbf{z} \mid \mathbf{x}) \underbrace{\left(\tfrac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right)}_{\text{call this } X(\mathbf{z})}\, d\mathbf{z}.$$

The term in parentheses can be considered a function of $\mathbf{z}$. Notice that

$$\int q_\phi(\mathbf{z} \mid \mathbf{x}) \left(\tfrac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right) d\mathbf{z} \;=\; \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\Big[X(\mathbf{z})\Big] \quad \text{where} \quad X(\mathbf{z}) \;=\; \tfrac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}.$$

So we can rewrite:

$$\log p_\theta(\mathbf{x}) \;=\; \log\Big(\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[X(\mathbf{z})\right]\Big),$$

where again,

$$X(\mathbf{z}) \;=\; \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x})}.$$

By applying Jensen's inequality (which states $\log \mathbb{E}[f(X)] \geq \mathbb{E}[\log f(X)]$), we get:

$$\log p_\theta(\mathbf{x}) \;\geq\; \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log\left(\frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x})}\right)\right].$$

Rewrite $p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x} \mid \mathbf{z})\, p(\mathbf{z})$, and split the log:

$$\log\left(\frac{p_\theta(\mathbf{x}, \mathbf{z})}{q_\phi(\mathbf{z} \mid \mathbf{x})}\right) = \log p_\theta(\mathbf{x} \mid \mathbf{z}) + \log p(\mathbf{z}) - \log q_\phi(\mathbf{z} \mid \mathbf{x}).$$

So the inequality becomes:

$$\log p_\theta(\mathbf{x}) \;\geq\; \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p_\theta(\mathbf{x} \mid \mathbf{z})\right] \;+\; \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left[\log p(\mathbf{z}) - \log q_\phi(\mathbf{z} \mid \mathbf{x})\right].$$

Recognize that

$$\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{z}) - \log q_\phi(\mathbf{z} \mid \mathbf{x})] = -\,\mathrm{KL}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, p(\mathbf{z})\big),$$

where $\mathrm{KL}(\cdot\|\cdot)$ is the Kullback–Leibler divergence. Hence, we arrive at the **ELBO**:

$$\underbrace{\log p_\theta(\mathbf{x})}_{\text{log-likelihood}} \;\geq\; \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\big[\log p_\theta(\mathbf{x} \mid \mathbf{z})\big] - \mathrm{KL}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, p(\mathbf{z})\big)}_{\mathrm{ELBO}(\theta,\phi)}.$$

Maximizing this ELBO w.r.t. θ (decoder parameters) and ϕ (encoder or inference parameters) is *equivalent* to minimizing the KL divergence between q_ϕ and the true posterior $\propto p_\theta(\mathbf{z} \mid \mathbf{x})$. In practice, we do *stochastic gradient ascent* on the ELBO.

VAE – basic setting

1. **Observed and Latent Variables**
   - Let $\mathbf{x}$ be observed data (e.g., an image).
   - Introduce a latent variable $\mathbf{z}$.
   - The model posits that $\mathbf{x}$ is generated from $\mathbf{z}$ via some **decoder** (generative model).
2. **Prior on Latent Variable**
   - Typically, we choose a simple prior $p(\mathbf{z})$, like $\mathcal{N}(\mathbf{0}, \mathbf{I})$.
   - So the joint distribution is $p_\theta(\mathbf{x}, \mathbf{z}) = p_\theta(\mathbf{x} \mid \mathbf{z})\, p(\mathbf{z})$.
3. **Intractable Posterior**
   - The posterior $p_\theta(\mathbf{z} \mid \mathbf{x}) = \frac{p_\theta(\mathbf{x},\mathbf{z})}{p_\theta(\mathbf{x})}$ is typically intractable to compute exactly.
   - $p_\theta(\mathbf{x}) = \int p_\theta(\mathbf{x} \mid \mathbf{z})\, p(\mathbf{z})\, d\mathbf{z}$ can't be computed in closed form with complex neural networks in the generative model.

Structure of VAE

A VAE is built from two main neural networks:

1. **Encoder (Inference Network):**

$$q_\phi(\mathbf{z} \mid \mathbf{x})$$

   - A neural net that takes $\mathbf{x}$ as input and outputs parameters of a distribution over $\mathbf{z}$ (e.g., mean $\boldsymbol{\mu}_\phi(\mathbf{x})$ and variance $\boldsymbol{\sigma}^2_\phi(\mathbf{x})$ for a Gaussian).
   - This is the *approximate posterior* over $\mathbf{z}$.

2. **Decoder (Generative Network):**

$$p_\theta(\mathbf{x} \mid \mathbf{z})$$

   - A neural net that takes $\mathbf{z}$ as input and outputs a distribution over $\mathbf{x}$ (e.g., a Gaussian or Bernoulli for each dimension).
   - This describes how $\mathbf{x}$ is "reconstructed" or generated from $\mathbf{z}$.

Why "autoencoder"?

- The *encoder* compresses $\mathbf{x}$ to $\mathbf{z}$-space (mean + variance).
- The *decoder* reconstructs (or generates) $\mathbf{x}$ from $\mathbf{z}$.

Deterministic vs stochastic encoder

Deterministic Encoder

$$\mathbf{h} = \mathrm{Encoder}(\mathbf{x})$$

In a conventional autoencoder, the encoder is a function (often a neural network) that deterministically maps the input $\mathbf{x}$ to a code or embedding $\mathbf{h}$. Given the same $\mathbf{x}$, it always outputs the same $\mathbf{h}$.

Probabilistic (or Variational) Encoder

$$q_\phi(\mathbf{z} \mid \mathbf{x}) \quad \longleftarrow \quad \mathrm{Encoder}$$

In a **probabilistic** or **variational** encoder (such as in a Variational Autoencoder, VAE), we map $\mathbf{x}$ to a *distribution* over latent codes $\mathbf{z}$. Concretely, the encoder might output the *mean* and *variance* of a Gaussian, from which we can then **sample** $\mathbf{z}$. This means that, for the same $\mathbf{x}$, we can draw slightly different $\mathbf{z}$-values every time—reflecting uncertainty or variability in how $\mathbf{x}$ might be explained by latent factors.

$$q_\phi(\mathbf{z} \mid \mathbf{x}) = \mathcal{N}\big(\mathbf{z};\, \boldsymbol{\mu}_\phi(\mathbf{x}),\, \mathrm{diag}(\boldsymbol{\sigma}^2_\phi(\mathbf{x}))\big).$$

The ELBO objective

We want to learn θ (decoder parameters) and ϕ (encoder parameters) so as to maximize the (log) likelihood of data $\mathbf{x}$. Because $p_\theta(\mathbf{x})$ is intractable, we maximize a *lower bound*, the **ELBO**:

$$\log p_\theta(\mathbf{x}) \;\; \geq \;\; \underbrace{\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\Big[\log p_\theta(\mathbf{x} \mid \mathbf{z})\Big] \; - \; \mathrm{KL}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, p(\mathbf{z})\big)}_{\mathrm{ELBO}(\theta,\phi)}.$$

1. **Likelihood Term:** $\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x} \mid \mathbf{z})]$ measures how well the decoder reconstructs $\mathbf{x}$ from latent samples $\mathbf{z}$.
2. **Regularization/KL Term**: $-\mathrm{KL}(q_\phi(\mathbf{z} \mid \mathbf{x}) \| \; p(\mathbf{z}))$ encourages the approximate posterior to stay close to the simple prior $p(\mathbf{z})$, preventing the latent space from "overfitting" the data.

Reparameterization trick

1. **We want**: A random variable z whose distribution depends on ϕ. In VAEs, for example, $z \sim \mathcal{N}(\mu(\phi), \sigma^2(\phi))$.
2. **Problem**: If you say "z = `sample from distribution that depends on \(\phi\)`," then inside your code or math, you have an operation that looks like:

   $$z = \mathrm{RandomGenerator}(\phi).$$

   You can't do normal $\frac{d}{d\phi}$ of that, because the random generator is like a "black box" that changes distribution with ϕ.
3. **Solution (Reparameterization)**: Break it into two parts:
   - (A) **Pure randomness:** a random draw ε from a **fixed** distribution that does not depend on ϕ.
   - (B) **Deterministic transform** of ϕ and ε:

   $$z = f(\phi,\ \varepsilon).$$

   In a Gaussian case, we do:

   $$\varepsilon \sim \mathcal{N}(0,1) \quad \text{(fixed distribution, no } \phi\text{)}, \quad z = \mu(\phi) + \sigma(\phi) \times \varepsilon \quad \text{(deterministic transform)}.$$

   Because ε does **not** depend on ϕ, we now have a direct algebraic expression for z. We can apply normal **chain rule** through $\mu(\phi)$, $\sigma(\phi)$, etc.

The whole steps of VAE

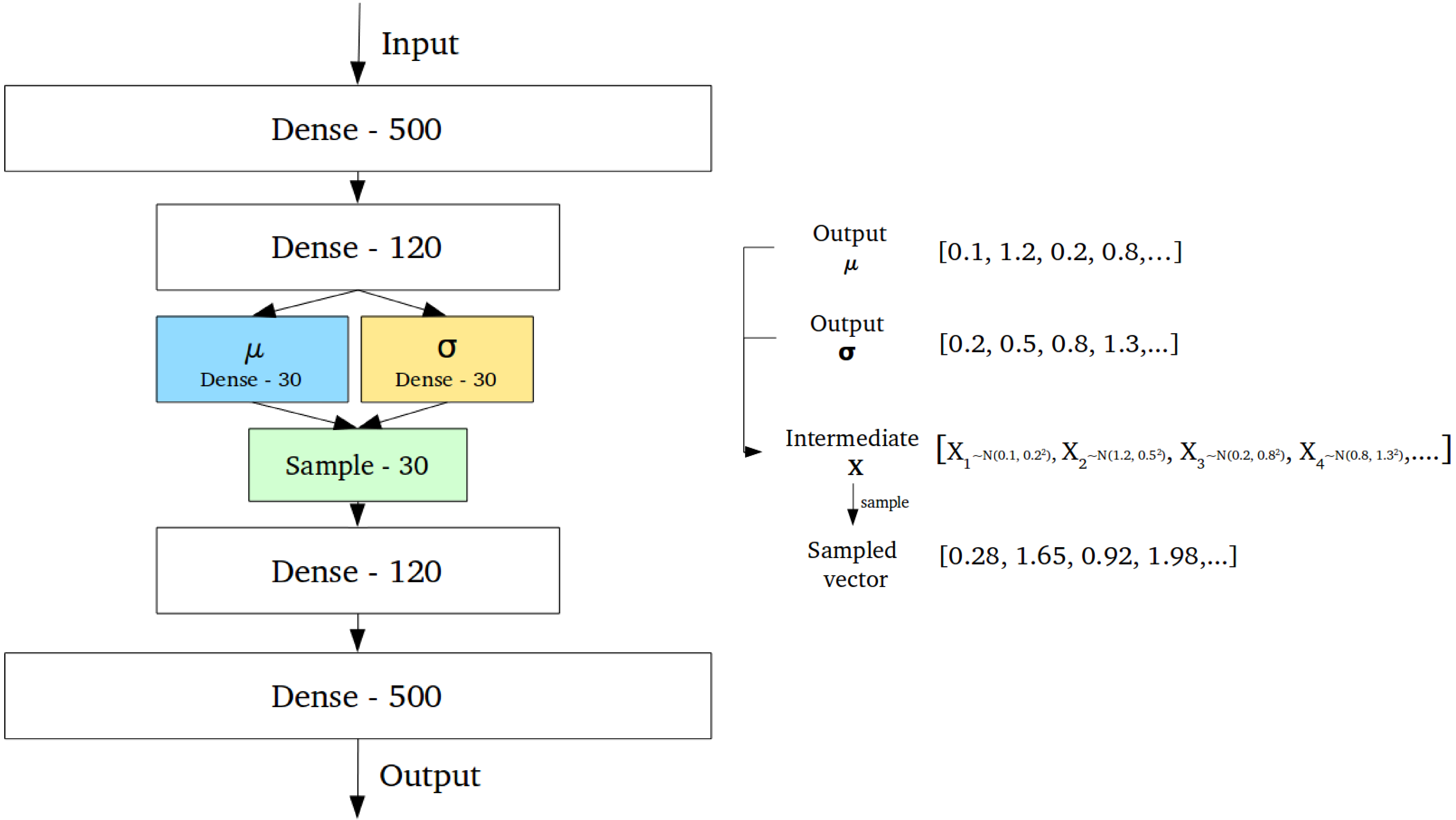
1. **Sample** a mini-batch of data points $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(M)}$.
2. For each $\mathbf{x}^{(i)}$:
   - Encode $\mathbf{x}^{(i)}$ into parameters $\boldsymbol{\mu}_\phi(\mathbf{x}^{(i)}), \boldsymbol{\sigma}_\phi(\mathbf{x}^{(i)})$.
   - Sample $\boldsymbol{\epsilon}^{(i)} \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$.
   - Compute $\mathbf{z}^{(i)} = \boldsymbol{\mu}_\phi(\mathbf{x}^{(i)}) + \boldsymbol{\sigma}_\phi(\mathbf{x}^{(i)}) \odot \boldsymbol{\epsilon}^{(i)}$.
   - Decode $\mathbf{z}^{(i)}$ to get $p_\theta(\mathbf{x} \mid \mathbf{z}^{(i)})$.
3. Compute the stochastic estimator of the ELBO:

$$\mathrm{ELBO}(\theta, \phi; \mathbf{x}^{(i)}) \approx \frac{1}{M} \sum_{i=1}^{M} \Big[\log p_\theta(\mathbf{x}^{(i)} \mid \mathbf{z}^{(i)}) - \mathrm{KL}\big(q_\phi(\mathbf{z}^{(i)} \mid \mathbf{x}^{(i)}) \,\|\, p(\mathbf{z}^{(i)})\big) \Big].$$

4. **Ascend** on this ELBO w.r.t. θ and ϕ (or equivalently, do gradient descent on the negative ELBO).

Some takeaways

1. We replace an intractable posterior $p(\mathbf{z} \mid \mathbf{x})$ with a tractable approximation $q_\phi(\mathbf{z} \mid \mathbf{x})$.
2. We measure "closeness" via $\mathrm{KL}(q_\phi \| p)$.
3. We then rearrange the log-likelihood to form the ELBO, which is a lower bound that becomes tight if q_ϕ matches the true posterior exactly.
4. We perform gradient-based optimization on that bound (often using the reparameterization trick or other gradient estimators).



Input
Dense - 500
Dense - 120
μ
Dense - 30
σ
Dense - 30
Sample - 30
Dense - 120
Dense - 500
Output
Output
μ
[0.1, 1.2, 0.2, 0.8,...]
Output
σ
[0.2, 0.5, 0.8, 1.3,...]
Intermediate
X
$[X_{1 \sim N(0.1,\, 0.2^2)}, X_{2 \sim N(1.2,\, 0.5^2)}, X_{3 \sim N(0.2,\, 0.8^2)}, X_{4 \sim N(0.8,\, 1.3^2)}, \ldots]$
sample
Sampled
vector
[0.28, 1.65, 0.92, 1.98,...]

Why does VAE latent space is disentangled

- **Factorized Prior:**
  VAEs typically use an isotropic Gaussian prior

$$p(\mathbf{z}) = \mathcal{N}(\mathbf{0}, \mathbf{I}),$$

  which is fully factorized (each latent dimension is independent).

- **KL Divergence Term:**
  In the ELBO, the KL divergence

$$\mathrm{KL}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, p(\mathbf{z})\big)$$

  forces the approximate posterior $q_\phi(\mathbf{z} \mid \mathbf{x})$ to be close to this independent prior. This regularization penalizes correlations among latent dimensions, which in theory encourages each dimension to capture distinct aspects of the data.

Beta VAE

β-**VAE** is a **variation** of the standard Variational Autoencoder in which one introduces a hyperparameter β to **weight** the Kullback–Leibler (KL) term in the Evidence Lower Bound (ELBO). Formally, instead of minimizing

$$\mathrm{Loss}(\theta, \phi) \;=\; -\,\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x} \mid \mathbf{z})] \;+\; \mathrm{KL}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, p(\mathbf{z})\big),$$

a β-**VAE** minimizes

$$\mathrm{Loss}_\beta(\theta, \phi) \;=\; -\,\mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}[\log p_\theta(\mathbf{x} \mid \mathbf{z})] \;+\; \beta\,\mathrm{KL}\big(q_\phi(\mathbf{z} \mid \mathbf{x}) \,\|\, p(\mathbf{z})\big).$$

- **When** $\beta > 1$:
  - You typically get **more disentangled** latent factors.
  - But the **reconstruction quality** might degrade, because the model is forced to compress the data more aggressively (i.e., it's "penalized" more heavily for large KL).
- **When** $\beta < 1$:
  - You put **less** pressure on the latent space to match the prior; the decoder can better "memorize" or more richly encode the data.

Hierarchical VAE

- **Encoder (Bottom-Up):**
  It maps the input $\mathbf{x}$ to two sets of latent parameters:
  - $q(\mathbf{z}_1 \mid \mathbf{x})$ with parameters $(\mu_{z1}, \log \sigma^2_{z1})$
  - $q(\mathbf{z}_2 \mid \mathbf{x})$ with parameters $(\mu_{z2}, \log \sigma^2_{z2})$
    (In more sophisticated designs, the posterior for the top latent variable may be conditioned on intermediate features or even on $\mathbf{z}_1$; here we keep it simple.)
- **Decoder (Top-Down):**
  It first defines a conditional prior $p(\mathbf{z}_1 \mid \mathbf{z}_2)$ and then generates $\mathbf{x}$ from $\mathbf{z}_1$ via $p(\mathbf{x} \mid \mathbf{z}_1)$.
  We assume a standard Gaussian prior for $\mathbf{z}_2$: $p(\mathbf{z}_2) = \mathcal{N}(\mathbf{0}, \mathbf{I})$.

- The generation of data is modeled as a **top-down** process. For example:

$$p(\mathbf{x}, \mathbf{z}_1, \mathbf{z}_2) = p(\mathbf{x} \mid \mathbf{z}_1)\, p(\mathbf{z}_1 \mid \mathbf{z}_2)\, p(\mathbf{z}_2),$$

  where:
  - $p(\mathbf{z}_2)$ is a simple prior (e.g., $\mathcal{N}(0, I)$).
  - $p(\mathbf{z}_1 \mid \mathbf{z}_2)$ models the dependency between higher and lower latent variables.
  - $p(\mathbf{x} \mid \mathbf{z}_1)$ decodes to the observed data.

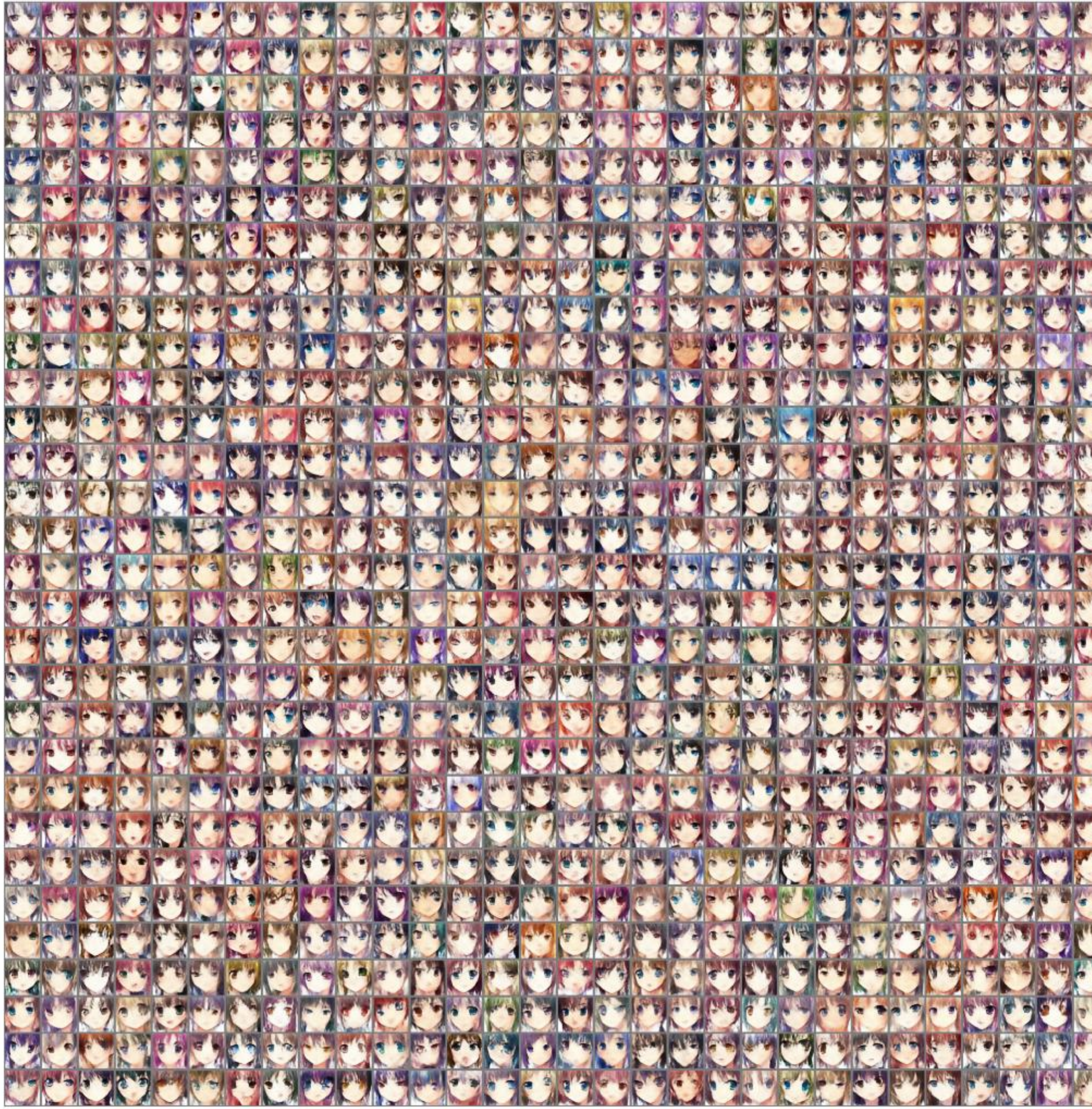
A demo codebase

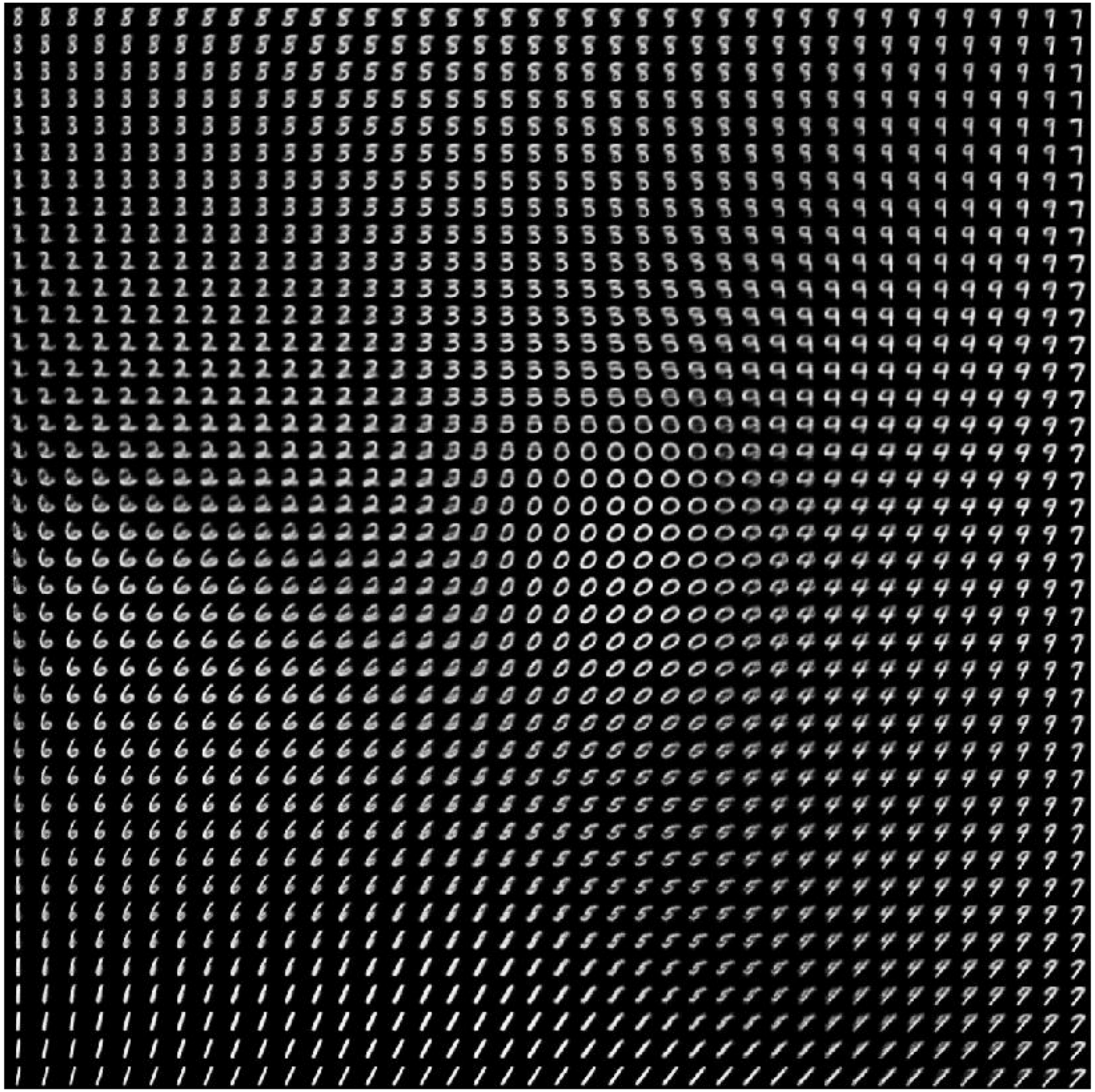
```python
class Encoder(nn.Module):
    def __init__(self, input_dim=784, hidden_dim=400, latent_dim=20):
        super().__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc_mu = nn.Linear(hidden_dim, latent_dim)      # outputs mean
        self.fc_logvar = nn.Linear(hidden_dim, latent_dim)  # outputs log-variance

    def forward(self, x):
        # x shape: [batch_size, 784]
        h = F.relu(self.fc1(x))
        mu = self.fc_mu(h)
        logvar = self.fc_logvar(h)
        return mu, logvar  # both [batch_size, latent_dim]
```

```python
class Decoder(nn.Module):
    def __init__(self, latent_dim=20, hidden_dim=400, output_dim=784):
        super().__init__()
        self.fc1 = nn.Linear(latent_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, output_dim)

    def forward(self, z):
        # z shape: [batch_size, latent_dim]
        h = F.relu(self.fc1(z))
        # Output is passed through a sigmoid for pixel intensities in [0,1]
        x_recon = torch.sigmoid(self.fc2(h))
        return x_recon  # shape: [batch_size, 784]
```

```python
class VAE(nn.Module):
    def __init__(self, input_dim=784, hidden_dim=400, latent_dim=20):
        super().__init__()
        self.encoder = Encoder(input_dim, hidden_dim, latent_dim)
        self.decoder = Decoder(latent_dim, hidden_dim, input_dim)

    def reparameterize(self, mu, logvar):
        """
        Reparameterization trick:
          z = mu + sigma * epsilon,
        where epsilon ~ N(0, I).
        """
        std = torch.exp(0.5 * logvar)
        eps = torch.randn_like(std)  # same shape as std
        return mu + eps * std

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

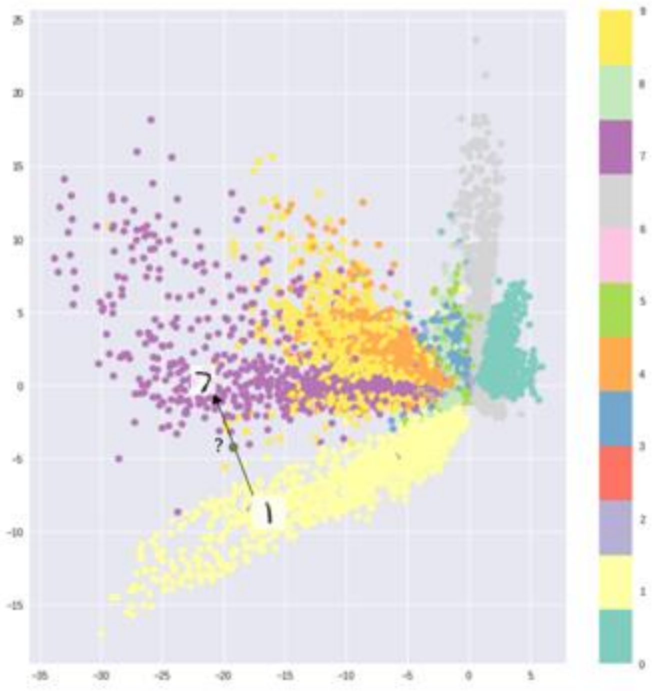
```python
def vae_loss(x, x_recon, mu, logvar):
    """
    1) Reconstruction term: Binary Cross-Entropy (BCE)
    2) KL Divergence term:
       D_KL(q(z|x) || p(z)) = -0.5 * sum(1 + logvar - mu^2 - exp(logvar))
    """
    # BCE expects x_recon in [0,1], x in [0,1].
    # 'reduction=sum' sums over ALL pixels in the batch.
    BCE = F.binary_cross_entropy(x_recon, x, reduction='sum')

    # KL divergence
    KLD = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())

    return BCE + KLD
```



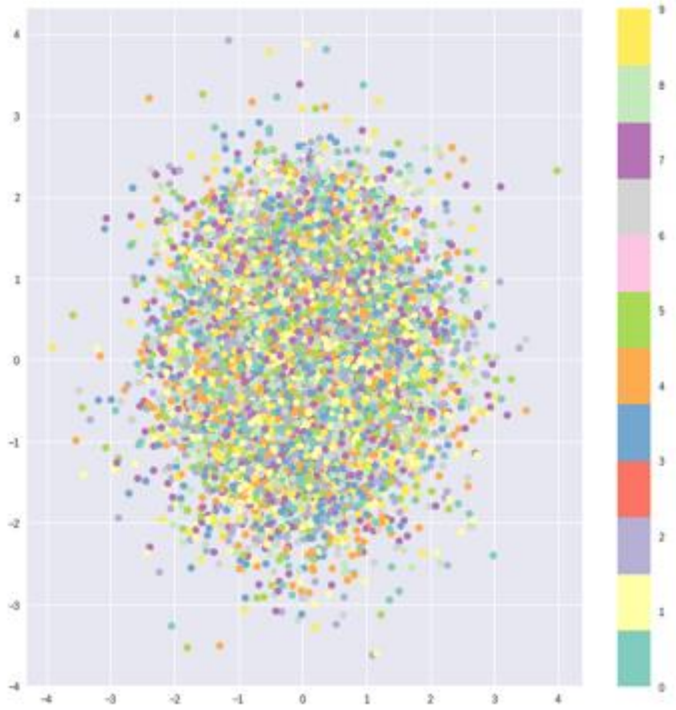


Visualization of the latent space

Only reconstruction loss



Only KL divergence



Combination