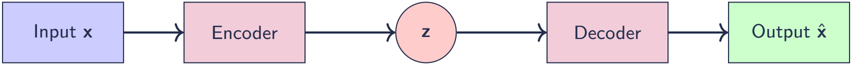
Lecture 12.3: GenAI: Variational Autoencoders

Heman Shakeri

Recall: The Autoencoder from Module 6

We've seen autoencoders as unsupervised learning tools:



Goal: Minimize reconstruction error $\|\mathbf{x} - \hat{\mathbf{x}}\|^2$

Latent code $\mathbf{z}$: Compressed representation of input

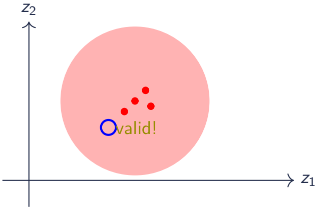
The Problem: Can we generate new samples?

Try sampling random $\mathbf{z}$ and decoding...

Recall the Latent Space Geometry

What we hope for:

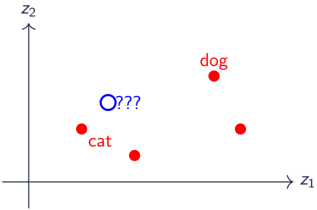
- Continuous latent space
- Smooth interpolation
- Every point decodes to something meaningful



Variational Autoencoder

What we actually get:

- Scattered, disconnected regions
- Random points → garbage
- No principled way to sample



Standard Autoencoder

We need the latent space to be a *continuous probability distribution*!

VAE

The Paper That Started It All

"Auto-Encoding Variational Bayes" (2013)
Diederik P. Kingma and Max Welling
`https://arxiv.org/abs/1312.6114`

Idea: Instead of learning a single point $\mathbf{z}$ for each input, learn a *distribution* over $\mathbf{z}$.

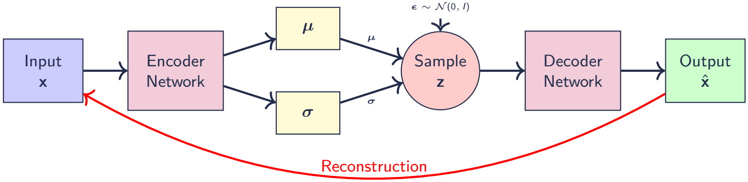
Specifically: Learn parameters $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}$ of a Gaussian

$$q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}(\mathbf{x}), \mathrm{diag}(\boldsymbol{\sigma}^2(\mathbf{x})))$$

Why this is genius: We can now sample from the latent space to generate!

VAE Architecture

Key difference from standard AE: Encoder outputs distribution parameters, not a point!



The Solution: A Lower Bound

Strategy: Use a Tractable Proxy

Since we can't maximize $\log p(\mathbf{x})$, we find a new, tractable function (the ELBO) that is a lower bound. By

maximizing this proxy, we push up the true likelihood.

The Jensen's Inequality Trick

We use it because log is a **concave** function, which means: $\log(\mathbb{E}[Y]) \geq \mathbb{E}[\log(Y)]$. This one rule lets us create the bound.

Terse Derivation

1. Start with $\log p(\mathbf{x})$ and introduce $q(\mathbf{z}|\mathbf{x})$:

$$= \log \int q(\mathbf{z}|\mathbf{x}) \frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z}|\mathbf{x})} d\mathbf{z}$$

2. Rewrite as an expectation:

$$= \log \left(\mathbb{E}_q \left[\frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \right] \right)$$

3. Apply Jensen's (move log inside):

$$\geq \mathbb{E}_q \left[\log \left(\frac{p(\mathbf{x}, \mathbf{z})}{q(\mathbf{z}|\mathbf{x})} \right) \right]$$

4. Rearrange... and we get the ELBO!

The Beautiful Math: ELBO

The Challenge: We want to maximize $\log p(\mathbf{x})$ (likelihood of our data), but it's intractable!

$$\log p(\mathbf{x}) = \log \int p(\mathbf{x}|\mathbf{z})p(\mathbf{z})d\mathbf{z}$$

Solution: Introduce approximate posterior $q(\mathbf{z}|\mathbf{x})$ and derive a lower bound

Using Jensen's inequality, we get the **Evidence Lower BOund (ELBO)**:

$$\log p(\mathbf{x}) \geq \mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})] - D_{KL}(q(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z}))$$

This is what we maximize!

Two components:

1. Reconstruction term: How well can we reconstruct $\mathbf{x}$ from $\mathbf{z}$?
2. Regularization term: How close is our posterior to the prior?

Understanding the ELBO Intuitively

$$\mathcal{L} = \underbrace{\mathbb{E}_{q(\mathbf{z}|\mathbf{x})}[\log p(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction term}} - \underbrace{D_{KL}(q(\mathbf{z}|\mathbf{x})\|p(\mathbf{z}))}_{\text{keep the latent space tidy!}}$$

Reconstruction term: Standard autoencoder objective

- "Can I decode $\mathbf{z}$ back to $\mathbf{x}$?"
- Encourages preserving information

KL divergence term: The magic ingredient!

- "Is $q(\mathbf{z}|\mathbf{x})$ close to standard normal $\mathcal{N}(0, I)$?"
- Forces latent codes to be well-behaved
- Prevents encoder from cheating by using arbitrary regions
- Creates continuous, smooth latent space

The trade-off: Balance reconstruction quality vs. latent space structure

The KL Divergence: Forcing Structure

For Gaussians, the KL divergence has a closed form!

$$D_{KL}(q(\mathbf{z}|\mathbf{x}) \| p(\mathbf{z})) = \frac{1}{2} \sum_{j=1}^{J} \left(\mu_j^2 + \sigma_j^2 - \log \sigma_j^2 - 1 \right)$$

where J is the latent dimension.

What does this do?

- Pulls $\boldsymbol{\mu}$ toward zero
- Pulls $\boldsymbol{\sigma}$ toward one
- Prevents collapse to deterministic encoding

Result: All latent codes live in a similar region around $\mathcal{N}(0, I)$

This means random samples from $\mathcal{N}(0, I)$ will decode to valid outputs!

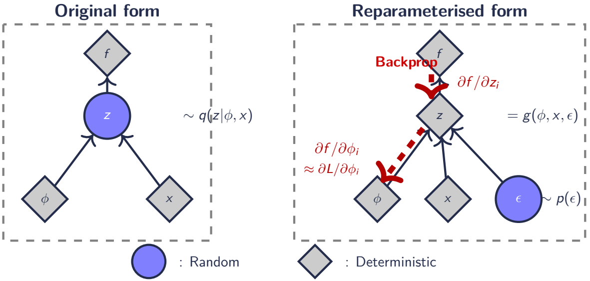
The Reparameterization Trick: Visual

Problem: We need to sample $\mathbf{z} \sim q(\mathbf{z}|\mathbf{x})$, but backpropagation cannot flow through a random sampling operation.

$$\mathbf{z} \sim q(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2)$$

Sampling is *not* differentiable! $\rightarrow$ Reparameterize the sampling:
Instead of sampling $\mathbf{z}$ directly, write:

$$\mathbf{z} = \boldsymbol{\mu} + \boldsymbol{\sigma} \odot \boldsymbol{\epsilon}, \quad \text{where } \boldsymbol{\epsilon} \sim \mathcal{N}(0, I)$$



[Kingma, 2013; Bengio, 2013; Rezende et al 2014]

β-VAE: Forcing Disentanglement

- A standard VAE ($\beta = 1$) has a problem.
- The reconstruction term often "wins" the trade-off, forcing the model to be a perfect autoencoder.
- The result: it ignores the KL term, creating a messy, **entangled** latent space just to pass information.

What is Disentanglement?

We want each latent dimension z_i to control *one single, independent* factor of the data.

Example:

- For faces, z_1 might control "smile," z_2 controls "head rotation," and z_3 controls "skin tone."
- A simple VAE fails at this; z_1 might control both smile *and* rotation (entangled).

The β-VAE Solution

We add a simple hyperparameter β to the KL term:

$$\mathcal{L} = \mathbb{E}_q[\log p(\mathbf{x}|\mathbf{z})] - \boldsymbol{\beta} \cdot D_{KL}(q \| p)$$

Intuition: When $\boldsymbol{\beta > 1}$, we put *more pressure* on the model to be structured and **forced** to find the most efficient encoding: the true, underlying, independent factors. This is disentanglement.

PyTorch Implementation

```python
class VAE(nn.Module):
    def __init__(self, input_dim, hidden_dim, latent_dim):
        super().__init__()
        # Encoder
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU()
        )
        self.fc_mu = nn.Linear(hidden_dim, latent_dim)
        self.fc_logvar = nn.Linear(hidden_dim, latent_dim)

        # Decoder
        self.decoder = nn.Sequential(
            nn.Linear(latent_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Linear(hidden_dim, input_dim),
            nn.Sigmoid()
        )
```

VAE Forward Pass and Loss

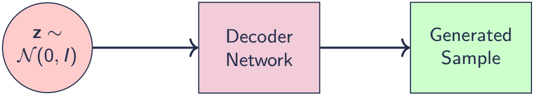
```
    def encode(self, x):
        h = self.encoder(x)
        mu = self.fc_mu(h)
        logvar = self.fc_logvar(h)
        return mu, logvar

    def reparameterize(self, mu, logvar):
        std = torch.exp(0.5 * logvar)  # sigma = exp(0.5 * log(sigma^2))
        eps = torch.randn_like(std)    # Sample epsilon ~ N(0,1)
        return mu + eps * std           # z = mu + sigma * epsilon

    def forward(self, x):
        mu, logvar = self.encode(x)
        z = self.reparameterize(mu, logvar)
        recon_x = self.decoder(z)
        return recon_x, mu, logvar

def vae_loss(recon_x, x, mu, logvar):
    # Reconstruction loss (binary cross-entropy)
    recon_loss = F.binary_cross_entropy(recon_x, x, reduction='sum')
    # KL divergence
    kl_loss = -0.5 * torch.sum(1 + logvar - mu.pow(2) - logvar.exp())
    return recon_loss + kl_loss
```

Generation: Sampling from the Prior

Training: Encode data $\rightarrow$ sample latent $\rightarrow$ decode

Generation: Skip the encoder!

1. Sample $\mathbf{z} \sim \mathcal{N}(0, I)$ directly from prior
2. Pass through decoder: $\hat{\mathbf{x}} = \text{Decoder}(\mathbf{z})$
3. Get a new sample!



Why this works: KL term forced all encodings near $\mathcal{N}(0, I)$

Random samples from this region decode to valid data!

Interpolation: The Power of Continuous Space

Because latent space is continuous, we can interpolate!

Procedure:

1. Encode two images: $\mathbf{z}_1 = \text{Encode}(\mathbf{x}_1)$, $\mathbf{z}_2 = \text{Encode}(\mathbf{x}_2)$
2. Interpolate: $\mathbf{z}_t = (1 - t)\mathbf{z}_1 + t\mathbf{z}_2$ for $t \in [0, 1]$
3. Decode: $\mathbf{x}_t = \text{Decode}(\mathbf{z}_t)$

Smooth transition!

This doesn't work with standard autoencoders because their latent space has holes!

Connection to Diffusion Models

Remember Latent Diffusion from the previous lecture?

Stable Diffusion uses a VAE!

- Pre-trained VAE encoder compresses images 8×
- Diffusion happens in latent space (64×64 instead of 512×512)
- VAE decoder upsamples final result

Why VAE specifically?

- Continuous latent space → perfect for diffusion
- Learned compression preserves important features
- Separate the compression problem from generation

This is a brilliant example of composing different generative models!

Limitations of VAEs

Blurriness problem:

- VAE samples tend to be blurrier than GANs
- Why? Reconstruction loss averages over possibilities
- The prior assumption (Gaussian) may be too restrictive

Posterior collapse:

- Sometimes decoder ignores latent code
- KL term goes to zero, no information in $\mathbf{z}$
- Solutions: Annealing β, architectural changes

Difficulty with complex distributions:

- Gaussian assumption may be too simple
- Real data distributions are multimodal, complex
- Led to variants: VQ-VAE, Normalizing Flows, etc.

VAE Variants and Extensions

Conditional VAE (CVAE):

- Condition on class labels or attributes
- Control what to generate

VQ-VAE (Vector Quantized VAE):

- Discrete latent space instead of continuous
- Used in DALL-E, better for high-quality images
- Learned codebook of latent vectors

Hierarchical VAE:

- Multiple levels of latent variables
- Captures structure at different scales

Importance Weighted AE (IWAE):

- Tighter bound on log-likelihood
- Better density estimation

Why VAEs Matter

Theoretical elegance:

- Principled probabilistic framework
- Interpretable objective (ELBO)
- Connections to information theory, Bayesian inference

Practical applications:

- Image generation and compression
- Anomaly detection (reconstruction error)
- Representation learning for downstream tasks
- Semi-supervised learning
- Data imputation and denoising

Foundation for modern generative models:

- Ideas appear in diffusion models
- VQ-VAE powers DALL-E
- Latent space manipulation techniques

Key Takeaways

1. **The Core Idea:**
   - Encode data as distributions, not points
   - Use reparameterization trick for backprop
2. **The ELBO:**
   - Reconstruction: preserve information
   - KL divergence: regularize latent space
3. **The Power:**
   - Principled generation by sampling from prior
   - Continuous latent space enables interpolation
   - Probabilistic framework with theoretical guarantees
4. **The Legacy:**
   - Foundation for modern generative AI
   - Still used in Stable Diffusion today
   - Inspired countless variants and improvements