

(Conditional) Variational Autoencoders

Gabriel Rolland

HES-SO University of Applied Sciences and Arts Western Switzerland

ABSTRACT

A report on Variational Autoencoders and their Conditional variant. To be read with the associated jupyter notebook.

6 December 2025

(Conditional) Variational Autoencoders

Gabriel Rolland

HES-SO University of Applied Sciences and Arts Western Switzerland

1. Autoencoders

An autoencoder is a type of neural network where we encode an input \mathbf{x} to a latent representation \mathbf{z} we then pass this latent representation to a decoder whose task is to reconstruct the output from the latent variable \mathbf{z} . Autoencoders *per se* are not capable of generation. They only reconstruct from a latent representation. This is why we augment them with variational methods to provide them with the ability to generate new samples.

2. Variational Autoencoders (VAE)

Variational autoencoders are made in such a way to allow generation of output instead of mere reconstruction. As said, instead of representing the latent \mathbf{z} as a single point, a VAE encodes it as a distribution. Since \mathbf{z} is a distribution, we can freely sample from this distribution and here we have our generation! Good.

Equation (1) shows the Expectation Lower Bound Objective. Where Λ denotes our ELBO objective function and E denotes expectation. We can see that it consists of the Expectation of the reconstruction and the KL divergence as a regularizer.

$$\Lambda(\theta, \phi, x) = E_{q_\phi(z|x)}[\log p_\theta(x|z)] - D_{KL}(q_\phi(z|x)||p(z)) \quad (1)$$

Our implementation of a VAE is thus:

- 1) Use a fully-connected MLP to encode our input in \mathbf{z}
- 2) Reparametrize our mean and logvar by sampling a Normal distribution
- 3) Use an MLP with reverse architecture to decode the reparametrized \mathbf{z}

```

def encode(self, x):
    # TODO: Implement encoder forward pass
    # 1. Flatten the input x to (batch_size, input_dim)
    # 2. Pass through encoder_shared
    # 3. Get mu and logvar from the heads
    # Return: mu, logvar
    x = x.view(-1, self.input_dim)
    x = self.encoder_shared(x)
    mu = self.fc_mu(x)
    logvar = self.fc_logvar(x)
    return mu, logvar

def reparameterize(self, mu, logvar):
    # TODO: Calculate the standard deviation from log variance
    # TODO: Sample epsilon from a standard normal distribution
    # TODO: Return z
    std = torch.exp(0.5 * logvar)
    eps = torch.randn_like(logvar).to(DEVICE)
    z = mu + std * eps
    return z

def decode(self, z):
    # TODO: Implement decoder forward pass
    # 1. Pass z through decoder
    # 2. Reshape output to (batch_size, 1, 28, 28)
    # Return: reconstructed image
    recon_x_flat = self.decoder(z)
    return recon_x_flat.view(-1, 1, 28, 28)

def forward(self, x):
    # TODO: Implement the full forward pass
    # 1. Encode
    # 2. Reparameterize
    # 3. Decode
    # Return: recon_x, mu, logvar
    mu, logvar = self.encode(x)
    z = self.reparameterize(mu, logvar)
    recon_x = self.decode(z)
    return recon_x, mu, logvar

```

3. Conditional Variational Autoencoders (cVAE)

Our VAE is able to generate image. cVAE has the extra power to generate according to some condition.

$$\Lambda(\theta, \phi, x) = E_{q_\phi(z|x, c)}[\log p_\theta(x|z, c)] - D_{KL}(q_\phi(z|x, c) || p(z)) \quad (2)$$

```

def encode(self, x, c):
    # 1. Flatten x
    # 2. Concatenate x_flat and c
    # 3. Pass through encoder_shared and heads
    # Return: mu, logvar
    x = x.view(-1, self.input_dim)
    x_c = torch.cat((x, c), dim=1)
    x = self.encoder_shared(x_c)
    mu = self.fc_mu(x)
    logvar = self.fc_logvar(x)
    return mu, logvar
# And we do the same trick in the decoder

```

4. Extra Task

Training excluding KL divergence from objective loss

If we just use the reconstruction error as the loss term, i. e. equation (2). We should expect the model to perform well on reconstruction, but be unable to generate a good approximation of the original sample. In layman terms our VAE would be good at reconstructing output, but bad at generating new images.

$$\Lambda = E_{q_\phi(z|x)}[\log p_\theta(x|z)] \quad (3)$$

Effectively, we can see in the generated samples by this version with just the log likelihood, that our model is as good at reconstructing input, but way worse at generating images.

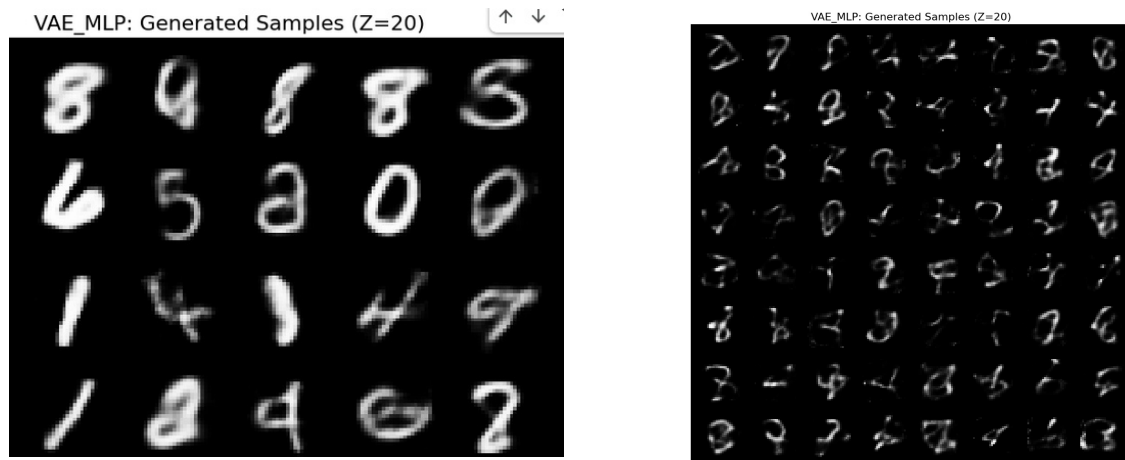


FIG 1: Generated images with ELBO vs BCE loss only.

Conclusion

We have summarized very shortly different kinds of autoencoders with emphasis on principles and higher order view rather than details of implementation. See the associated notebook for visualizations and step by step implementation.