Generative Models: An introduction

MUSTAFA HAJIJ

Generative Models

A generative model is a mathematical framework that learns the underlying distribution of a given dataset and then generates new samples that are similar to the original data.

More formally, let's consider a dataset of samples $\mathrm{D} = \{x_1, x_2, \ldots, x_n\}$, where each x_i is a data point. The goal of a generative model is to estimate the underlying probability distribution $p_{data}(x)$ from which the samples in X are drawn.

Generative Models

Explicitly, we are usually given a dataset $D = \{xi \in R^d\}^n{}_{i=1}$, where each x_i is i.i.d sampled from an unknown probability distribution $p_{data} : R^d \rightarrow R$. Within this setting, we are interested in estimating the distribution p_{data} by learning a parameterized density function $p_\theta : R^d \rightarrow R$, where θ is the parameter of p_θ , such that $p_\theta \approx p_{data}$.

The question is how to model the distribution p_θ ?

Text2Image Diffusion Models

User input:

An astronaut riding a horse

source

Text2Image Diffusion Models

User input:

A perfect Italian meal

source

Text2Image Diffusion Models

User input:

泰迪熊穿着戏服，站在太和殿前唱京剧

A teddy bear, wearing a costume, is standing in front of the Hall of Supreme Harmony and singing Beijing opera

source

Dalle3

A minimap diorama of a cafe adorned with indoor plants. Wooden beams crisscross above, and a cold brew station stands out with tiny bottles and glasses

source

Conditional Variational AutoEncoders

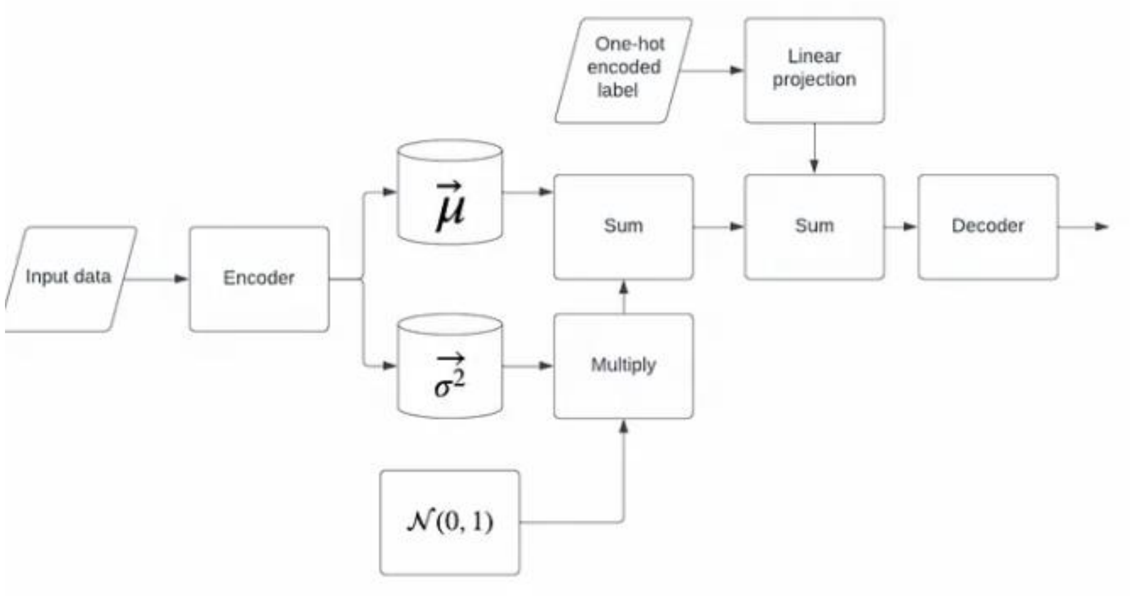

Image ref

Conditional Variational AutoEncoders

```python
class ConditionalVAE(VAE):
    # VAE implementation from the article linked above
    def __init__(self, num_classes):
        super().__init__()
        # Add a linear layer for the class label
        self.label_projector = nn.Sequential(
            nn.Linear(num_classes, self.num_hidden),
            nn.ReLU(),
        )

    def condition_on_label(self, z, y):
        projected_label = self.label_projector(y.float())
        return z + projected_label

    def forward(self, x, y):
        # Pass the input through the encoder
        encoded = self.encoder(x)
        # Compute the mean and log variance vectors
        mu = self.mu(encoded)
        log_var = self.log_var(encoded)
        # Reparameterize the latent variable
        z = self.reparameterize(mu, log_var)
        # Pass the latent variable through the decoder
        decoded = self.decoder(self.condition_on_label(z, y))
        # Return the encoded output, decoded output, mean, and log variance
        return encoded, decoded, mu, log_var

    def sample(self, num_samples, y):
        with torch.no_grad():
            # Generate random noise
            z = torch.randn(num_samples, self.num_hidden).to(device)
            # Pass the noise through the decoder to generate samples
            samples = self.decoder(self.condition_on_label(z, y))
        # Return the generated samples
        return samples
```

Image ref