Deep Learning-Variational Autoencoders

Variational Autoencoders (VAEs) are a type of generative model in deep learning that can learn to represent complex probability distributions over data. Here's an overview, along with examples:

Motivation VAEs aim to solve the following problems:

1. **Dimensionality reduction**: VAEs can compress high-dimensional data into lower-dimensional representations while retaining most of the information.
2. **Generative modeling**: VAEs can generate new samples from the input data distribution.

Architecture

A VAE consists of two main components:

1. **Encoder (E)**: Maps input data `x` to a latent representation `z`.
2. **Decoder (D)**: Maps the latent representation `z` back to the original input space.

The encoder and decoder are both neural networks, typically implemented using a Variational Autoencoder architecture:

Variational Autoencoder Architecture

- Encoder (`E(x)`): `x → z`
  - Input layer (e.g., 784 dimensions for MNIST)
  - Hidden layers (e.g., multiple fully connected layers with ReLU activation)
  - Output layer: latent space (mean and log variance of the normal distribution) `μ` and `σ²`
- Decoder (`D(z)`): `z → x`
  - Input layer (latent space, e.g., 2 dimensions for a simple example)
  - Hidden layers (e.g., multiple fully connected layers with ReLU activation)
  - Output layer: reconstructed input data

Objective Function The VAE is trained to maximize the Evidence Lower Bound (ELBO) of the log likelihood of the data. The ELBO can be written as:

```
ELBO = E[log p(x|z)] - KL[q(z|x)||p(z)]
```

where:

- `E[log p(x|z)]`: reconstruction term, measures how well the VAE can reconstruct the input
- `KL[q(z|x)||p(z)]`: Kullback-Leibler divergence between the approximate posterior distribution `q(z|x)` and the prior distribution `p(z)`, encourages the VAE to learn a meaningful representation

Training

To train a VAE, we typically use stochastic gradient descent (SGD) with the following loss function:

```
Loss = -ELBO
```

The VAE is trained by minimizing this loss function.

Example

Suppose we have a dataset of 784x784 images from MNIST. We can implement a simple VAE using PyTorch:

```
 import torch
import torch.nn as nn

class Encoder(nn.Module):
    def __init__(self, input_dim=784, hidden_dim=256, latent_dim=2):
        super(Encoder, self).__init__()
        self.fc1 = nn.Linear(input_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, latent_dim*2)  # mean and log variance

    def forward(self, x):
        x = torch.relu(self.fc1(x))
        z_mean_logvar = self.fc2(x)
        return z_mean_logvar

class Decoder(nn.Module):
    def __init__(self, latent_dim=2, hidden_dim=256, output_dim=784):
        super(Decoder, self).__init__()
        self.fc1 = nn.Linear(latent_dim, hidden_dim)
        self.fc2 = nn.Linear(hidden_dim, output_dim)

    def forward(self, z):
        x = torch.relu(self.fc1(z))
        x = torch.sigmoid(self.fc2(x))
        return x

# Initialize the VAE
vae = VariationalAutoencoder(Encoder, Decoder)

# Train the VAE using SGD and ELBO as loss function
optimizer = torch.optim.Adam(vae.parameters(), lr=0.001)
for epoch in range(100):
    # Forward pass
    z_mean_logvar = vae.encoder(x)
    x_reconstructed = vae.decoder(z_mean_logvar)

    # Compute the ELBO
    elbo = -vae.loss_function(x, z_mean_logvar)

    # Backward pass and update parameters
    optimizer.zero_grad()
    elbo.backward()
    optimizer.step()

# Example use case: generate new samples from the learned distribution
new_samples = vae.decoder(z_mean_logvar)
```

This example illustrates a basic VAE architecture for dimensionality reduction and generative modeling.