Variational\_autoencoders.ipynb:

## ## FC-VAE Encoder

Now lets start building our fully-connected VAE network. We'll start with the encoder, which will take our images as input (after flattening C,H,W to D shape) and pass them through a three Linear+ReLU layers. We'll use this hidden dimension representation to predict both the posterior mu and posterior log-variance using two separate linear layers (both shape (N,Z)).

Note that we are calling this the 'logvar' layer because we'll use the log-variance (instead of variance or standard deviation) to stabilize training. This will specifically matter more when you compute reparametrization and the loss function later.

\*Define an `encoder`, `hidden\_dim` (H), `mu\_layer`, and `logvar\_layer` in the initialization of the VAE class in `vae.py`. Use nn.Sequential to define the encoder, and separate Linear layers for the mu and logvar layers. In all of these layers, H will be a hidden dimension you set and will be the same across all encoder and decoder layers. Architecture for the encoder is described below:\*

\* `Flatten` (Hint: nn.Flatten)

\* Fully connected layer with input size 784 (`input\_size`) and output size H

\* `ReLU`

\* Fully connected layer with input\_size H and output size H

\* `ReLU`

\* Fully connected layer with input\_size H and output size H

\* `ReLU`

* The Encoder is defined as an nn.Sequential module containing the layers specified in the architecture.
* nn.Flatten() is used to flatten the input images.
* Three fully connected layers with ReLU activation functions are used to transform the flattened input into a hidden representation (hidden\_dim).
* Separate linear layers (mu\_layer and logvar\_layer) are defined to predict the posterior mean (mu) and posterior log-variance (logvar), respectively, from the hidden representation.
* The hidden\_dim is stored as an attribute because it's needed later when defining the decoder.

## ## FC-VAE Decoder

We'll now define the decoder, which will take the latent space representation and generate a reconstructed image. The architecture is as follows:

\* Fully connected layer with input size as the latent size (Z) and output size H

\* `ReLU`

\* Fully connected layer with input\_size H and output size H

\* `ReLU`

\* Fully connected layer with input\_size H and output size H

\* `ReLU`

\* Fully connected layer with input\_size H and output size 784 (`input\_size`)

\* `Sigmoid`

\* `Unflatten` (nn.Unflatten)

\*Define a `decoder` in the initialization of the VAE class in `vae.py`. Like the encoding step, use `nn.Sequential`\*

* The Decoder is defined as an nn.Sequential module containing the layers specified in the architecture.
* Three fully connected layers with ReLU activation functions are used to transform the latent space representation into a hidden representation (hidden\_dim).
* A final fully connected layer with a sigmoid activation function is used to generate the reconstructed image.
* nn.Unflatten(1, (1, 28, 28)) is used to reshape the output back to the original image shape (1, 28, 28). Adjust the shape according to the dimensions of your input images if needed.

## ## Reparametrization

Now we'll apply a reparametrization trick in order to estimate the posterior 𝑧 during our forward pass, given the 𝜇 and 𝜎2 estimated by the encoder. A simple way to do this could be to simply generate a normal distribution centered at our 𝜇 and having a std corresponding to our 𝜎2 . However, we would have to backpropogate through this random sampling that is not differentiable. Instead, we sample initial random data 𝜖 from a fixed distrubtion, and compute 𝑧 as a function of ( 𝜖 , 𝜎2 , 𝜇 ). Specifically:

𝑧=𝜇+𝜎𝜖

We can easily find the partial derivatives w.r.t 𝜇 and 𝜎2 and backpropagate through 𝑧 . If 𝜖=(0,1) , then it's easy to verify that the result of our forward pass calculation will be a distribution centered at 𝜇 with variance 𝜎2 .

Implement reparametrization in vae.py and verify your mean and std error are at or less than 1e-4.

mu is the mean of the Gaussian distribution.

logvar is the log-variance of the Gaussian distribution.

epsilon is sampled from a standard normal distribution with the same shape as mu.

z is computed using the reparametrization trick: z = exp(0.5 \* logvar) \* epsilon + mu.

# Implementation

* VAE Class:
  + The VAE class inherits from nn.Module.
  + It takes input\_size and latent\_size as parameters, where input\_size represents the size of the input images, and latent\_size represents the size of the latent space.
  + The class initializes parameters for encoder, decoder, mu (mean), and logvar (log variance).
  + In the \_\_init\_\_ method, the encoder architecture is defined using fully connected layers followed by ReLU activations.
  + The mu\_layer and logvar\_layer are defined as linear layers to output the mean and log-variance of the posterior distribution over latent vectors.
  + The decoder architecture is also defined using fully connected layers followed by ReLU activations, ending with a Sigmoid activation to map values to [0, 1].
* Forward Pass:
  + The forward method performs the forward pass through the VAE model.
  + It takes a batch of input images x and passes them through the encoder to obtain posterior mean (mu) and log-variance (logvar).
  + It then reparametrizes to compute the latent vector z.
  + The latent vector z is passed through the decoder to reconstruct the input images (x\_hat).
  + The method returns the reconstructed images (x\_hat), the mean (mu), and the log-variance (logvar).
  + The input x is passed through the encoder to obtain the posterior mean (mu) and log-variance (logvar).
  + The reparametrization trick is applied to sample the latent variable z from the posterior distribution.
  + The latent variable z is then passed through the decoder to reconstruct the input x.
  + Finally, the reconstructed input x\_hat, along with mu and logvar, are returned from the forward pass.

In this loss\_function function, we first compute the reconstruction loss using binary cross-entropy between the reconstructed input x\_hat and the original input x. Then, we compute the Kullback-Leibler (KL) divergence between the estimated posterior distribution and a standard normal distribution. Finally, we sum the reconstruction loss and the KL divergence to obtain the total loss for the negative variational lower bound.