

WEEK-II

Aim:- Experiments using variational Auto encoder. To implement VAE for compressing and generating MNIST digit images using a probabilistic approach.

Objective:-

- 1.) To learn latent space representation of MNIST digits.
- 2.) To generate new images by sampling from the learned latent distribution.
- 3.) To compare the reconstructed and generated images with the original dataset.

Algorithm:-

- 1.) Load and normalize MNIST dataset.
- 2.) Define encoder \rightarrow outputs mean(μ) and variance(σ^2).
- 3.) Apply reparameterization trick to sample latent vector "z".
- 4.) Define decoder \rightarrow reconstruct image from "z".
- 5.) Define loss = reconstruction loss + KL divergence.
- 6.) Compile and train VAE model.
- 7.) Generate new digit samples by sampling from latent space.

Pseudocode:-

Load MNIST

Normalize data

Encoder : Input \rightarrow Dense (128, relu) \rightarrow Dense (64, relu)

Outputs : z_mean, z_log_var

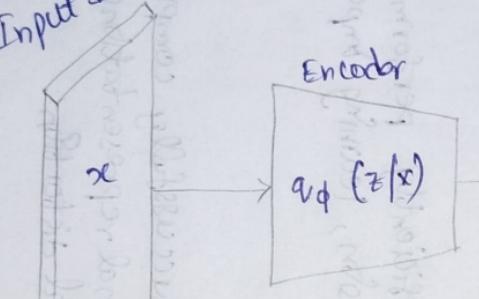
$z = z_{\text{mean}} + \exp(0.5 * z_{\log_var}) * \text{random_noise}$

Decoder :

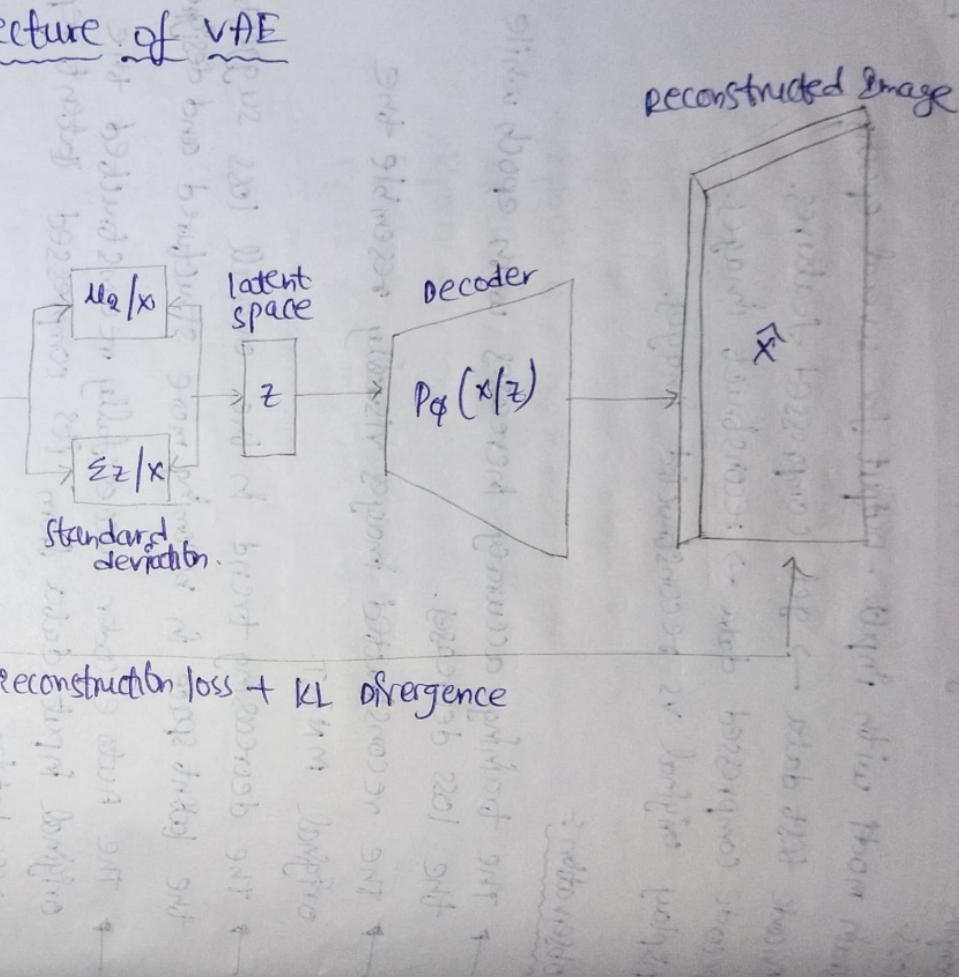
$z \rightarrow \text{dense}(64, \text{relu}) \rightarrow \text{dense}(784, \text{sigmoid})$

VAD = Encoder + Decoder

Input Image



Architecture of VAE



$\text{LOSS} = \text{reconstruction_loss} + \text{KL_divergence}$.

Compile (optimizer = 'adam'), loss = VAE-loss

train(x-train, y-train)

Generate new images by sampling "z"
Display results.

Observation :-

- Instead of mapping input to a fixed point, VAE encode input into a probability distribution in latent space.
- VAE's use the reparameterization trick to allow for back propagation through sampling process.
- The loss function of VAE includes a reconstruction loss and KL divergence to regularize space and encourage to follow a prior distribution.

Conclusion :-

VAE's combine autoencoding and probabilistic modeling enabling both compression and generation of new synthetic data.

Result :-

successfully implemented VAE (variational autoencoder) using MNIST dataset.

Output :-

Epoch 1/10 :- loss : 2.5003.6801

Epoch 2/10 :- loss : 21888.4032

Epoch 3/10 :- loss : 21232.1173

Epoch 4/10 :- loss : 20888.5366

Epoch 5/10 :- loss : 20658.487

Epoch 6/10 :- loss : 20486.1709

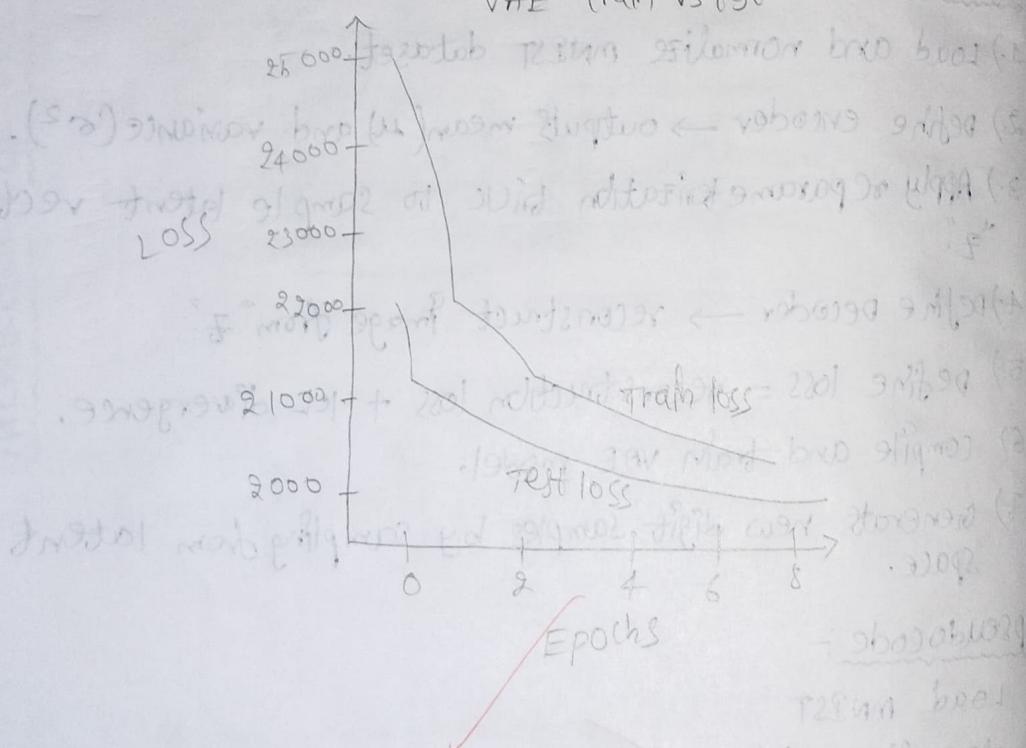
Epoch 7/10 :- loss : 20346.4544

Epoch 8/10 , loss : 20285.4947

Epoch 9/10 , loss : 20187.8976

Epoch 10/10 , loss : 20055.5366

VAE train vs test



```
import tensorflow as tf
from tensorflow.keras import layers, Model
import matplotlib.pyplot as plt
import numpy as np

(x_train, _), (x_test, _) = tf.keras.datasets.mnist.load_data()
x_train = x_train.astype("float32") / 255.0
x_test = x_test.astype("float32") / 255.0
x_train = x_train.reshape(-1, 784)
x_test = x_test.reshape(-1, 784)

latent_dim = 2
inputs = tf.keras.Input(shape=(784,), name="encoder_input")
x = layers.Dense(512, activation="relu")(inputs)
z_mean = layers.Dense(latent_dim, name="z_mean")(x)
z_log_var = layers.Dense(latent_dim, name="z_log_var")(x)

def sampling(args):
    z_mean, z_log_var = args
    epsilon = tf.random.normal(shape=tf.shape(z_mean))
    return z_mean + tf.exp(0.5 * z_log_var) * epsilon

z = layers.Lambda(sampling, name="z")([z_mean, z_log_var])
encoder = Model(inputs, [z_mean, z_log_var, z], name="encoder")
encoder.summary()

latent_inputs = tf.keras.Input(shape=(latent_dim,), name="z_sampling")
x = layers.Dense(512, activation="relu")(latent_inputs)
outputs = layers.Dense(784, activation="sigmoid")(x)
decoder = Model(latent_inputs, outputs, name="decoder")
decoder.summary()

class VAE(Model):
    def __init__(self, encoder, decoder, **kwargs):
        super(VAE, self).__init__(**kwargs)
        self.encoder = encoder
        self.decoder = decoder

    def call(self, inputs):
        z_mean, z_log_var, z = self.encoder(inputs)
        return self.decoder(z)

    def train_step(self, data):
        if isinstance(data, tuple):
            data = data[0]
        with tf.GradientTape() as tape:
            z_mean, z_log_var, z = self.encoder(data)
            reconstruction = self.decoder(z)
            # █ Fixed reconstruction loss
            reconstruction_loss = tf.reduce_mean(
                tf.keras.losses.binary_crossentropy(data, reconstruction) * 784
            )
            kl_loss = -0.5 * tf.reduce_mean(
                1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var)
            )
            total_loss = reconstruction_loss + kl_loss
        grads = tape.gradient(total_loss, self.trainable_variables)
        self.optimizer.apply_gradients(zip(grads, self.trainable_variables))
        return {"loss": total_loss}

def test_standalone():
    pass
```

```
    vae = vae(vae)
    z_mean, z_log_var, z = self.encoder(data)
    reconstruction = self.decoder(z)
    reconstruction_loss = tf.reduce_mean(
        tf.keras.losses.binary_crossentropy(data, reconstruction) * 784
    )
    kl_loss = -0.5 * tf.reduce_mean(
        1 + z_log_var - tf.square(z_mean) - tf.exp(z_log_var)
    )
    total_loss = reconstruction_loss + kl_loss
    return {"loss": total_loss}

# 3 Compile and Train
vae = VAE(encoder, decoder)
vae.compile(optimizer=tf.keras.optimizers.Adam())
history = vae.fit(
    x_train,
    x_train,
    epochs=10,
    batch_size=256,
    validation_data=(x_test, x_test),
)

# 4 Plot Training vs Validation Loss
plt.plot(history.history["loss"], label="Training Loss")
plt.plot(history.history["val_loss"], label="Validation Loss")
plt.xlabel("Epochs")
plt.ylabel("Loss")
plt.legend()
plt.title("VAE Training vs Validation Loss")
plt.show()

# 5 Reconstruct Test Images
decoded_imgs = vae.decoder(vae.encoder(x_test)[2]).numpy()

n = 10
plt.figure(figsize=(20, 4))
for i in range(n):
    # Original
    ax = plt.subplot(2, n, i + 1)
    plt.imshow(x_test[i].reshape(28, 28), cmap="gray")
    plt.axis("off")
    # Reconstructed
    ax = plt.subplot(2, n, i + 1 + n)
    plt.imshow(decoded_imgs[i].reshape(28, 28), cmap="gray")
    plt.axis("off")
plt.suptitle("Top: Original Images | Bottom: VAE Reconstructions")
plt.show()
```

Model: "encoder"

| Layer (type) | Output Shape | Param # | Connected to |
|----------------------------|--------------|---------|-------------------------------|
| encoder_input (InputLayer) | (None, 784) | 0 | - |
| dense_47 (Dense) | (None, 512) | 401,920 | encoder_input[0] |
| z_mean (Dense) | (None, 2) | 1,026 | dense_47[0][0] |
| z_log_var (Dense) | (None, 2) | 1,026 | dense_47[0][0] |
| z (Lambda) | (None, 2) | 0 | z_mean[0][0], z_log_var[0][0] |

Total params: 403,972 (1.54 MB)

Trainable params: 403,972 (1.54 MB)

Non-trainable params: 0 (0.00 B)

Model: "decoder"

| Layer (type) | Output Shape | Param # |
|-------------------------|--------------|---------|
| z_sampling (InputLayer) | (None, 3) | 0 |
| dense_48 (Dense) | (None, 512) | 1,536 |
| dense_49 (Dense) | (None, 784) | 402,192 |

Total params: 403,728 (1.54 MB)

Trainable params: 403,728 (1.54 MB)

Non-trainable params: 0 (0.00 B)

Epoch 1/10

235/235 - 14s 59ms/step - loss: 219.1661 - val_loss: 0.0000e+00

Epoch 2/10

235/235 - 9s 37ms/step - loss: 171.5112 - val_loss: 0.0000e+00

Epoch 3/10

235/235 - 10s 42ms/step - loss: 164.4683 - val_loss: 0.0000e+00

Epoch 4/10

235/235 - 10s 42ms/step - loss: 161.6138 - val_loss: 0.0000e+00

Epoch 5/10

235/235 - 10s 40ms/step - loss: 159.8511 - val_loss: 0.0000e+00

Epoch 6/10

235/235 - 9s 39ms/step - loss: 158.4258 - val_loss: 0.0000e+00

Epoch 7/10

235/235 - 11s 44ms/step - loss: 157.1982 - val_loss: 0.0000e+00

Epoch 8/10

235/235 - 10s 43ms/step - loss: 156.0268 - val_loss: 0.0000e+00

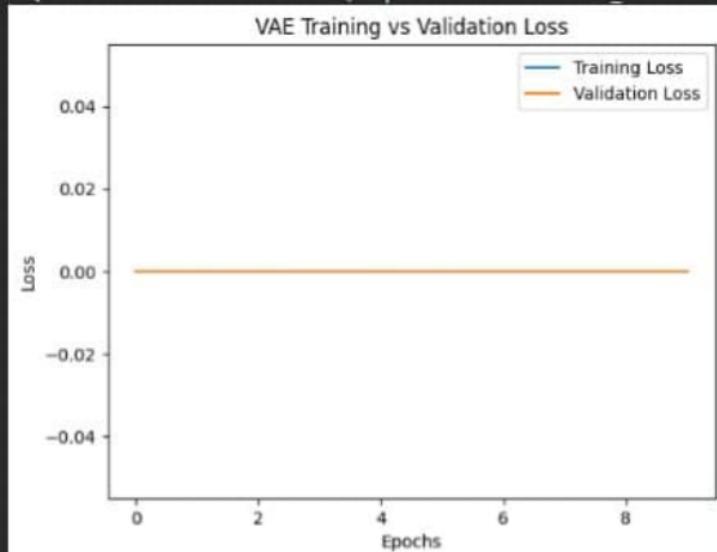
Epoch 9/10

235/235 - 9s 40ms/step - loss: 154.9958 - val_loss: 0.0000e+00

Epoch 10/10

235/235 - 9s 39ms/step - loss: 154.0985 - val_loss: 0.0000e+00

VAE Training vs Validation Loss



Top: Original Images | Bottom: VAE Reconstructions

