CS3232 Fundamental of Deep learning

Parts of the contents are from deeplearning.ai, jalFaizy and other resources on the Internet

Variational Autoencoders (VAE)

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Variational Autoencoders (VAE)

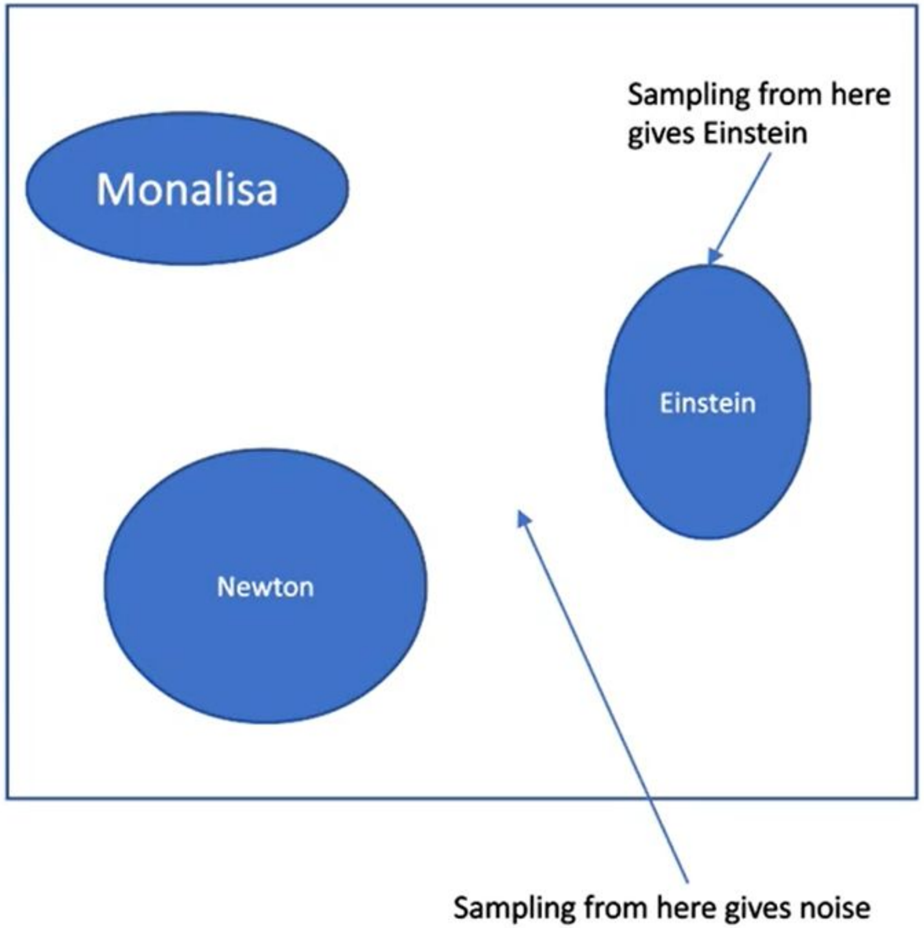
RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

- Variational Autoencoders (VAEs) are a type of autoencoder designed for unsupervised learning of latent variables.
- VAEs are used to generate new data that is similar to the training data.
- Unlike traditional autoencoders, which learn to compress the data into a latent space and then reconstruct it, VAEs learn the probability distribution of the data and can generate new samples by sampling from this distribution.

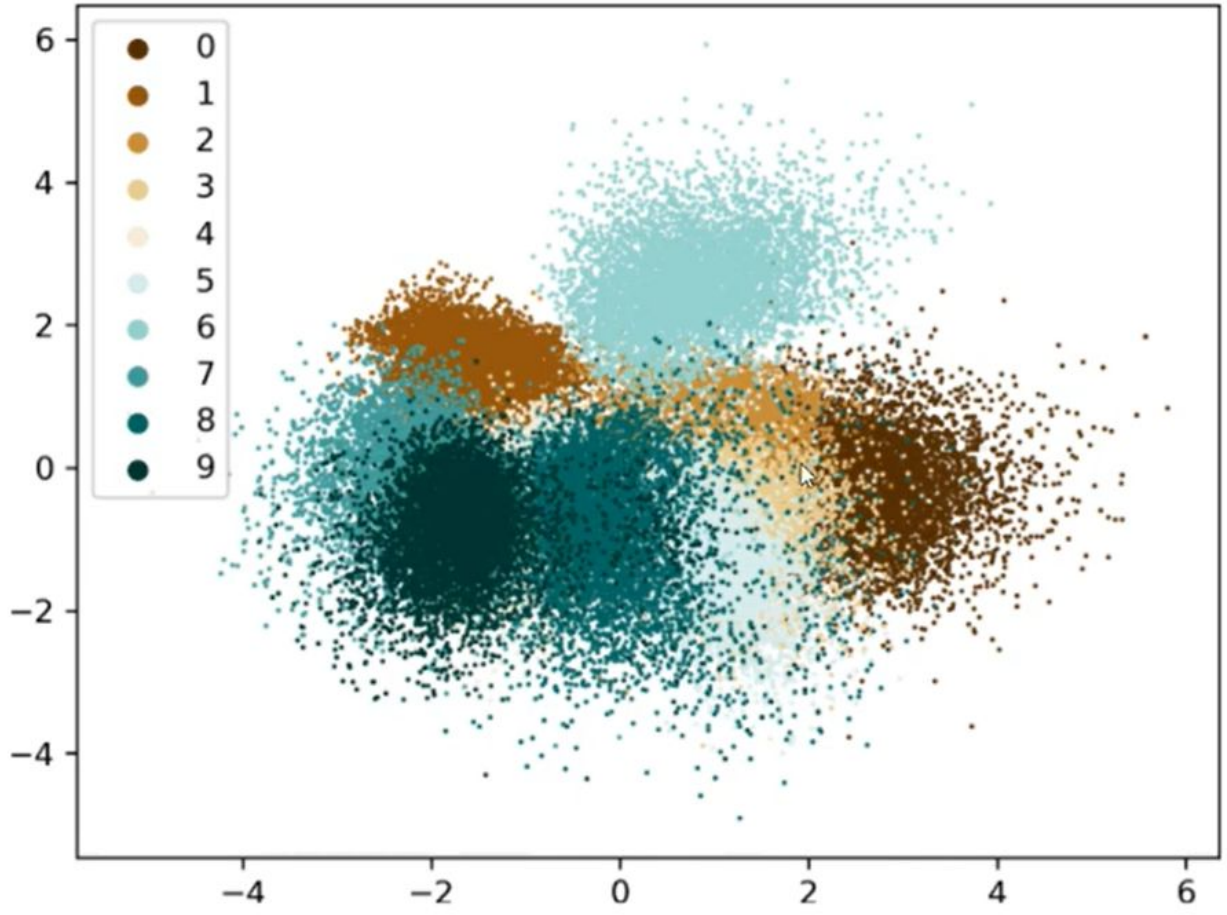
Variational Autoencoder

Sampling

Latent distribution for Autoencoder

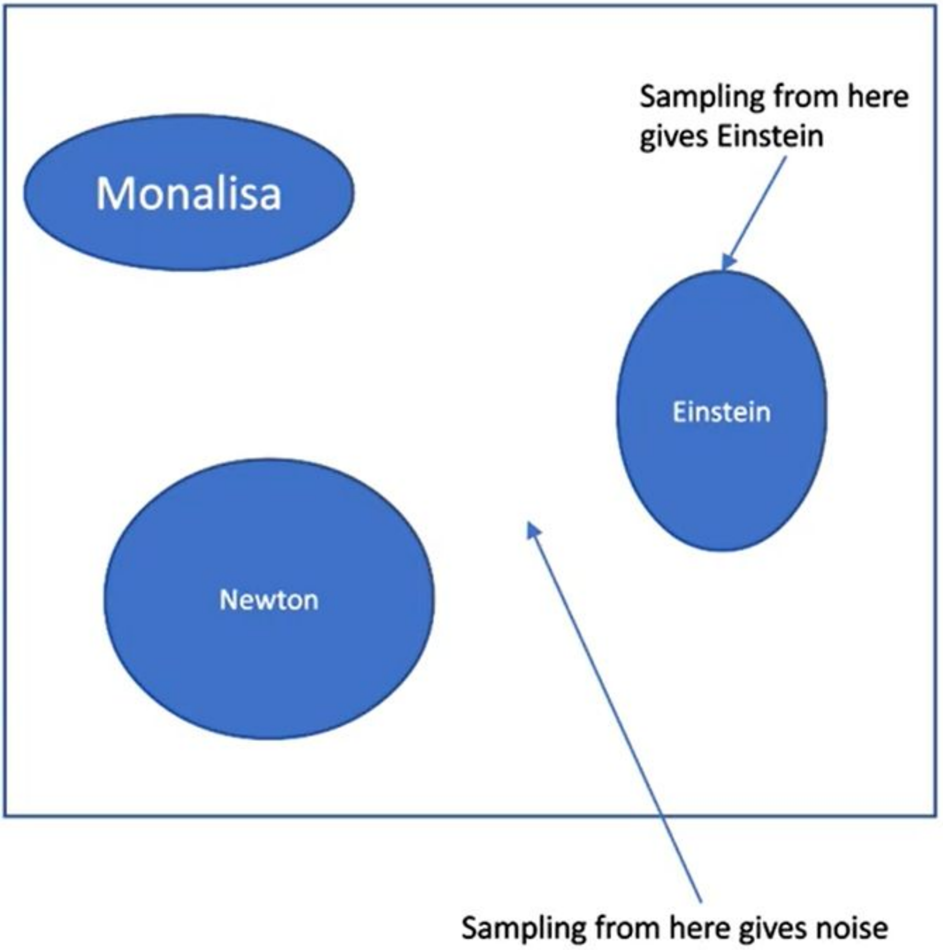


What if we know how to pick appropriate latent vectors?

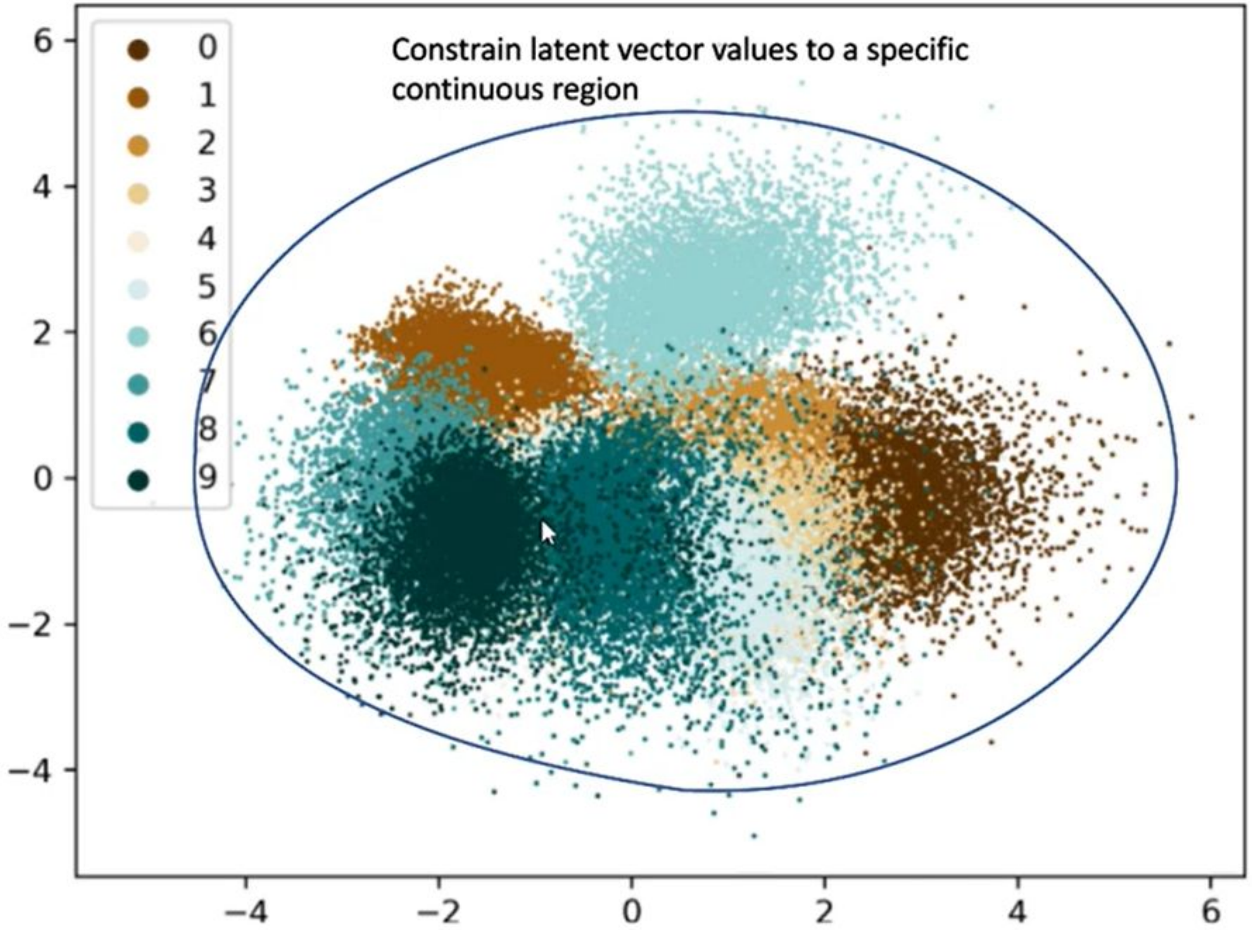


Variational Autoencoder
Sampling

Latent distribution for Autoencoder

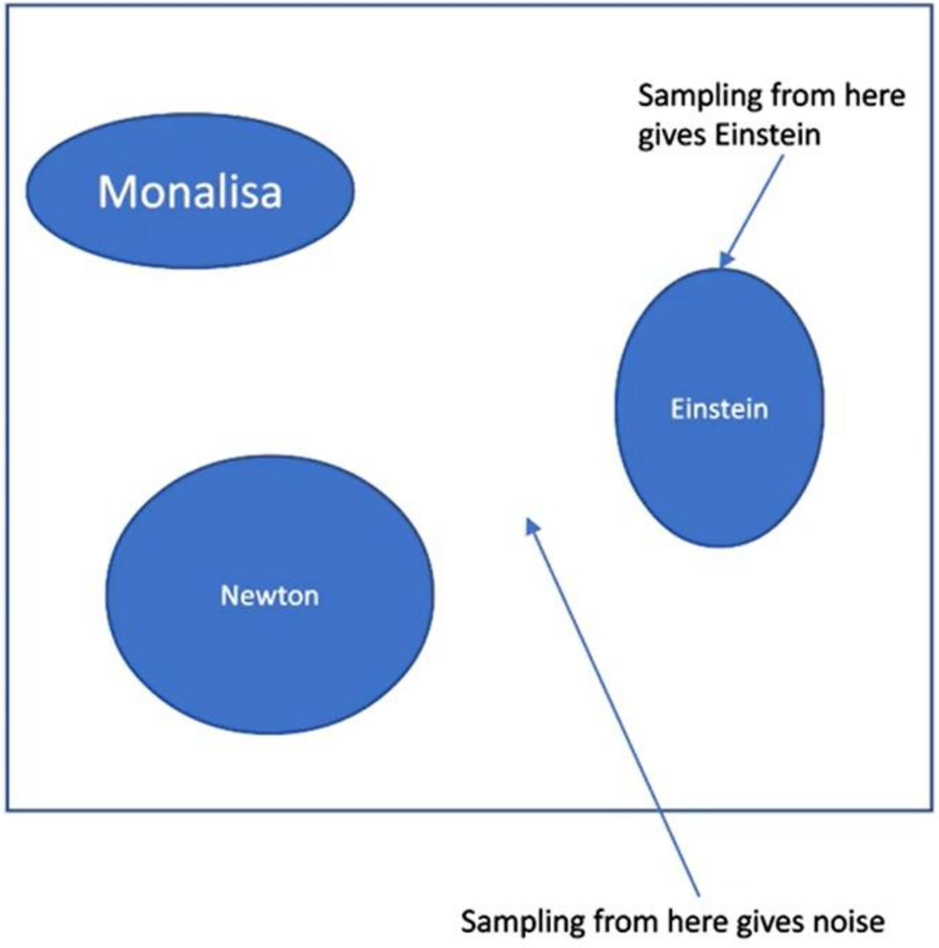


What if we know how to pick appropriate latent vectors?

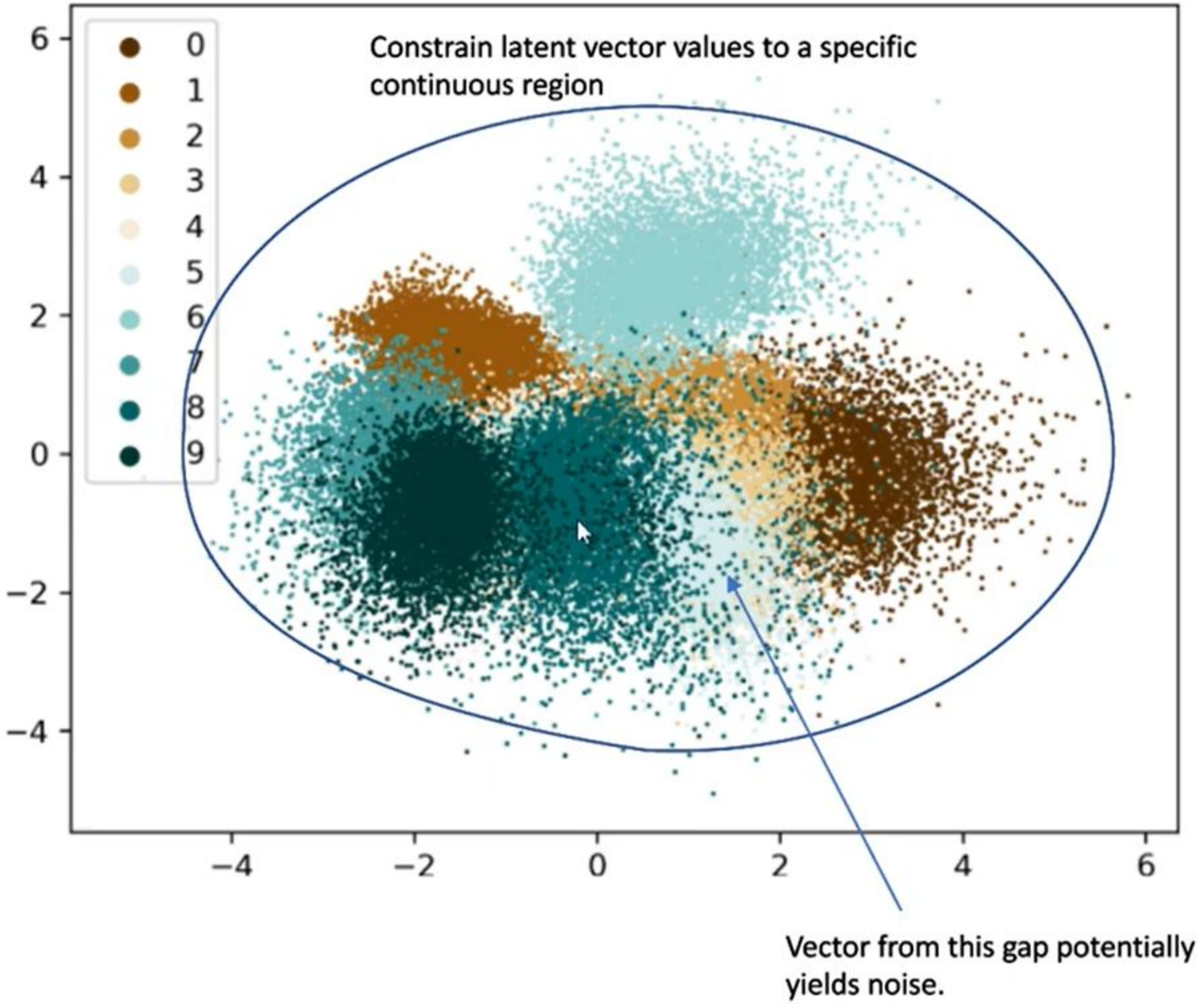


Variational Autoencoder
Sampling

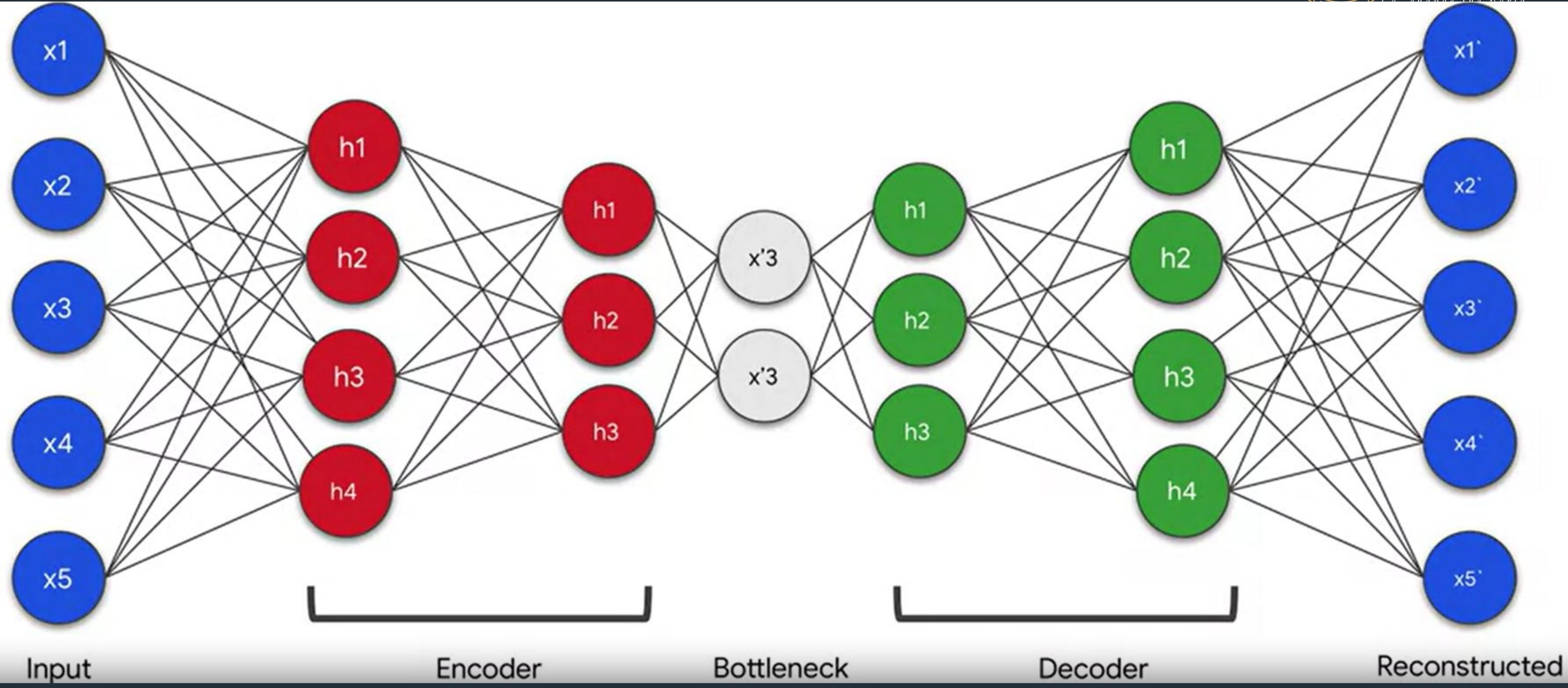
Latent distribution for Autoencoder



What if we know how to pick appropriate latent vectors?



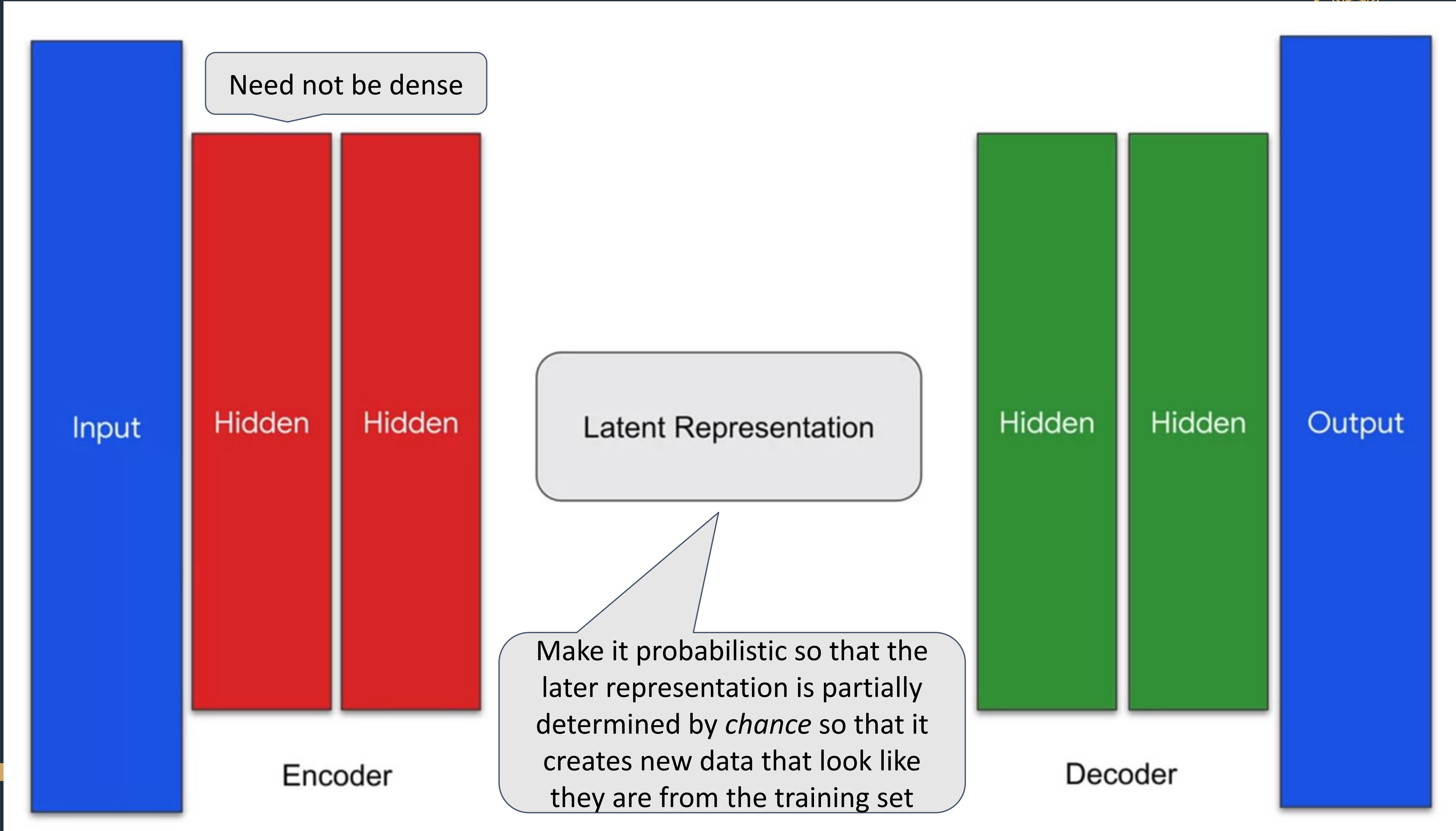
Recall autoencoders



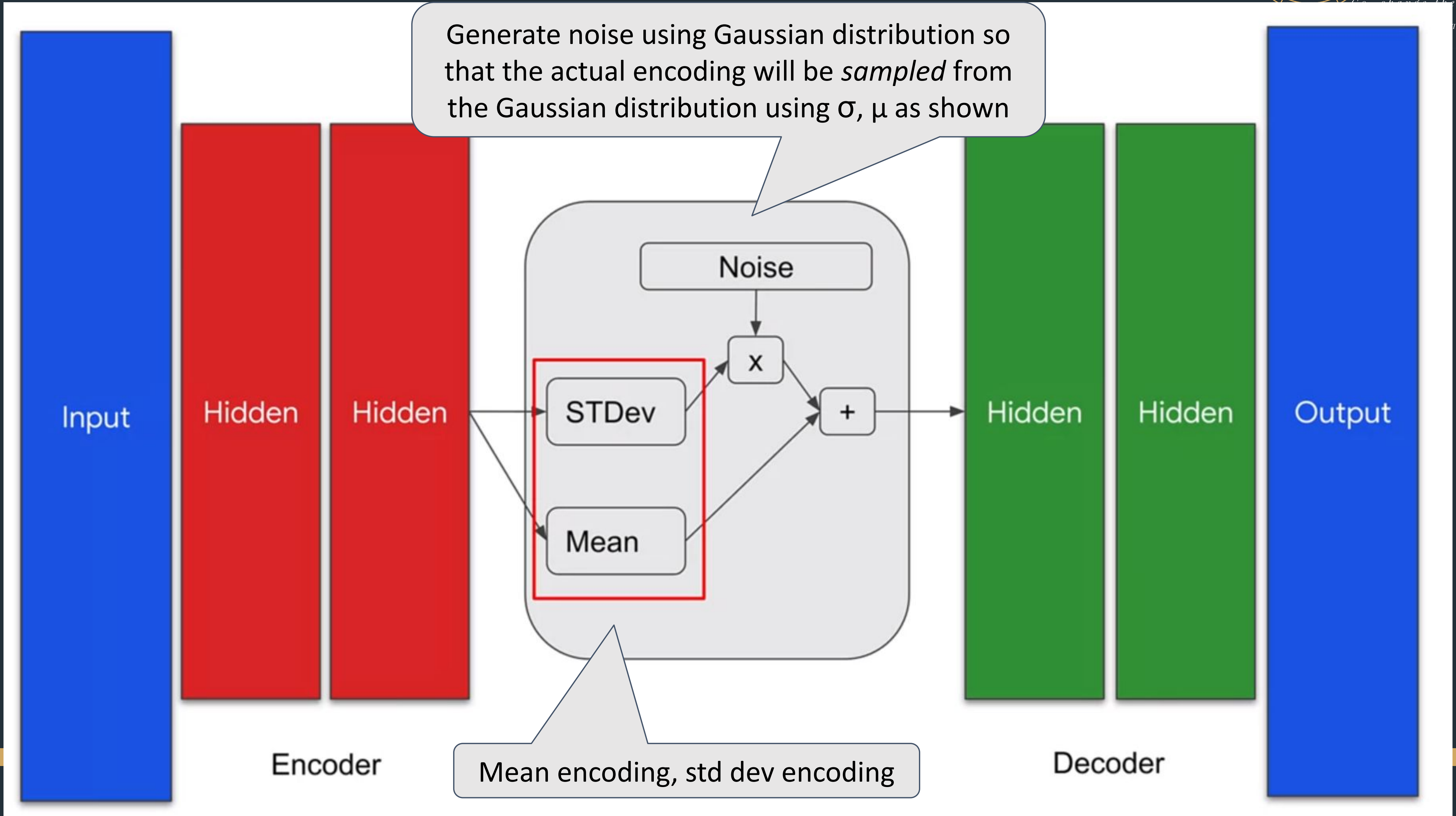
Variational Autoencoders (VAE)

- Instead of mapping input to a fixed latent vector, we map it to a distribution.
- Force latent variables to be normally distributed.
- Instead of passing the encoder output to the decoder, use mean and standard deviation describing the distribution.
- Quantify the distance between learned distribution and standard normal distribution using Kullback–Leibler (KL) divergence.

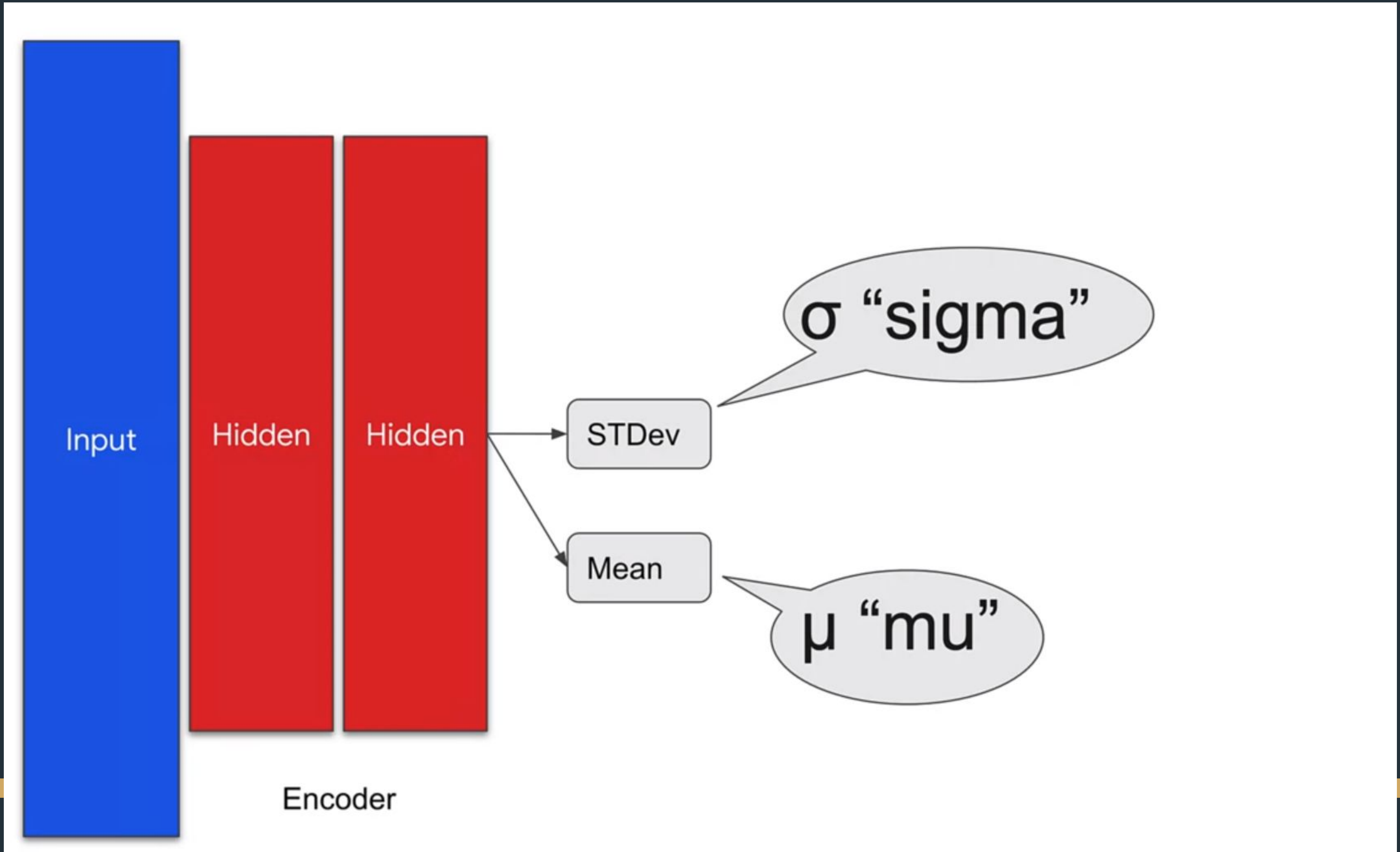
Rewrite it for variational autoencoders (VAEs)



Variational Autoencoders (VAE)



Variational Autoencoders (VAE)



Variational Autoencoders (VAE)

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Probability Distribution

Gaussian probability density function or **Normal Distribution.**

Normal Distribution is controlled by:

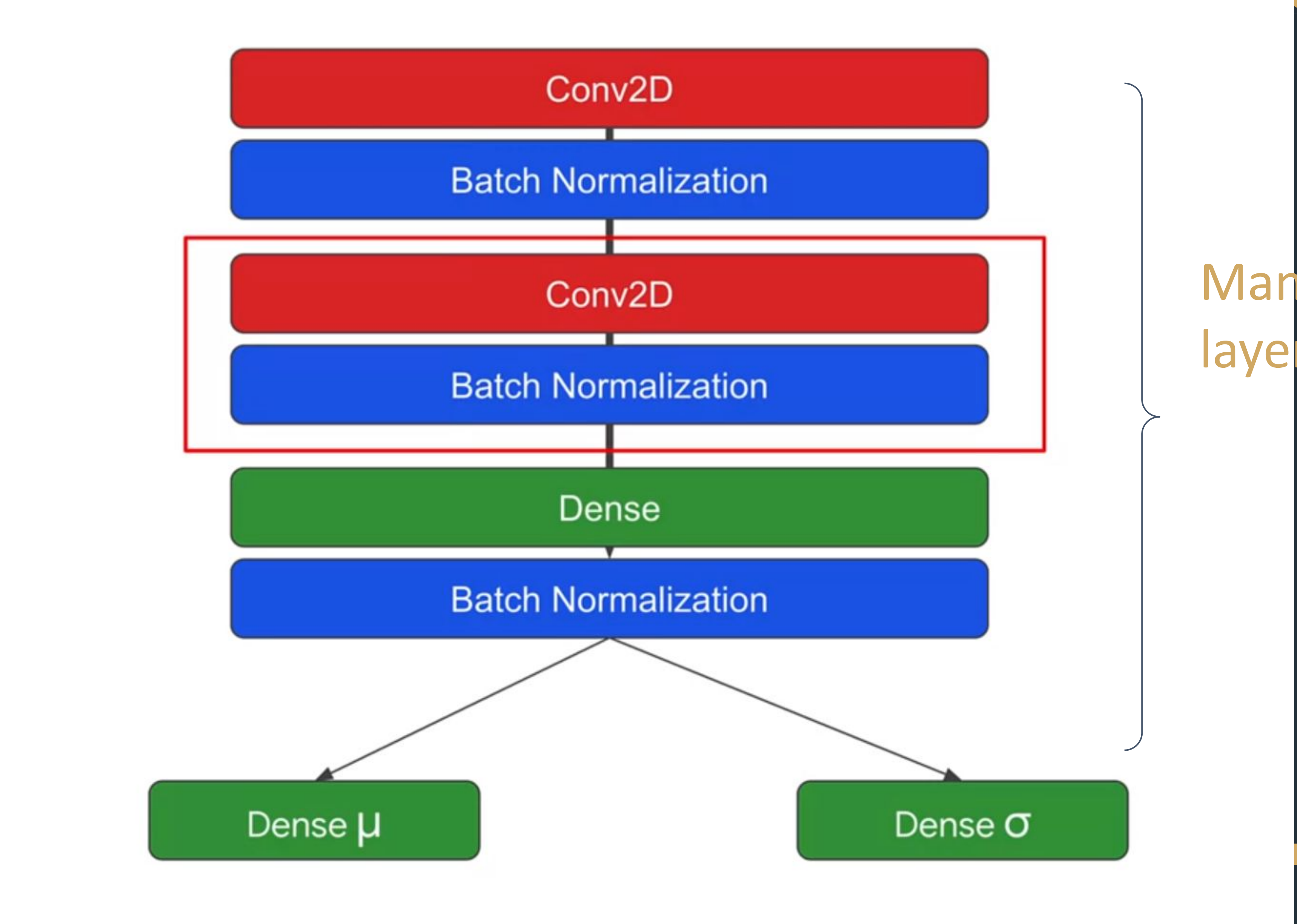
- **μ “mean”**
- **σ “standard deviation”**

Learnt

$$N(\mu, \sigma)$$

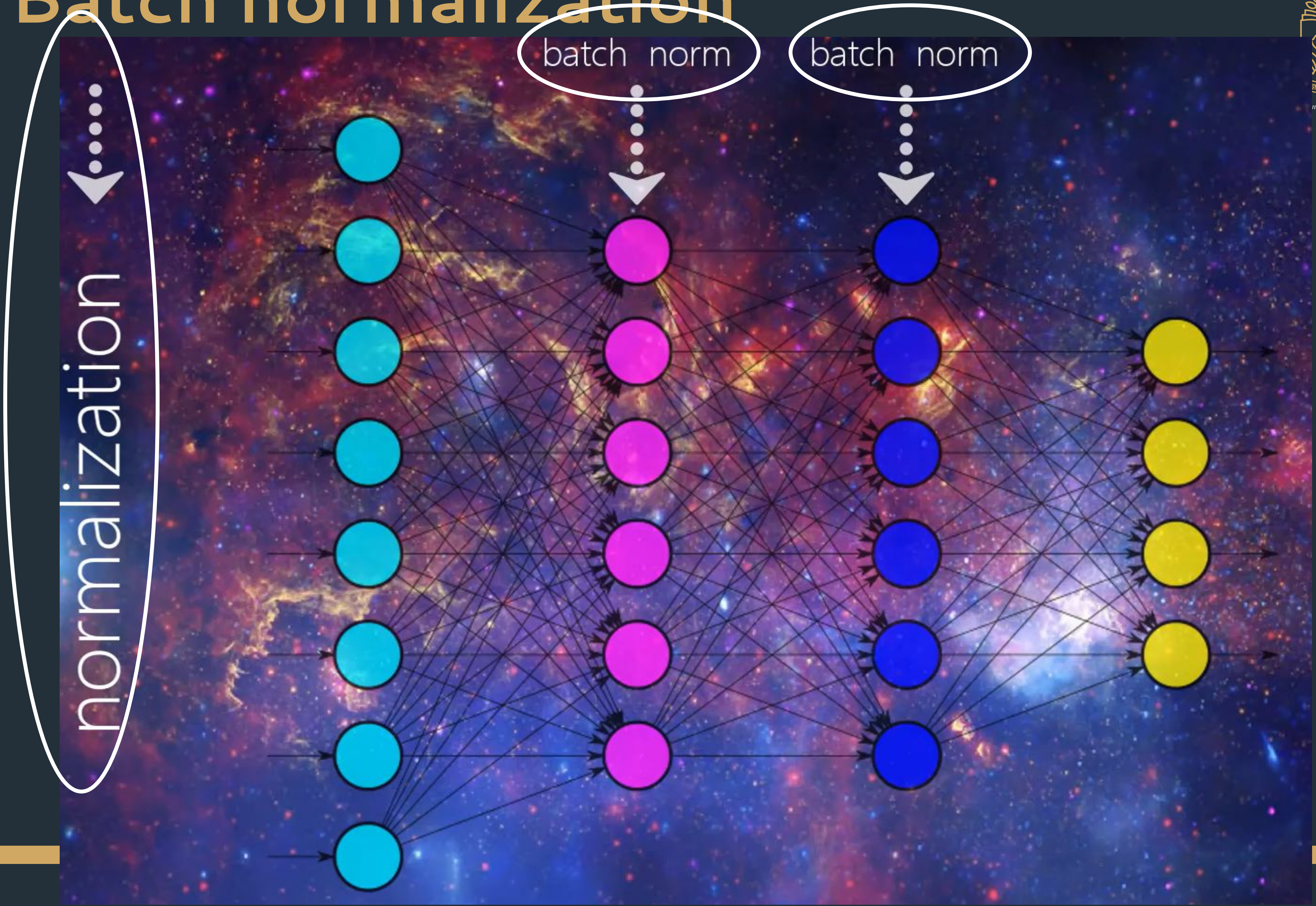
Output is generated using the normal distribution

Variational Autoencoders (VAE)

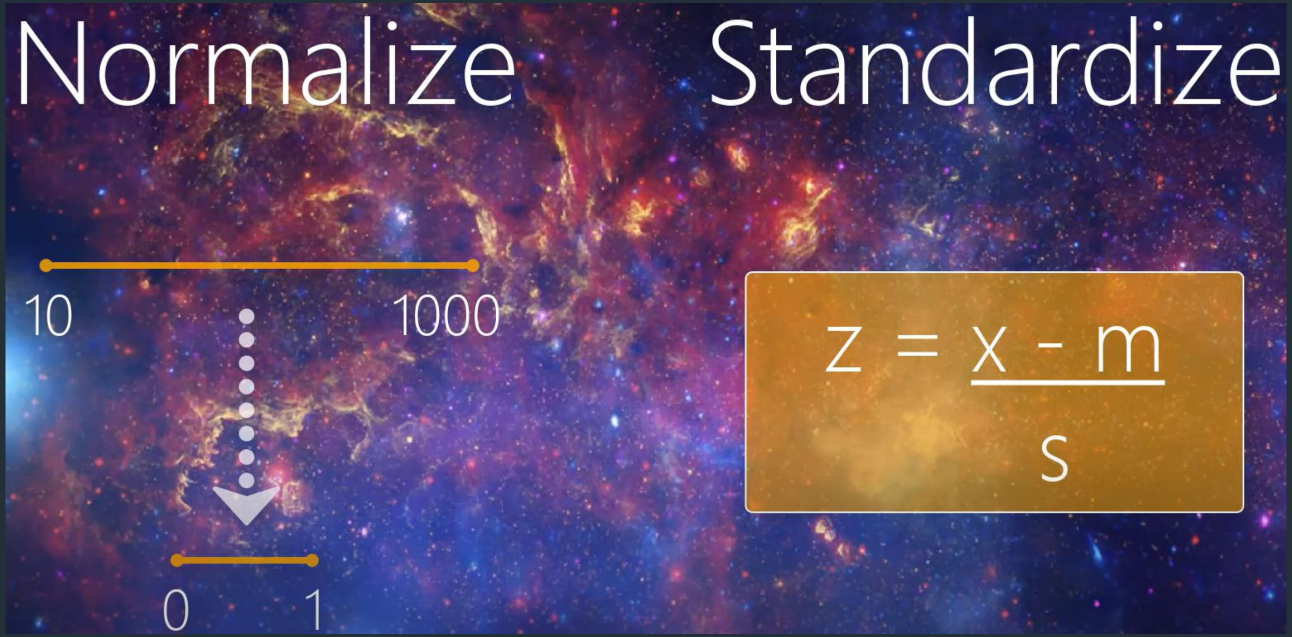


Many hidden layers

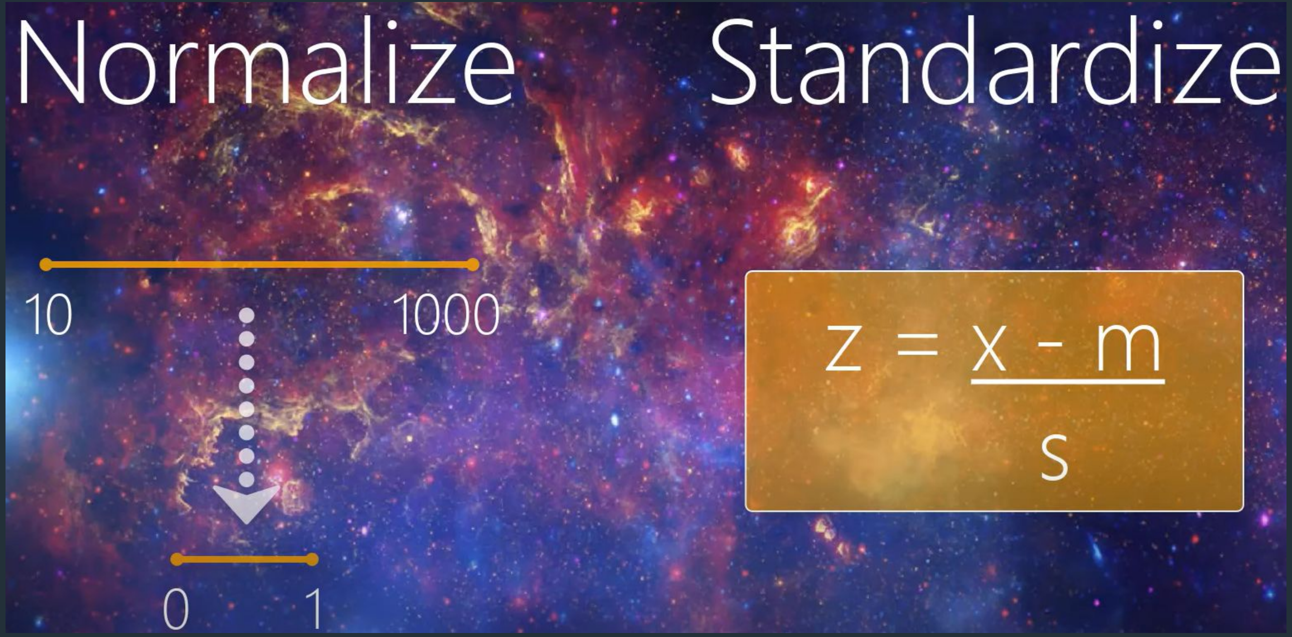
Batch normalization



Batch normalization

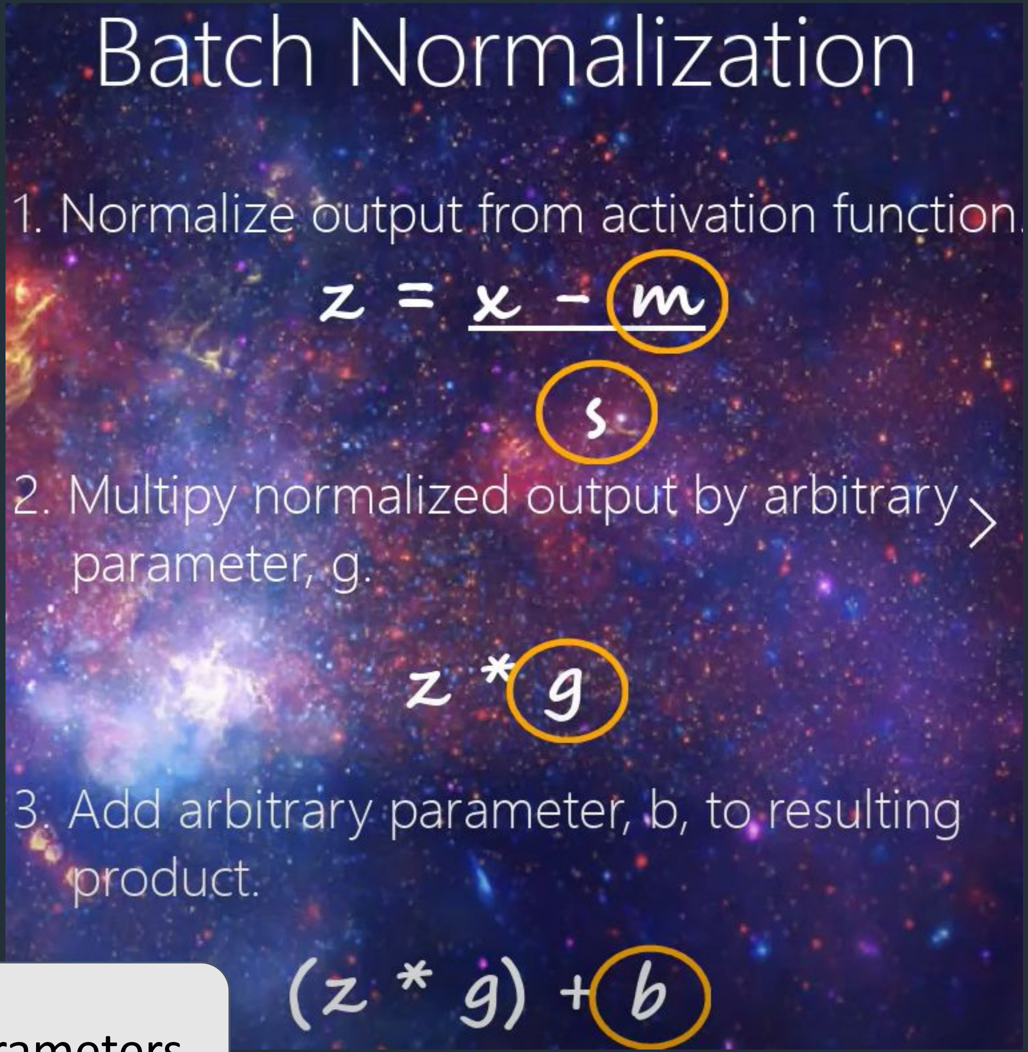


Batch normalization



Batch norm

To make the training faster and more stable through normalization of the layers' inputs by re-centering and re-scaling

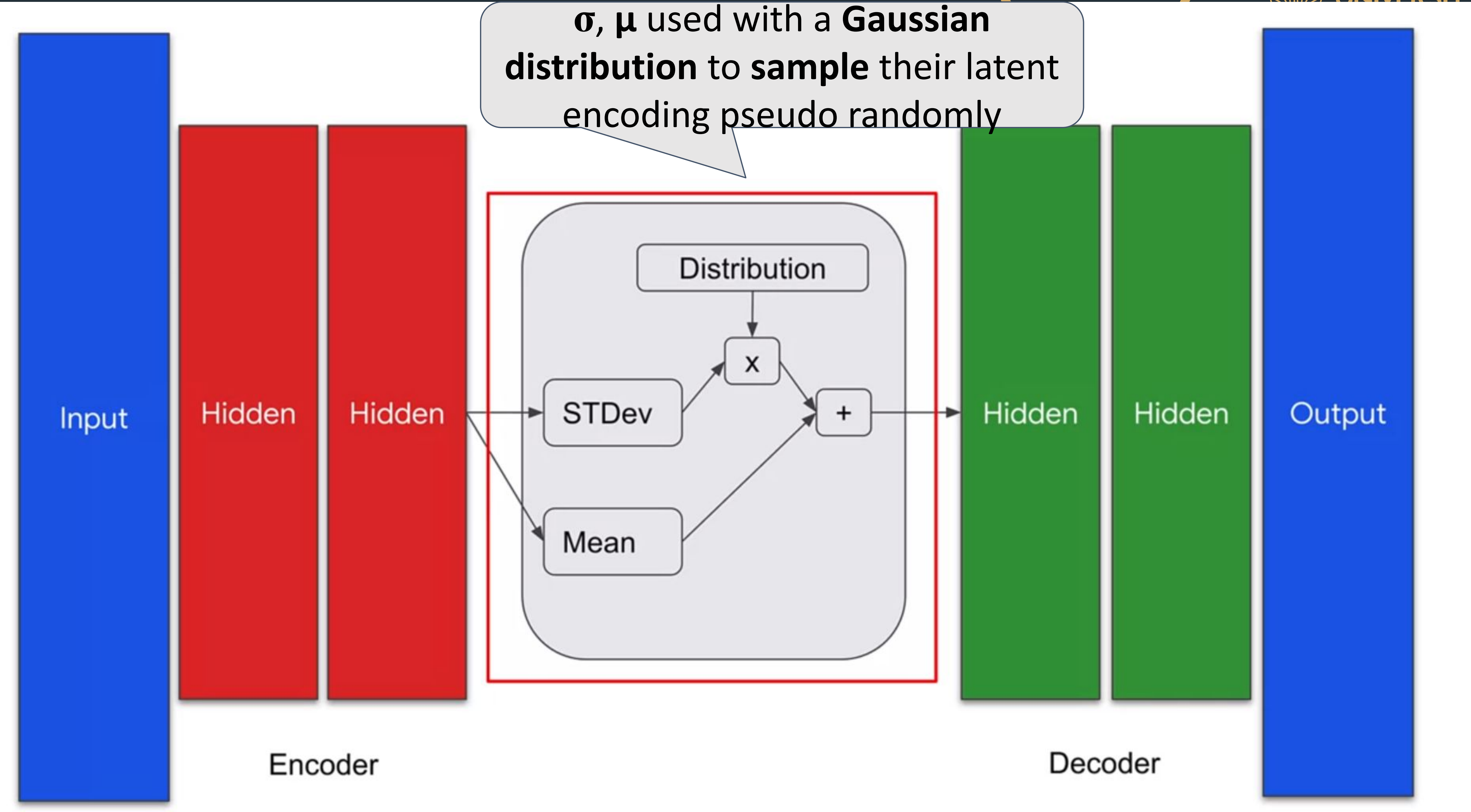


Trainable parameters

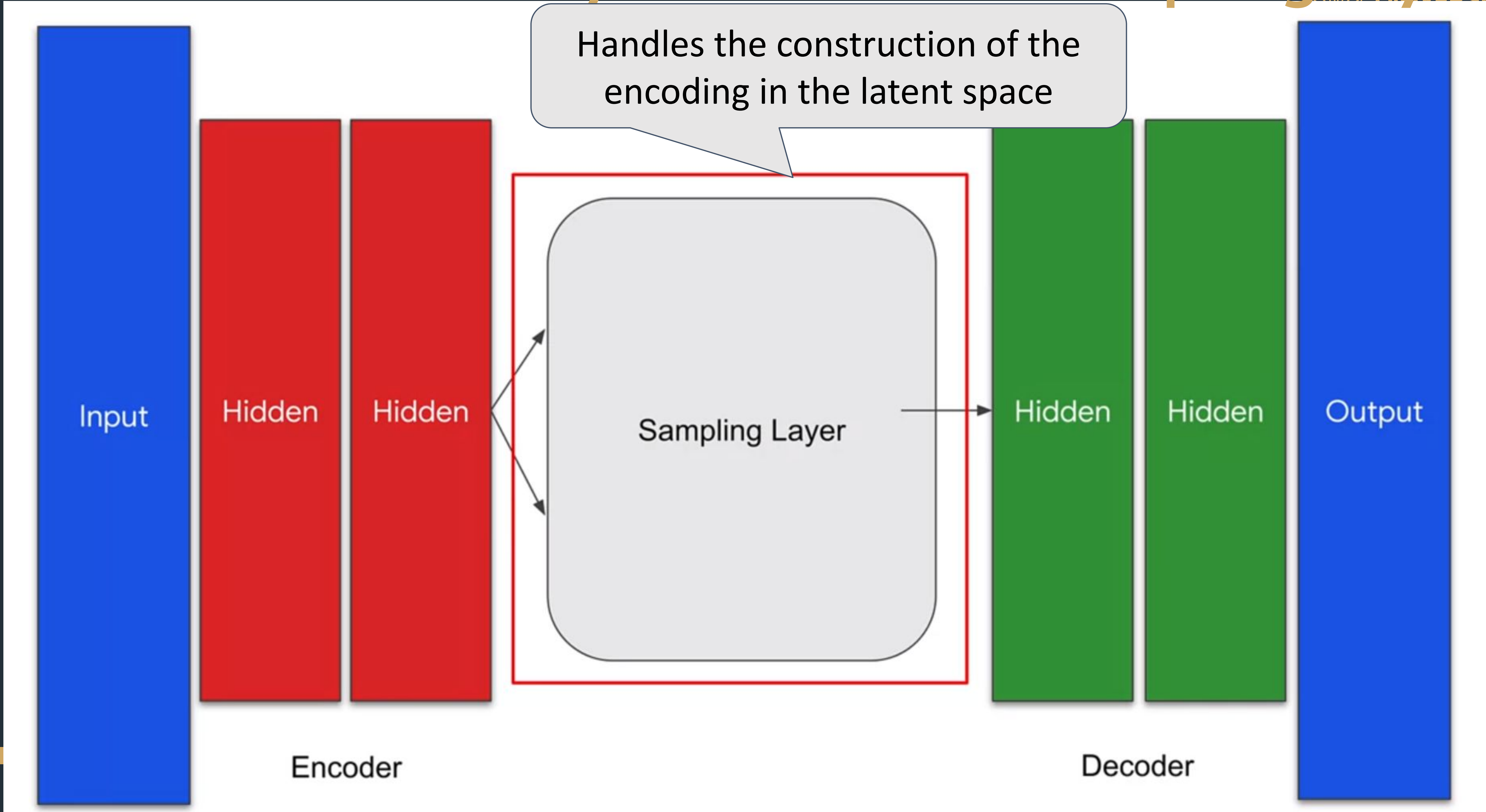
```python
def encoder_layers(inputs, latent_dim):
  x = tf.keras.layers.Conv2D(filters=32, kernel_size=3, strides=2,
                             padding="same", activation='relu',
                             name="encode_conv1")(inputs)
  x = tf.keras.layers.BatchNormalization()(x)

  x = tf.keras.layers.Conv2D(filters=64, kernel_size=3, strides=2,
                             padding='same', activation='relu',
                             name="encode_conv2")(x)
  batch_2 = tf.keras.layers.BatchNormalization()(x)

  x = tf.keras.layers.Flatten(name="encode_flatten")(batch_2)
  x = tf.keras.layers.Dense(20, activation='relu', name="encode_dense")(x)
  x = tf.keras.layers.BatchNormalization()(x)

  mu = tf.keras.layers.Dense(latent_dim, name='latent_mu')(x)
  sigma = tf.keras.layers.Dense(latent_dim, name ='latent_sigma')(x)
  return mu, sigma, batch_2.shape
```

Variational Autoencoders (VAE)

σ, μ used with a **Gaussian distribution** to **sample** their latent encoding pseudo randomly



A new custom layer called the sampling layer



Variational Autoencoders (VAE)

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

Sampling() calls the instance (similar to calling a function)

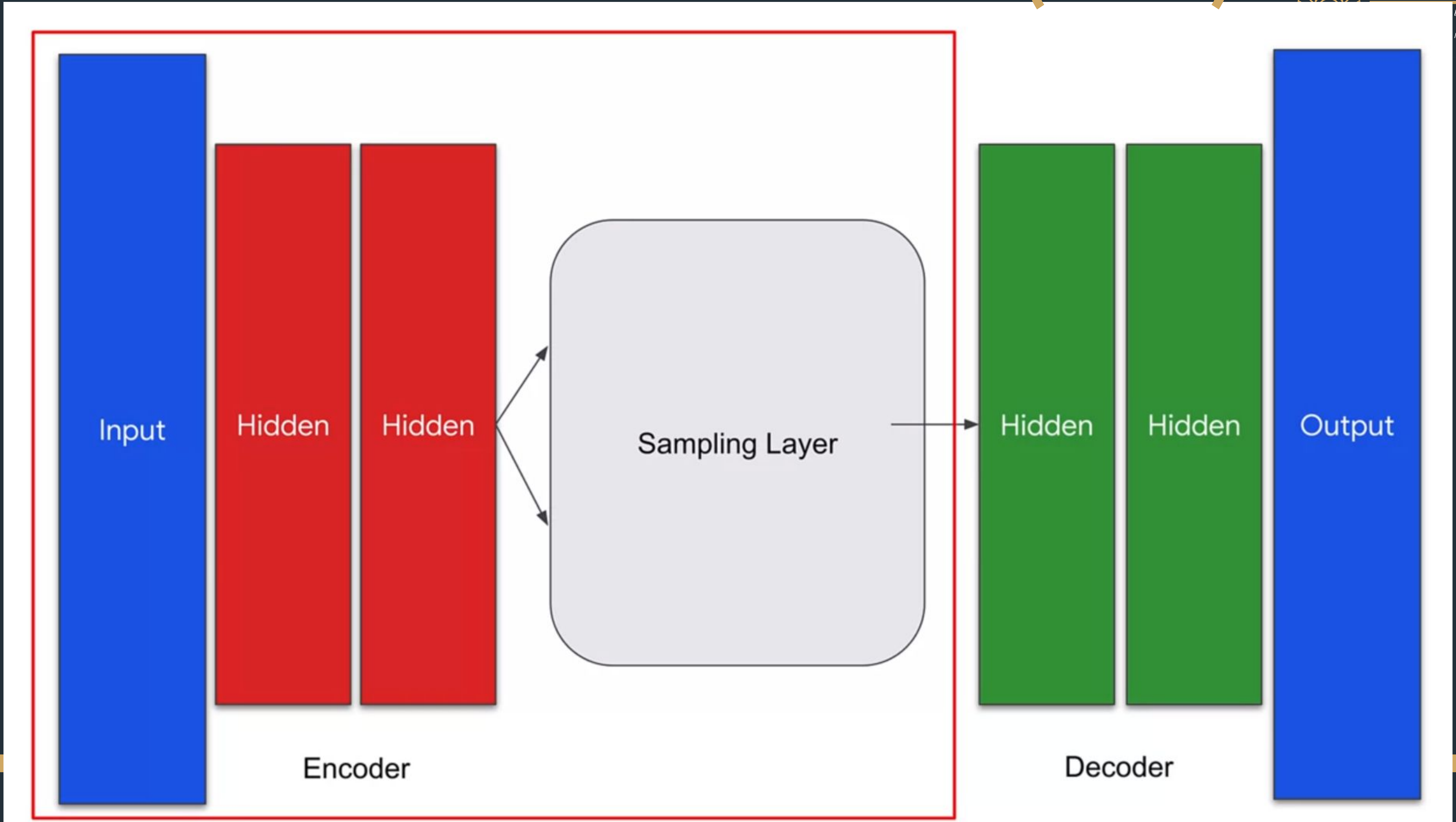
```
class Sampling(tf.keras.layers.Layer):
  def call(self, inputs):
    mu, sigma = inputs
    batch = tf.shape(mu)[0]
    dim = tf.shape(mu)[1]
    epsilon = tf.keras.backend.random_normal(shape=(batch, dim))
    return mu + tf.exp(0.5 * sigma) * epsilon
```

Tuple to a nested function

shape() returns a tuple.
[0] index accesses the first element which is the size of the first dimension of the mu.
Batch's size, dim are needed to get normal distri across that batch size for that dimension

Found to be effective (among the many different ways)

Variational Autoencoders (VAE)



Variational Autoencoders (VAE)

2

28x28

```
def encoder_model(LATENT_DIM, input_shape):
  inputs = tf.keras.layers.Input(shape=input_shape)
  mu, sigma, conv_shape = encoder_layers(inputs, latent_dim=LATENT_DIM)
  z = Sampling()((mu, sigma))
  model = tf.keras.Model(inputs, outputs=[mu, sigma, z])
  return model, conv_shape
```

Decoder needs it for reconstruction

Though the decoder does not need mu, sigma, the reconstruction loss function does

Decoder

- Latent representation of the data that included sampling from a Gaussian distribution
- Next, the decoder
  - Reconstruct data from the latent space
- Then, train the network to be able to generate new data based on the training set

```python
def decoder_layers(inputs, conv_shape):
  units = conv_shape[1] * conv_shape[2] * conv_shape[3]
  x = tf.keras.layers.Dense(units, activation = 'relu',
                            name="decode_dense1")(inputs)
  x = tf.keras.layers.BatchNormalization()(x)

  x = tf.keras.layers.Reshape((conv_shape[1], conv_shape[2], conv_shape[3]),
                               name="decode_reshape")(x)
  x = tf.keras.layers.Conv2DTranspose(filters=64, kernel_size=3, strides=2,
                                      padding='same', activation='relu',
                                      name="decode_conv2d_2")(x)
  x = tf.keras.layers.BatchNormalization()(x)

  x = tf.keras.layers.Conv2DTranspose(filters=32, kernel_size=3, strides=2,
                                      padding='same', activation='relu',
                                      name="decode_conv2d3")(x)
  x = tf.keras.layers.BatchNormalization()(x)

  x = tf.keras.layers.Conv2DTranspose(filters=1, kernel_size=3, strides=1, padding='same',
                                      activation='sigmoid', name="decode_final")(x)

  return x
```

7x7x64 (64 7x7 images)

Reverses the flattening done by the encoder

Inverts the convolutional filters

Creates a 2D transposed convolutional layer that is applied on x

Variational Autoencoders (VAE)

From the encoder

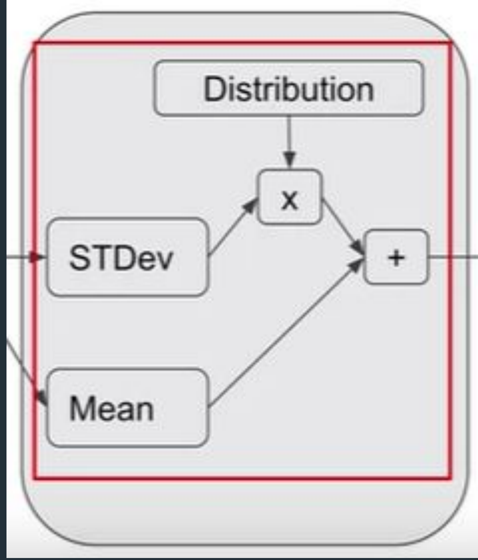
```
def decoder_model(latent_dim, conv_shape):
  inputs = tf.keras.layers.Input(shape=(latent_dim,))
  outputs = decoder_layers(inputs, conv_shape)
  model = tf.keras.Model(inputs, outputs)
  return model
```

Final step

- Create a reconstruction loss
  - to ensure that our random normal data is good
    - for reconstructing images
- Latent space in the middle
  - More complex than the dimension reduction

Final step

- Create a reconstruction loss
  - to ensure that our random normal data is good
    - for reconstructing images
- Latent space in the middle
  - More complex than the dimension reduction
- Loss function is good in having the network learn how to reconstruct the data that comes directly from the encoder
- But the encoding in the latent space is more complex
  - taking into account a random normal distribution and having this act on what the encoder learnt
- Need a loss function that quantifies how much a probability distribution differs from another
  - Kullback-Leibler (KL) loss function
    - Commonly used in VAEs
    - Encourages the encoder network to generate a latent distribution that is close to a known distribution, such as a unit Gaussian distribution



The closer the encoder's distribution is to the known distribution, the lower the loss

KL divergence loss function example

- In the context of multiclass classification, KL divergence is commonly used as a loss function in neural networks to measure the dissimilarity between the predicted probability distribution and the true distribution of the labels.
- $$KL(P \,||\, Q) = \Sigma(i=1 \text{ to } N)\ P(i) \log [P(i) / Q(i)]$$
- Pronounced as "the KL divergence of P from Q"
- P is the true probability distribution of the labels (i.e., the one-hot encoded representation of the correct label)
- Q is the predicted probability distribution of the labels
- N is the number of classes

KL divergence loss function example

Say, we have a multiclass classification problem with 5 classes, and our true distribution (P) for a particular data point is

[0, 0, 0.2, 0.8, 0]

Say, our model predicts a distribution (Q) for the same data point as [0.1, 0.1, 0.3, 0.4, 0.1]

KL(P || Q) = ?

KL divergence loss function example

Say, we have a multiclass classification problem with 5 classes, and our true distribution (P) for a particular data point is

[0, 0, 0.2, 0.8, 0]

Say, our model predicts a distribution (Q) for the same data point as [0.1, 0.1, 0.3, 0.4, 0.1]

KL(P || Q)

$= (0 * \ln(0/0.1) + 0 * \ln(0/0.1) + 0.2 * \ln(0.2/0.3) + 0.8 * \ln(0.8/0.4) + 0 * \ln(0/0.1))$

$= 0 + 0 - 0.0810 + 0.5545 + 0$

$= 0.4735$

The amount of information lost when approximating the true distribution with the predicted distribution

In VAEs

KL divergence is used as a regularization term to encourage the learned distribution of the latent space to be similar to a known distribution.

$$\text{KL Loss} = -\frac{1}{2}\sum_{i=1}^{D}\left(1 + \log(\sigma_i^2) - \mu_i^2 - \sigma_i^2\right)$$

how much information is lost when using the approximate distribution to represent the true distribution

- Approximate distribution refers to the distribution of the latent space (variables or code) generated by the encoder q(z|x) z the latent variable
- True distribution is a unit Gaussian distribution (mean=0, var=1)
- Mu, sigma are of the encoded latent variable z
- Summation is over all the values of z

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

```
# Define a kl reconstruction loss function
def kl_reconstruction_loss(mu, sigma):
  kl_loss = 1 + sigma - tf.square(mu) - tf.math.exp(sigma)
  return tf.reduce_mean(kl_loss) * -0.5
```

Instead of y, y^

https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence

https://arxiv.org/abs/2002.07514

```
def vae_model(encoder, decoder, input_shape):
  inputs = tf.keras.layers.Input(shape=input_shape)
  mu = encoder(inputs)[0]
  sigma = encoder(inputs)[1]
  z = encoder(inputs)[2]
  reconstructed = decoder(z)
  model = tf.keras.Model(inputs=inputs, outputs=reconstructed)
  loss = kl_reconstruction_loss(mu, sigma)
  model.add_loss(loss)
  return model
```

z is the output of the latent space by combining the random Gaussian, mu, sigma

Train the VAE model

```
for epoch in range(epochs):
  for step, x_batch_train in enumerate(train_dataset):
    with tf.GradientTape() as tape:
      reconstructed = vae(x_batch_train)
      flattened_inputs = tf.reshape(x_batch_train, shape=[-1])
      flattened_outputs = tf.reshape(reconstructed, shape=[-1])
      loss = bce_loss(flattened_inputs, flattened_outputs) * 784
      loss += sum(vae.losses)  # Add KLD regularization loss

    grads = tape.gradient(loss, vae.trainable_weights)
    optimizer.apply_gradients(zip(grads, vae.trainable_weights))
```

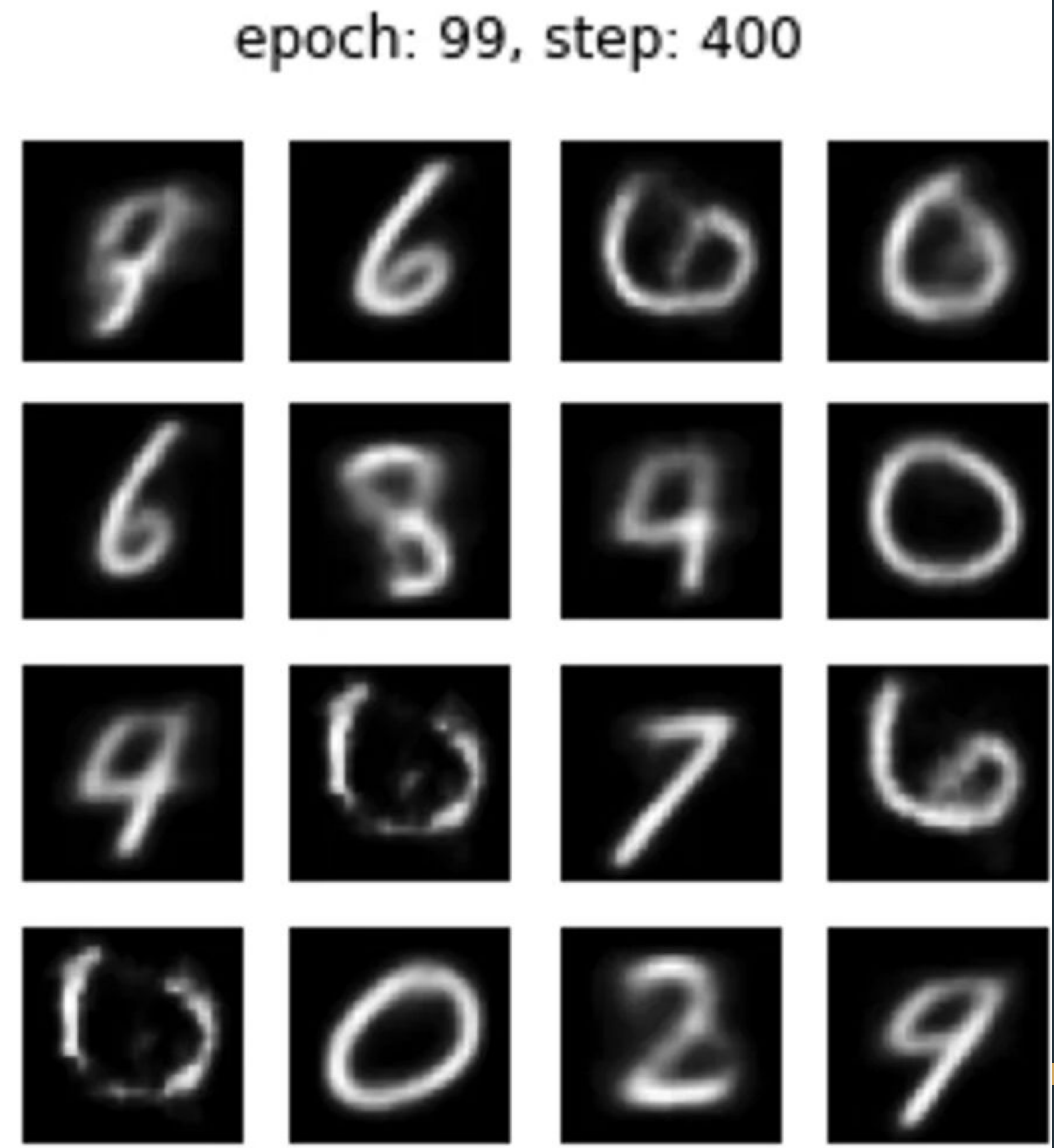
Binary cross entropy loss

28x28

Binary cross entropy loss function

- Gives the minimum value when the prediction = the ground truth
  - (i.e.) when outputs == inputs
- Unlike binary classification though where the ground truth is only 0 or 1, here, the minimal value will not be necessarily = 0 because the normalized MNIST pixel values are in the range [0,1]

Output



Applications of autoencoders

RV UNIVERSITY
Go, change the world
an initiative of RV EDUCATIONAL INSTITUTIONS

- Dimensionality Reduction
- Data Denoising
- Anomaly Detection
- Generative Models