

Your grade: 100%

Next item →

Your latest: 100% • Your highest: 100% • To pass you need at least 80%. We keep your highest score.

1. For Variational AutoEncoders, which of the following are the correct operations performed in the *latent space*?

1 / 1 point

- ☐ encoder mean + encoder STDev + gaussian distribution
- ☐ encoder mean * encoder STDev * gaussian distribution
- ☐ encoder mean * encoder STDev + gaussian distribution
- ☒ encoder mean + encoder STDev * gaussian distribution

✓ Correct

Correct!

2. Consider the following code, which is used in Variational AutoEncoder to represent the latent space. Fill in the missing piece of code.

1 / 1 point

(Note: Use shape as `shape=(batch, dim)`)

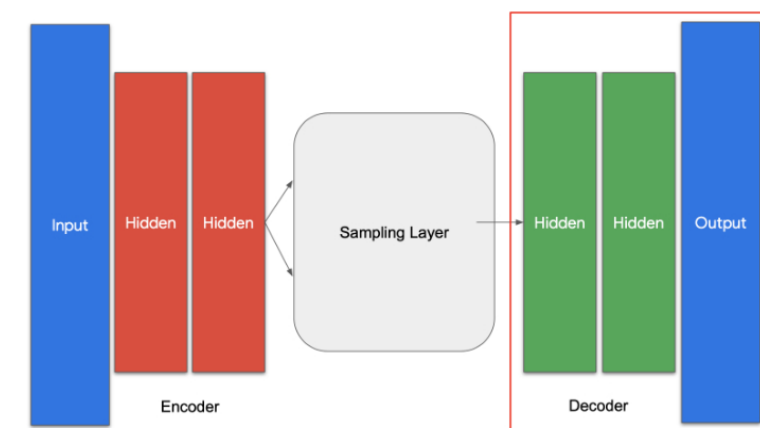
```
class Sampling(tf.keras.layers.Layer):
    def call(self, inputs):
        mu, sigma = inputs
        batch = tf.shape(mu)[0]
        dim = tf.shape(mu)[1]
        epsilon = # YOUR CODE HERE
        return mu + tf.exp(0.5 * sigma) * epsilon
```

`tf.keras.backend.random_normal(shape=(batch, dim))`

✓ Correct

3. When building the architecture for the decoder for a *convolutional Variational AutoEncoder*, what type of layers will you use ? Below is a screenshot of the code with # layer name # written in place of the actual layer that you would use. What goes in place of # layer name #?

1 / 1 point



```
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="# layer name #")(inputs)
    x = tf.keras.layers.BatchNormalization()(x)

    x = tf.keras.layers.Reshape((conv_shape[1], conv_shape[2], conv_shape[3]),
                                name="# layer name #")(x)
    x = tf.keras.layers.# layer name #(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.# layer name #(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.# layer name #(filters=1, kernel_size=3, strides=1,
        padding='same', activation='sigmoid', name="decode_final")(x)

return x

```

- ☒ Conv2DTranspose
- ☐ Conv2D
- ☐ Global AveragePooling2D
- ☐ MaxPooling2D.

 **Correct**

Correct! This will help you invert the convolutional filters applied during encoding.

4. Fill in the missing code for Kullback-Leibler cost function.

1 / 1 point

```

def kl_reconstruction_loss(inputs, outputs, mu, sigma):
    kl_loss = # YOUR CODE HERE
    return tf.reduce_mean(kl_loss) * - 0.5

```

- ☒ $kl_loss = 1 + \sigma - \mu^2 - \exp(\sigma)$
- ☐ $kl_loss = \sigma - \mu^2 - \exp(\sigma)$
- ☐ $\mu - \sigma^2 - \exp(\mu)$
- ☐ $kl_loss = 1 + \mu - \sigma^2 - \exp(\mu)$

 **Correct**

Correct!