Congratulations! You passed!

TO PASS 80% or higher

Keep Learning

GRADE

100%

Variational AutoEncoders

LATEST SUBMISSION GRADE

100%

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
```

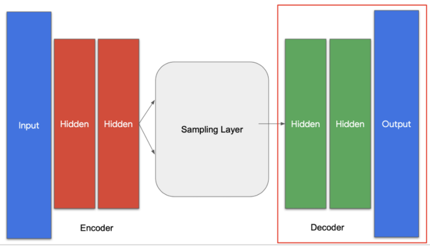
tf.keras.backend.random_normal(shape=(batch, dim))

Correct

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="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.# 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
```

- ( ) Conv2D
- (•) Conv2DTranspose
- ( ) 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
```

- ( ) mu - tf.square(sigma) - tf.math.exp(mu)
- (•) kl_loss = 1 + sigma - tf.square(mu) - tf.math.exp(sigma)
- ( ) kl_loss = 1 + mu - tf.square(sigma) - tf.math.exp(mu)
- ( ) kl_loss = sigma - tf.square(mu) - tf.math.exp(sigma)

Correct

Correct!

5. Which of the following is true regarding *loss* (the variable)?

1 / 1 point

```
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))
```

Correct!

5. Which of the following is true regarding *loss* (the variable)?

1 / 1 point

```
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))
```

- ( ) The closer the values of binary cross entropy loss function are to 0, the closer is the output to expected results.
- (•) The closer the values of binary cross entropy loss function are to 1, the closer is the output to expected results.

Correct

Correct!