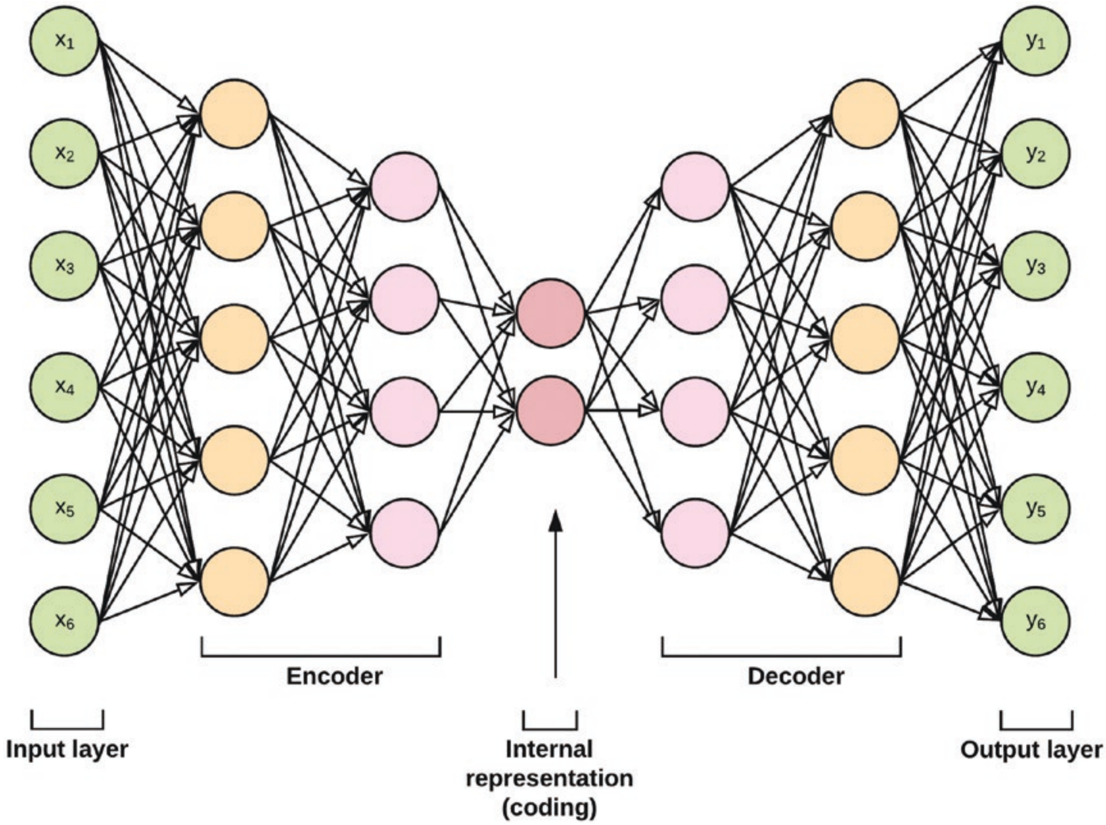

Figure 37-2. *Stacked or deep autoencoder. The hidden layers are added symmetrically at both the Encoder and Decoder*

Stacked Autoencoders with TensorFlow 2.0

The code example in this section shows how to implement an autoencoder network using TensorFlow 2.0. For simplicity, the MNIST handwriting dataset is used to create reconstructions of the original images. In this example, a stacked autoencoder is implemented with the original and reconstructed image shown in Figure 37-3. The code listing is presented in the following, and corresponding notes on the code are shown thereafter.

```
# import TensorFlow 2.0 with GPU
!pip install -q tf-nightly-gpu-2.0-preview

# import packages
import tensorflow as tf
```

```
import numpy as np
import matplotlib.pyplot as plt

# import dataset
(x_train, _), (x_test, _) = tf.keras.datasets.mnist.load_data()

# change datatype to float
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# scale the dataset from 0 -> 255 to 0 -> 1
x_train /= 255
x_test /= 255

# flatten the 28x28 images into vectors of size 784
x_train = x_train.reshape((len(x_train), np.prod(x_train.shape[1:])))
x_test = x_test.reshape((len(x_test), np.prod(x_test.shape[1:])))

# create the autoencoder model
def model_fn():
  model_input = tf.keras.layers.Input(shape=(784,))
  encoded = tf.keras.layers.Dense(units=512, activation='relu')(model_input)
  encoded = tf.keras.layers.Dense(units=128, activation='relu')(encoded)
  encoded = tf.keras.layers.Dense(units=64, activation='relu')(encoded)
  coding_layer = tf.keras.layers.Dense(units=32)(encoded)
  decoded = tf.keras.layers.Dense(units=64, activation='relu')(coding_layer)
  decoded = tf.keras.layers.Dense(units=128, activation='relu')(decoded)
  decoded = tf.keras.layers.Dense(units=512, activation='relu')(decoded)
  decoded_output = tf.keras.layers.Dense(units=784)(decoded)

  # the autoencoder model
  autoencoder_model = tf.keras.Model(inputs=model_input, outputs=decoded_output)

  # compile the model
  autoencoder_model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])

  return autoencoder_model
```

```
# build the model
autoencoder_model = model_fn()

# print autoencoder model summary
autoencoder_model.summary()

# train the model
autoencoder_model.fit(x_train, x_train, epochs=1000, batch_size=256,
                      shuffle=True, validation_data=(x_test, x_test))

# visualize reconstruction
sample_size = 6
test_image = x_test[:sample_size]
# reconstruct test samples
test_reconstruction = autoencoder_model.predict(test_image)

plt.figure(figsize = (8,25))
plt.suptitle('Stacked Autoencoder Reconstruction', fontsize=16)
for i in range(sample_size):
    plt.subplot(sample_size, 2, i*2+1)
    plt.title('Original image')
    plt.imshow(test_image[i].reshape((28, 28)), cmap="Greys",
    interpolation="nearest", aspect='auto')
    plt.subplot(sample_size, 2, i*2+2)
    plt.title('Reconstructed image')
    plt.imshow(test_reconstruction[i].reshape((28, 28)), cmap="Greys",
    interpolation="nearest", aspect='auto')
plt.show()
```

From the preceding code listing, take note of the following:

- Observe the arrangement of the encoder layers and the decoder layers of the stacked autoencoder. Specifically note how the corresponding layer arrangement of the encoder and the decoder has the same number of neurons.
- The loss error measures the squared difference between the inputs into the autoencoder network and the decoder output.

The image in Figure 37-3 contrasts the reconstructed images from the autoencoder network with the original images in the dataset.

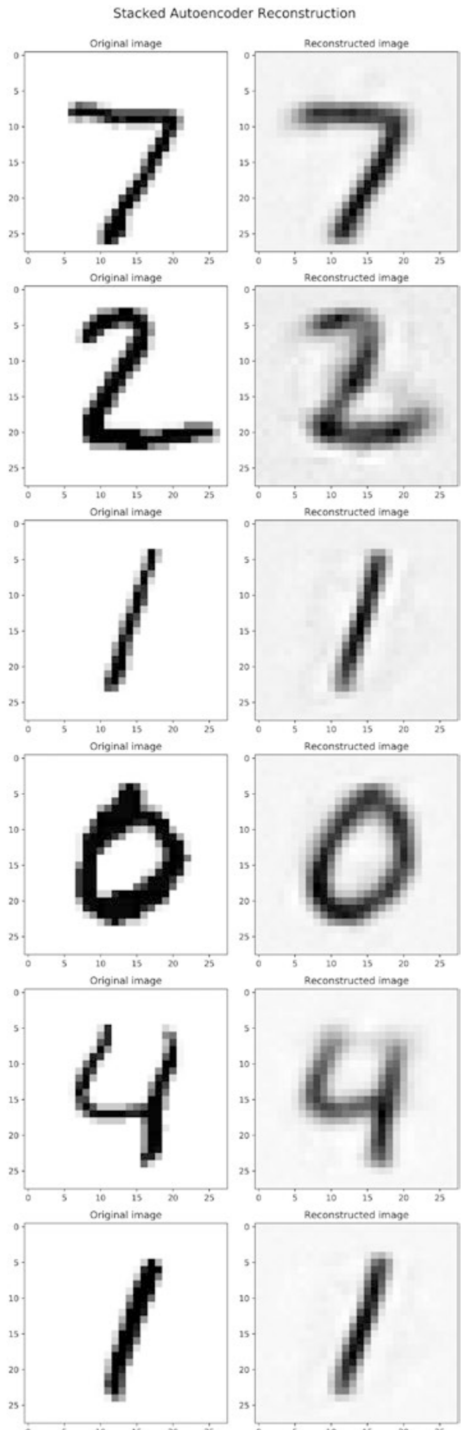

***Figure 37-3.** Stacked autoencoder reconstruction. Left: Original image. Right: Reconstructed image.*

Denoising Autoencoders

Denoising autoencoders add a different type of constraint to the network by imputing some Gaussian noise into the inputs. This noise injection forces the autoencoder to learn the uncorrupted form of the input features; by doing so, the autoencoder learns the internal representation of the dataset without memorizing the inputs.

Another way a denoising autoencoder constrains the input is by deactivating some input neurons in a similar fashion to the Dropout technique. Denoising autoencoders use an overcomplete network architecture. This means that the dimensions of the hidden Encoder and Decoder layers are not restricted; hence, they are overcomplete. An illustration of a denoising autoencoder architecture is shown in Figure 37-4.

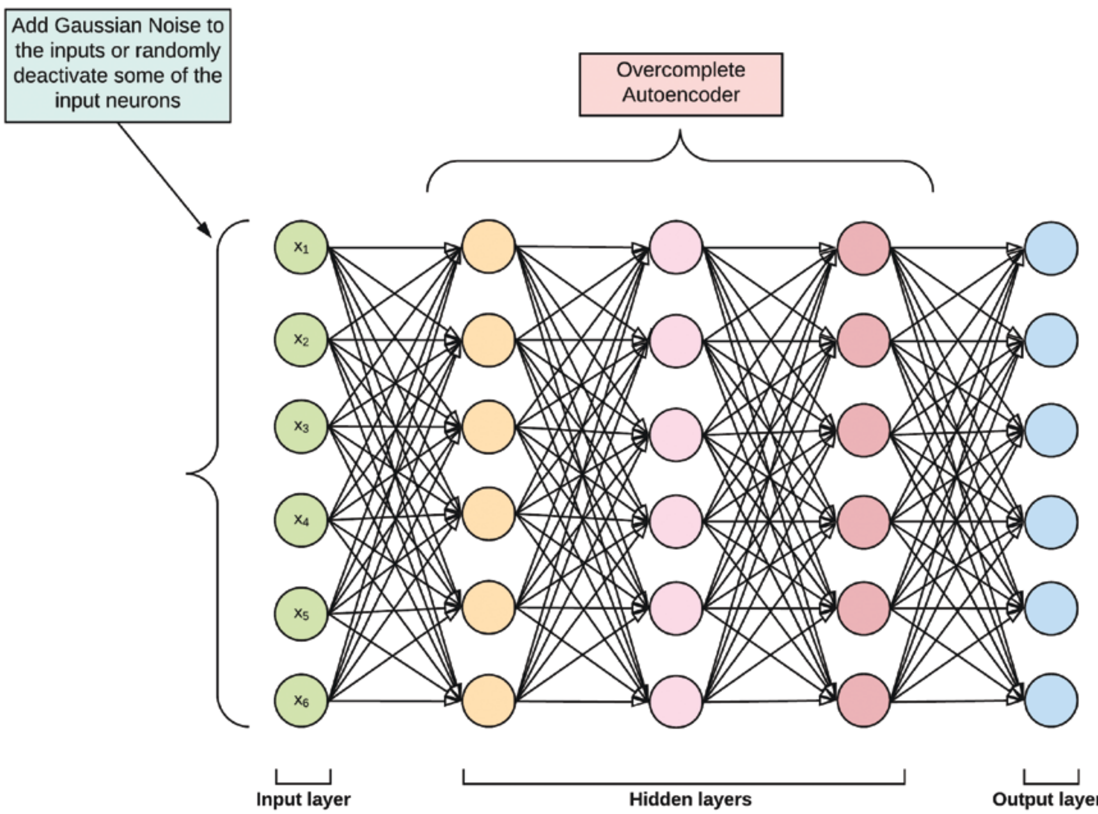

Figure 37-4. *Denoising autoencoder. Constraint is applied by either adding Gaussian noise or by switching off some a random selection of the input neurons.*