

layer composed of two neurons (the encoder), and one output layer composed of three neurons (the decoder). The outputs are often called the *reconstructions* since the autoencoder tries to reconstruct the inputs, and the cost function contains a *reconstruction loss* that penalizes the model when the reconstructions are different from the inputs.

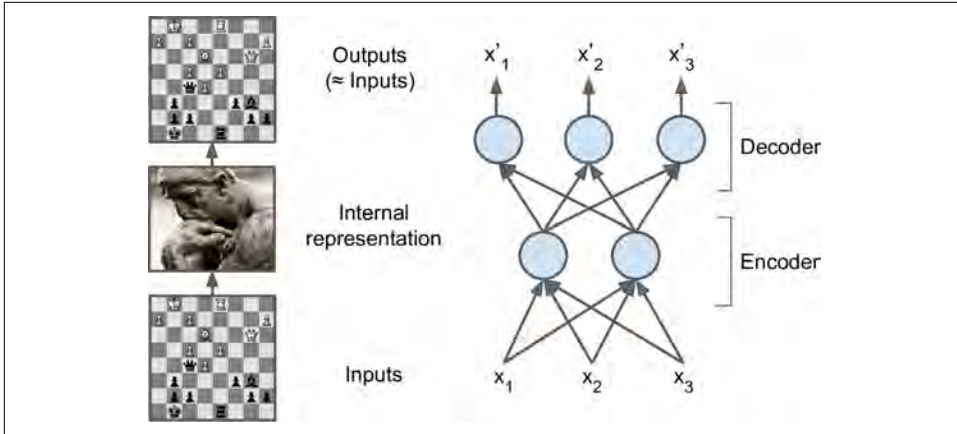


Figure 15-1. The chess memory experiment (left) and a simple autoencoder (right)

Because the internal representation has a lower dimensionality than the input data (it is 2D instead of 3D), the autoencoder is said to be *undercomplete*. An undercomplete autoencoder cannot trivially copy its inputs to the codings, yet it must find a way to output a copy of its inputs. It is forced to learn the most important features in the input data (and drop the unimportant ones).

Let's see how to implement a very simple undercomplete autoencoder for dimensionality reduction.

## Performing PCA with an Undercomplete Linear Autoencoder

If the autoencoder uses only linear activations and the cost function is the Mean Squared Error (MSE), then it can be shown that it ends up performing Principal Component Analysis (see [Chapter 8](#)).

The following code builds a simple linear autoencoder to perform PCA on a 3D dataset, projecting it to 2D:

```
import tensorflow as tf
from tensorflow.contrib.layers import fully_connected

n_inputs = 3 # 3D inputs
n_hidden = 2 # 2D codings
```

```

n_outputs = n_inputs

learning_rate = 0.01

X = tf.placeholder(tf.float32, shape=[None, n_inputs])
hidden = fully_connected(X, n_hidden, activation_fn=None)
outputs = fully_connected(hidden, n_outputs, activation_fn=None)

reconstruction_loss = tf.reduce_mean(tf.square(outputs - X)) # MSE

optimizer = tf.train.AdamOptimizer(learning_rate)
training_op = optimizer.minimize(reconstruction_loss)

init = tf.global_variables_initializer()

```

This code is really not very different from all the MLPs we built in past chapters. The two things to note are:

- The number of outputs is equal to the number of inputs.
- To perform simple PCA, we set `activation_fn=None` (i.e., all neurons are linear) and the cost function is the MSE. We will see more complex autoencoders shortly.

Now let's load the dataset, train the model on the training set, and use it to encode the test set (i.e., project it to 2D):

```

X_train, X_test = [...] # load the dataset

n_iterations = 1000
codings = hidden # the output of the hidden layer provides the codings

with tf.Session() as sess:
    init.run()
    for iteration in range(n_iterations):
        training_op.run(feed_dict={X: X_train}) # no labels (unsupervised)
    codings_val = codings.eval(feed_dict={X: X_test})

```

Figure 15-2 shows the original 3D dataset (at the left) and the output of the autoencoder's hidden layer (i.e., the coding layer, at the right). As you can see, the autoencoder found the best 2D plane to project the data onto, preserving as much variance in the data as it could (just like PCA).

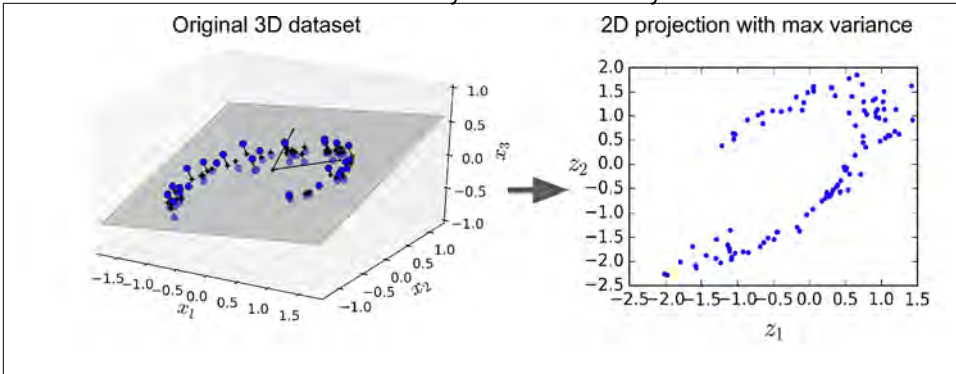


Figure 15-2. PCA performed by an undercomplete linear autoencoder

## Stacked Autoencoders

Just like other neural networks we have discussed, autoencoders can have multiple hidden layers. In this case they are called *stacked autoencoders* (or *deep autoencoders*). Adding more layers helps the autoencoder learn more complex codings. However, one must be careful not to make the autoencoder too powerful. Imagine an encoder so powerful that it just learns to map each input to a single arbitrary number (and the decoder learns the reverse mapping). Obviously such an autoencoder will reconstruct the training data perfectly, but it will not have learned any useful data representation in the process (and it is unlikely to generalize well to new instances).

The architecture of a stacked autoencoder is typically symmetrical with regards to the central hidden layer (the coding layer). To put it simply, it looks like a sandwich. For example, an autoencoder for MNIST (introduced in [Chapter 3](#)) may have 784 inputs, followed by a hidden layer with 300 neurons, then a central hidden layer of 150 neurons, then another hidden layer with 300 neurons, and an output layer with 784 neurons. This stacked autoencoder is represented in [Figure 15-3](#).

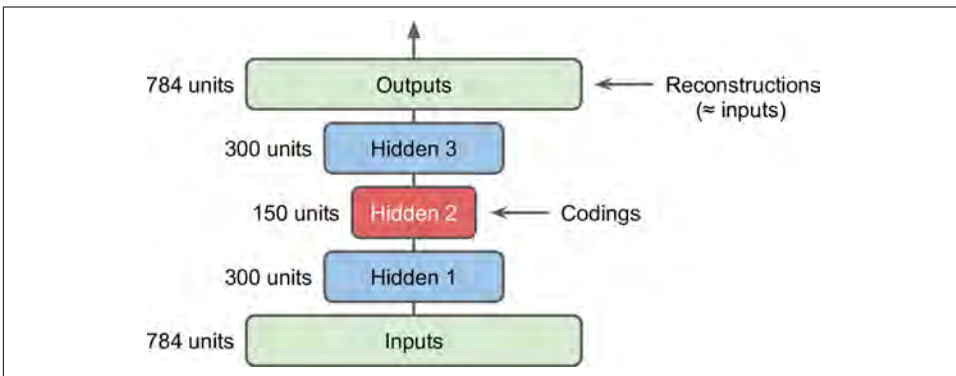


Figure 15-3. Stacked autoencoder