Download from finelybook www.finelybook.com

```
    weights1_val = weights1.eval()

for i in range(5):
    plt.subplot(1, 5, i + 1)
    plot_image(weights1_val.T[i])
```

You may get low-level features such as the ones shown in Figure 15-7.

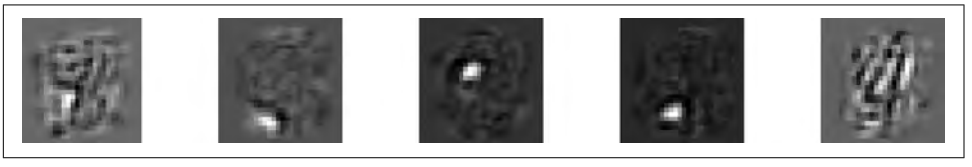

Figure 15-7. Features learned by five neurons from the first hidden layer

The first four features seem to correspond to small patches, while the fifth feature seems to look for vertical strokes (note that these features come from the stacked denoising autoencoder that we will discuss later).

Another technique is to feed the autoencoder a random input image, measure the activation of the neuron you are interested in, and then perform backpropagation to tweak the image in such a way that the neuron will activate even more. If you iterate several times (performing gradient ascent), the image will gradually turn into the most exciting image (for the neuron). This is a useful technique to visualize the kinds of inputs that a neuron is looking for.

Finally, if you are using an autoencoder to perform unsupervised pretraining—for example, for a classification task—a simple way to verify that the features learned by the autoencoder are useful is to measure the performance of the classifier.

Unsupervised Pretraining Using Stacked Autoencoders

As we discussed in Chapter 11, if you are tackling a complex supervised task but you do not have a lot of labeled training data, one solution is to find a neural network that performs a similar task, and then reuse its lower layers. This makes it possible to train a high-performance model using only little training data because your neural network won't have to learn all the low-level features; it will just reuse the feature detectors learned by the existing net.

Similarly, if you have a large dataset but most of it is unlabeled, you can first train a stacked autoencoder using all the data, then reuse the lower layers to create a neural network for your actual task, and train it using the labeled data. For example, Figure 15-8 shows how to use a stacked autoencoder to perform unsupervised pretraining for a classification neural network. The stacked autoencoder itself is typically trained one autoencoder at a time, as discussed earlier. When training the classifier, if

Download from finelybook www.finelybook.com
you really don't have much labeled training data, you may want to freeze the pretrained layers (at least the lower ones).

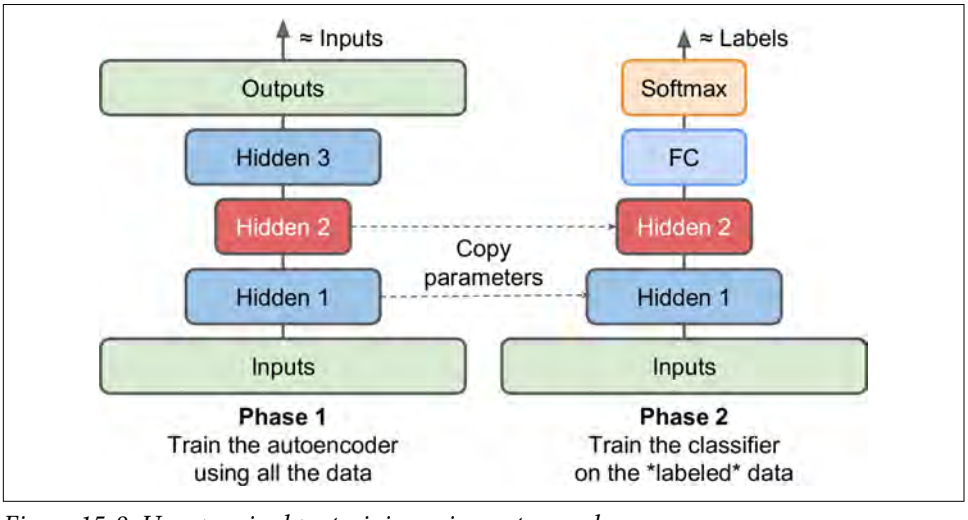

Figure 15-8. Unsupervised pretraining using autoencoders

This situation is actually quite common, because building a large unlabeled dataset is often cheap (e.g., a simple script can download millions of images off the internet), but labeling them can only be done reliably by humans (e.g., classifying images as cute or not). Labeling instances is time-consuming and costly, so it is quite common to have only a few thousand labeled instances.

As we discussed earlier, one of the triggers of the current Deep Learning tsunami is the discovery in 2006 by Geoffrey Hinton et al. that deep neural networks can be pretrained in an unsupervised fashion. They used restricted Boltzmann machines for that (see Appendix E), but in 2007 Yoshua Bengio et al. showed[2] that autoencoders worked just as well.

There is nothing special about the TensorFlow implementation: just train an autoencoder using all the training data, then reuse its encoder layers to create a new neural network (see Chapter 11 for more details on how to reuse pretrained layers, or check out the code examples in the Jupyter notebooks).

Up to now, in order to force the autoencoder to learn interesting features, we have limited the size of the coding layer, making it undercomplete. There are actually many other kinds of constraints that can be used, including ones that allow the cod-

2 "Greedy Layer-Wise Training of Deep Networks," Y. Bengio et al. (2007).

Download from finelybook www.finelybook.com
ing layer to be just as large as the inputs, or even larger, resulting in an *overcomplete autoencoder*. Let's look at some of those approaches now.

Denoising Autoencoders

Another way to force the autoencoder to learn useful features is to add noise to its inputs, training it to recover the original, noise-free inputs. This prevents the autoencoder from trivially copying its inputs to its outputs, so it ends up having to find patterns in the data.

The idea of using autoencoders to remove noise has been around since the 1980s (e.g., it is mentioned in Yann LeCun's 1987 master's thesis). In a 2008 paper,[3] Pascal Vincent et al. showed that autoencoders could also be used for feature extraction. In a 2010 paper,[4] Vincent et al. introduced *stacked denoising autoencoders*.

The noise can be pure Gaussian noise added to the inputs, or it can be randomly switched off inputs, just like in dropout (introduced in Chapter 11). Figure 15-9 shows both options.

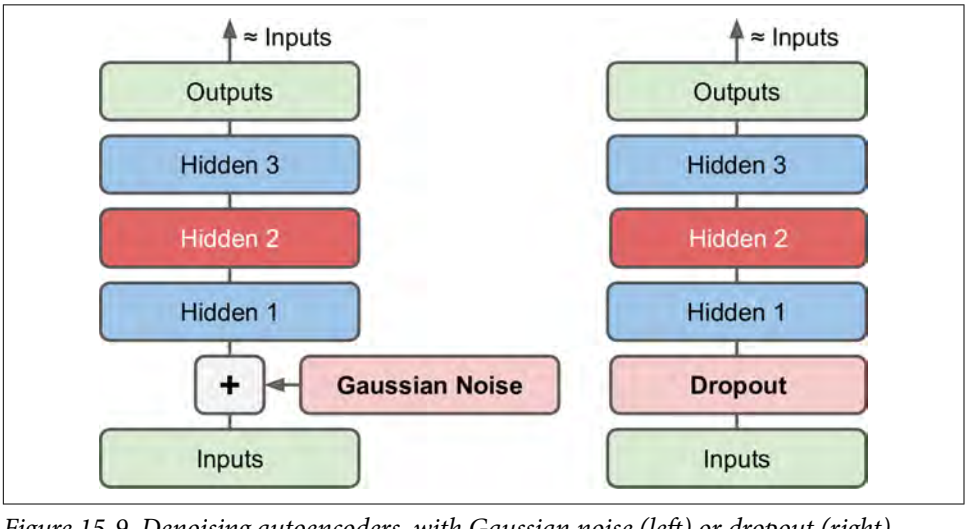

Figure 15-9. Denoising autoencoders, with Gaussian noise (left) or dropout (right)

3 "Extracting and Composing Robust Features with Denoising Autoencoders," P. Vincent et al. (2008).

4 "Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion," P. Vincent et al. (2010).