you have a complex task to solve, no similar model you can reuse, and little labeled training data but plenty of unlabeled training data.[9]

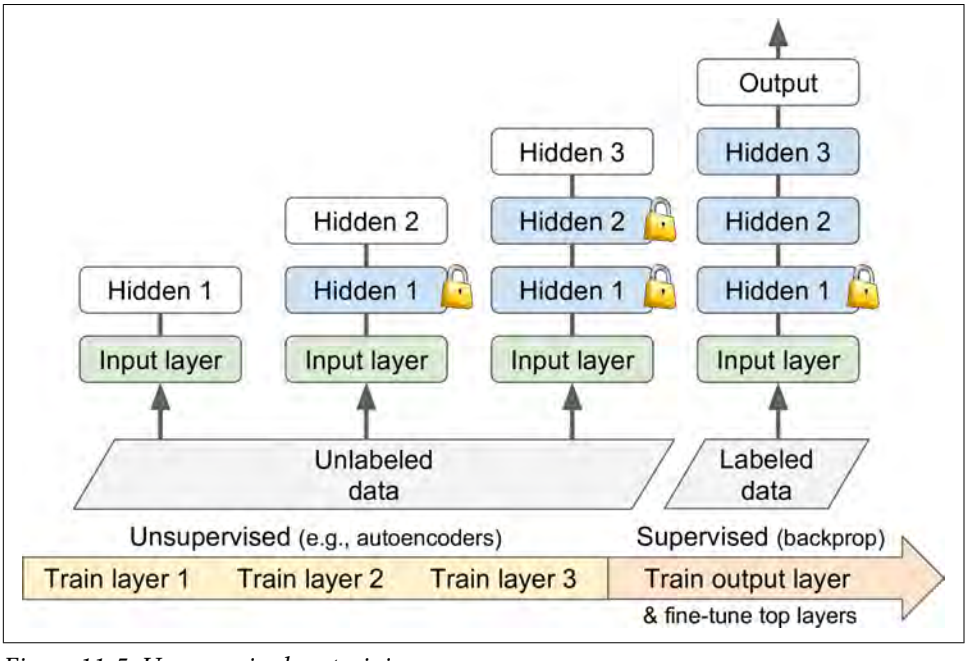

Figure 11-5. Unsupervised pretraining

Pretraining on an Auxiliary Task

One last option is to train a first neural network on an auxiliary task for which you can easily obtain or generate labeled training data, then reuse the lower layers of that network for your actual task. The first neural network's lower layers will learn feature detectors that will likely be reusable by the second neural network.

For example, if you want to build a system to recognize faces, you may only have a few pictures of each individual—clearly not enough to train a good classifier. Gathering hundreds of pictures of each person would not be practical. However, you could gather a lot of pictures of random people on the internet and train a first neural network to detect whether or not two different pictures feature the same person. Such a

9 Another option is to come up with a supervised task for which you can easily gather a lot of labeled training data, then use transfer learning, as explained earlier. For example, if you want to train a model to identify your friends in pictures, you could download millions of faces on the internet and train a classifier to detect whether two faces are identical or not, then use this classifier to compare a new picture with each picture of your friends.

network would learn good feature detectors for faces, so reusing its lower layers would allow you to train a good face classifier using little training data.

It is often rather cheap to gather unlabeled training examples, but quite expensive to label them. In this situation, a common technique is to label all your training examples as "good," then generate many new training instances by corrupting the good ones, and label these corrupted instances as "bad." Then you can train a first neural network to classify instances as good or bad. For example, you could download millions of sentences, label them as "good," then randomly change a word in each sentence and label the resulting sentences as "bad." If a neural network can tell that "The dog sleeps" is a good sentence but "The dog they" is bad, it probably knows quite a lot about language. Reusing its lower layers will likely help in many language processing tasks.

Another approach is to train a first network to output a score for each training instance, and use a cost function that ensures that a good instance's score is greater than a bad instance's score by at least some margin. This is called *max margin learning*.

Faster Optimizers

Training a very large deep neural network can be painfully slow. So far we have seen four ways to speed up training (and reach a better solution): applying a good initialization strategy for the connection weights, using a good activation function, using Batch Normalization, and reusing parts of a pretrained network. Another huge speed boost comes from using a faster optimizer than the regular Gradient Descent optimizer. In this section we will present the most popular ones: Momentum optimization, Nesterov Accelerated Gradient, AdaGrad, RMSProp, and finally Adam optimization.

Spoiler alert: the conclusion of this section is that you should almost always use Adam optimization,[10] so if you don't care about how it works, simply replace your `GradientDescentOptimizer` with an `AdamOptimizer` and skip to the next section! With just this small change, training will typically be several times faster. However, Adam optimization does have three hyperparameters that you can tune (plus the learning rate); the default values usually work fine, but if you ever need to tweak them it may be helpful to know what they do. Adam optimization combines several ideas from other optimization algorithms, so it is useful to look at these algorithms first.

10 At least for now: research is moving fast, especially in the field of optimization. Be sure to take a look at the latest and greatest optimizers every time a new version of TensorFlow is released.