arbitrary degree of accuracy, provided that it has enough hidden units. This means that an autoencoder with a single hidden layer is able to represent the identity function along the domain of the data arbitrarily well. However, the mapping from input to code is shallow. This means that we are not able to enforce arbitrary constraints, such as that the code should be sparse. A deep autoencoder, with at least one additional hidden layer inside the encoder itself, can approximate any mapping from input to code arbitrarily well, given enough hidden units.

Depth can exponentially reduce the computational cost of representing some functions. Depth can also exponentially decrease the amount of training data needed to learn some functions. See section 6.4.1 for a review of the advantages of depth in feedforward networks.

Experimentally, deep autoencoders yield much better compression than corresponding shallow or linear autoencoders (Hinton and Salakhutdinov, 2006).

A common strategy for training a deep autoencoder is to greedily pretrain the deep architecture by training a stack of shallow autoencoders, so we often encounter shallow autoencoders, even when the ultimate goal is to train a deep autoencoder.

14.4 Stochastic Encoders and Decoders

Autoencoders are just feedforward networks. The same loss functions and output unit types that can be used for traditional feedforward networks are also used for autoencoders.

As described in section 6.2.2.4, a general strategy for designing the output units and the loss function of a feedforward network is to define an output distribution $p(\boldsymbol{y} \mid \boldsymbol{x})$ and minimize the negative log-likelihood $-\log p(\boldsymbol{y} \mid \boldsymbol{x})$. In that setting, $\boldsymbol{y}$ was a vector of targets, such as class labels.

In the case of an autoencoder, $\boldsymbol{x}$ is now the target as well as the input. However, we can still apply the same machinery as before. Given a hidden code $\boldsymbol{h}$, we may think of the decoder as providing a conditional distribution $p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h})$. We may then train the autoencoder by minimizing $-\log p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h})$. The exact form of this loss function will change depending on the form of p_{decoder}. As with traditional feedforward networks, we usually use linear output units to parametrize the mean of a Gaussian distribution if $\boldsymbol{x}$ is real-valued. In that case, the negative log-likelihood yields a mean squared error criterion. Similarly, binary $\boldsymbol{x}$ values correspond to a Bernoulli distribution whose parameters are given by a sigmoid output unit, discrete $\boldsymbol{x}$ values correspond to a softmax distribution, and so on.