optimization algorithm. Training proceeds by minimizing

$$||\boldsymbol{x} - g(\boldsymbol{h})||^2 + \lambda|\boldsymbol{h}|_1 + \gamma||\boldsymbol{h} - f(\boldsymbol{x})||^2. \tag{14.19}$$

Like in sparse coding, the training algorithm alternates between minimization with respect to $\boldsymbol{h}$ and minimization with respect to the model parameters. Minimization with respect to $\boldsymbol{h}$ is fast because $f(\boldsymbol{x})$ provides a good initial value of $\boldsymbol{h}$ and the cost function constrains $\boldsymbol{h}$ to remain near $f(\boldsymbol{x})$ anyway. Simple gradient descent can obtain reasonable values of $\boldsymbol{h}$ in as few as ten steps.

The training procedure used by PSD is different from first training a sparse coding model and then training $f(\boldsymbol{x})$ to predict the values of the sparse coding features. The PSD training procedure regularizes the decoder to use parameters for which $f(\boldsymbol{x})$ can infer good code values.

Predictive sparse coding is an example of **learned approximate inference**. In section 19.5, this topic is developed further. The tools presented in chapter 19 make it clear that PSD can be interpreted as training a directed sparse coding probabilistic model by maximizing a lower bound on the log-likelihood of the model.

In practical applications of PSD, the iterative optimization is only used during training. The parametric encoder f is used to compute the learned features when the model is deployed. Evaluating f is computationally inexpensive compared to inferring $\boldsymbol{h}$ via gradient descent. Because f is a differentiable parametric function, PSD models may be stacked and used to initialize a deep network to be trained with another criterion.

14.9 Applications of Autoencoders

Autoencoders have been successfully applied to dimensionality reduction and information retrieval tasks. Dimensionality reduction was one of the first applications of representation learning and deep learning. It was one of the early motivations for studying autoencoders. For example, Hinton and Salakhutdinov (2006) trained a stack of RBMs and then used their weights to initialize a deep autoencoder with gradually smaller hidden layers, culminating in a bottleneck of 30 units. The resulting code yielded less reconstruction error than PCA into 30 dimensions and the learned representation was qualitatively easier to interpret and relate to the underlying categories, with these categories manifesting as well-separated clusters.

Lower-dimensional representations can improve performance on many tasks, such as classification. Models of smaller spaces consume less memory and runtime.