nonlinear transformation. Equivalently, we can apply the kernel trick described in section 5.7.2, to obtain a nonlinear learning algorithm based on implicitly applying the ϕ mapping. We can think of ϕ as providing a set of features describing $\boldsymbol{x}$, or as providing a new representation for $\boldsymbol{x}$.

The question is then how to choose the mapping ϕ.

1. One option is to use a very generic ϕ, such as the infinite-dimensional ϕ that is implicitly used by kernel machines based on the RBF kernel. If $\phi(\boldsymbol{x})$ is of high enough dimension, we can always have enough capacity to fit the training set, but generalization to the test set often remains poor. Very generic feature mappings are usually based only on the principle of local smoothness and do not encode enough prior information to solve advanced problems.

2. Another option is to manually engineer ϕ. Until the advent of deep learning, this was the dominant approach. This approach requires decades of human effort for each separate task, with practitioners specializing in different domains such as speech recognition or computer vision, and with little transfer between domains.

3. The strategy of deep learning is to learn ϕ. In this approach, we have a model $y = f(\boldsymbol{x}; \boldsymbol{\theta}, \boldsymbol{w}) = \phi(\boldsymbol{x}; \boldsymbol{\theta})^\top \boldsymbol{w}$. We now have parameters $\boldsymbol{\theta}$ that we use to learn ϕ from a broad class of functions, and parameters $\boldsymbol{w}$ that map from $\phi(\boldsymbol{x})$ to the desired output. This is an example of a deep feedforward network, with ϕ defining a hidden layer. This approach is the only one of the three that gives up on the convexity of the training problem, but the benefits outweigh the harms. In this approach, we parametrize the representation as $\phi(\boldsymbol{x}; \boldsymbol{\theta})$ and use the optimization algorithm to find the $\boldsymbol{\theta}$ that corresponds to a good representation. If we wish, this approach can capture the benefit of the first approach by being highly generic—we do so by using a very broad family $\phi(\boldsymbol{x}; \boldsymbol{\theta})$. This approach can also capture the benefit of the second approach. Human practitioners can encode their knowledge to help generalization by designing families $\phi(\boldsymbol{x}; \boldsymbol{\theta})$ that they expect will perform well. The advantage is that the human designer only needs to find the right general function family rather than finding precisely the right function.

This general principle of improving models by learning features extends beyond the feedforward networks described in this chapter. It is a recurring theme of deep learning that applies to all of the kinds of models described throughout this book. Feedforward networks are the application of this principle to learning deterministic