describe here some of the basic intuitions motivating each type of hidden units. These intuitions can help decide when to try out each of these units. It is usually impossible to predict in advance which will work best. The design process consists of trial and error, intuiting that a kind of hidden unit may work well, and then training a network with that kind of hidden unit and evaluating its performance on a validation set.

Some of the hidden units included in this list are not actually differentiable at all input points. For example, the rectified linear function $g(z) = \max\{0, z\}$ is not differentiable at $z = 0$. This may seem like it invalidates g for use with a gradient-based learning algorithm. In practice, gradient descent still performs well enough for these models to be used for machine learning tasks. This is in part because neural network training algorithms do not usually arrive at a local minimum of the cost function, but instead merely reduce its value significantly, as shown in figure 4.3. These ideas will be described further in chapter 8. Because we do not expect training to actually reach a point where the gradient is $\mathbf{0}$, it is acceptable for the minima of the cost function to correspond to points with undefined gradient. Hidden units that are not differentiable are usually non-differentiable at only a small number of points. In general, a function $g(z)$ has a left derivative defined by the slope of the function immediately to the left of z and a right derivative defined by the slope of the function immediately to the right of z. A function is differentiable at z only if both the left derivative and the right derivative are defined and equal to each other. The functions used in the context of neural networks usually have defined left derivatives and defined right derivatives. In the case of $g(z) = \max\{0, z\}$, the left derivative at $z = 0$ is 0 and the right derivative is 1. Software implementations of neural network training usually return one of the one-sided derivatives rather than reporting that the derivative is undefined or raising an error. This may be heuristically justified by observing that gradient-based optimization on a digital computer is subject to numerical error anyway. When a function is asked to evaluate $g(0)$, it is very unlikely that the underlying value truly was 0. Instead, it was likely to be some small value ϵ that was rounded to 0. In some contexts, more theoretically pleasing justifications are available, but these usually do not apply to neural network training. The important point is that in practice one can safely disregard the non-differentiability of the hidden unit activation functions described below.

Unless indicated otherwise, most hidden units can be described as accepting a vector of inputs $\boldsymbol{x}$, computing an affine transformation $\boldsymbol{z} = \boldsymbol{W}^\top \boldsymbol{x} + \boldsymbol{b}$, and then applying an element-wise nonlinear function $g(\boldsymbol{z})$. Most hidden units are distinguished from each other only by the choice of the form of the activation function $g(\boldsymbol{z})$.