the search. The simplest method for doing so is known as **Newton's method**. Newton's method is based on using a second-order Taylor series expansion to approximate $f(\boldsymbol{x})$ near some point $\boldsymbol{x}^{(0)}$:

$$f(\boldsymbol{x}) \approx f(\boldsymbol{x}^{(0)}) + (\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(0)}) + \frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}^{(0)})^\top \boldsymbol{H}(f)(\boldsymbol{x}^{(0)})(\boldsymbol{x} - \boldsymbol{x}^{(0)}). \quad (4.11)$$

If we then solve for the critical point of this function, we obtain:

$$\boldsymbol{x}^* = \boldsymbol{x}^{(0)} - \boldsymbol{H}(f)(\boldsymbol{x}^{(0)})^{-1} \nabla_{\boldsymbol{x}} f(\boldsymbol{x}^{(0)}). \quad (4.12)$$

When f is a positive definite quadratic function, Newton's method consists of applying equation 4.12 once to jump to the minimum of the function directly. When f is not truly quadratic but can be locally approximated as a positive definite quadratic, Newton's method consists of applying equation 4.12 multiple times. Iteratively updating the approximation and jumping to the minimum of the approximation can reach the critical point much faster than gradient descent would. This is a useful property near a local minimum, but it can be a harmful property near a saddle point. As discussed in section 8.2.3, Newton's method is only appropriate when the nearby critical point is a minimum (all the eigenvalues of the Hessian are positive), whereas gradient descent is not attracted to saddle points unless the gradient points toward them.

Optimization algorithms that use only the gradient, such as gradient descent, are called **first-order optimization algorithms**. Optimization algorithms that also use the Hessian matrix, such as Newton's method, are called **second-order optimization algorithms** (Nocedal and Wright, 2006).

The optimization algorithms employed in most contexts in this book are applicable to a wide variety of functions, but come with almost no guarantees. Deep learning algorithms tend to lack guarantees because the family of functions used in deep learning is quite complicated. In many other fields, the dominant approach to optimization is to design optimization algorithms for a limited family of functions.

In the context of deep learning, we sometimes gain some guarantees by restricting ourselves to functions that are either **Lipschitz continuous** or have Lipschitz continuous derivatives. A Lipschitz continuous function is a function f whose rate of change is bounded by a **Lipschitz constant** $\mathcal{L}$:

$$\forall \boldsymbol{x}, \forall \boldsymbol{y}, |f(\boldsymbol{x}) - f(\boldsymbol{y})| \leq \mathcal{L} ||\boldsymbol{x} - \boldsymbol{y}||_2. \quad (4.13)$$

This property is useful because it allows us to quantify our assumption that a small change in the input made by an algorithm such as gradient descent will have