can easily become a limitation on how quickly and effectively different architectures can be explored empirically.

Section 8.1.4

A second approach is to evaluate the gradients numerically using finite differences. This requires only a software implementation of the forward propagation equations. One problem with numerical differentiation is that it has limited computational accuracy, although this is unlikely to be an issue for network training as we may be using stochastic gradient descent in which each evaluation is only a very noisy estimate of the local gradient. The main drawback of this approach is that it scales poorly with the size of the network. However, the technique is useful for debugging other approaches, because the gradients are evaluated using only the forward propagation code and so can be used to confirm the correctness of backpropagation or other code used to evaluate gradients.

A third approach is called *symbolic differentiation* and makes use of specialist software to automate the analytical manipulations that are done by hand in the first approach. This process is an example of *computer algebra* or *symbolic computation* and involves the automatic application of the rules of calculus, such as the chain rule, in a completely mechanistic process. The resulting expressions are then implemented in standard software. An obvious advantage of this approach is that it avoids human error in the manual derivation of the backpropagation equations. Moreover, the gradients are again calculated to machine precision, and the poor scaling seen with numerical differentiation is avoided. The major downside of symbolic differentiation, however, is that the resulting expressions for derivatives can become exponentially longer than the original function, with correspondingly long evaluation times. Consider a function $f(x)$ given by the product of $u(x)$ and $v(x)$. The function and its derivative are given by

$$f(x) = u(x)v(x) \tag{8.42}$$

$$f'(x) = u'(x)v(x) + u(x)v'(x). \tag{8.43}$$

We see that there is redundant computation in that $u(x)$ and $v(x)$ must be evaluated both for the calculation of $f(x)$ and for $f'(x)$. If the factors $u(x)$ and $v(x)$ themselves involve factors, then we end up with a nested duplication of expressions, which rapidly grow in complexity. This problem is called *expression swell*.

As a further illustration, consider a function that is structured like two layers of a neural network (Grosse, 2018) with a single input x, a hidden unit with activation z, and an output y in which

$$z = h(w_1 x + b_1) \tag{8.44}$$

$$y = h(w_2 z + b_2) \tag{8.45}$$

where $h(a)$ is the soft ReLU:

$$\zeta(a) = \ln\left(1 + \exp(a)\right). \tag{8.46}$$

The overall function is therefore given by

$$y(x) = h\left(w_2 h(w_1 x + b_1) + b_2\right) \tag{8.47}$$