Exercise 8.13

and the derivative of the network output with respect to w_1, evaluated symbolically, is given by

$$\frac{\partial y}{\partial w_1} = \frac{w_2 x \exp\left(w_1 x + b_1 + b_2 + w_2 \ln[1 + e^{w_1 x + b_1}]\right)}{(1 + e^{w_1 x + b_1})\,(1 + \exp(b_2 + w_2 \ln[1 + e^{w_1 x + b_1}]))}. \tag{8.48}$$

As well as being significantly more complex than the original function, we also see redundant computation where expressions such as $w_1 x + b_1$ occur in several places.

A further major drawback with symbolic differentiation is that it requires that the expression to be differentiated is expressed in closed form. It therefore excludes important control flow operations such as loops, recursions, conditional execution, and procedure calls, which are valuable constructs that we might wish to use when defining the network function.

We therefore turn to the fourth technique for evaluating derivatives in neural networks called *automatic differentiation*, also known as 'autodiff' or 'algorithmic differentiation' (Baydin *et al.*, 2018). Unlike symbolic differentiation, the goal of automatic differentiation is not to find a mathematical expression for the derivatives but to have the computer automatically generate the code that implements the gradient calculations given only the code for the forward propagation equations. It is accurate to machine precision, just as with symbolic differentiation, but is more efficient because it is able to exploit intermediate variables used in the definition of the forward propagation equations and thereby avoid redundant evaluations. It is important to note that not only can automatic differentiation handle conventional closed-form mathematical expressions but it can also deal with flow control elements such as branches, loops, recursion, and procedure calls, and is therefore significantly more powerful than symbolic differentiation. Automatic differentiation is a well-established field with broad applicability that was developed largely outside of the machine learning community. Modern deep learning is a largely empirical process, involving evaluating and comparing different architectures, and automatic differentiation therefore plays a key role in enabling this experimentation to be done accurately and efficiently.

The key idea of automatic differentiation is to take the code that evaluates a function, for example the forward propagation equations that evaluate the error function for a neural network, and augment the code with additional variables whose values are accumulated during code execution to obtain the required derivatives. There are two principal forms of automatic differentiation, known as forward mode and reverse mode. We start by looking at forward mode, which is conceptually somewhat simpler.

8.2.1 Forward-mode automatic differentiation

In forward-mode automatic differentiation, we augment each intermediate variable z_i, known as a 'primal' variable, involved in the evaluation of a function, such as the error function of a neural network, with an additional variable representing the value of some derivative of that variable, which we can denote $\dot{z}_i$, known as a 'tangent' variable. The tangent variables and their associated code are generated