Each operation `op` is also associated with a `bprop` operation. This `bprop` operation can compute a Jacobian-vector product as described by equation 6.47. This is how the back-propagation algorithm is able to achieve great generality. Each operation is responsible for knowing how to back-propagate through the edges in the graph that it participates in. For example, we might use a matrix multiplication operation to create a variable $\boldsymbol{C} = \boldsymbol{AB}$. Suppose that the gradient of a scalar z with respect to $\boldsymbol{C}$ is given by $\boldsymbol{G}$. The matrix multiplication operation is responsible for defining two back-propagation rules, one for each of its input arguments. If we call the `bprop` method to request the gradient with respect to $\boldsymbol{A}$ given that the gradient on the output is $\boldsymbol{G}$, then the `bprop` method of the matrix multiplication operation must state that the gradient with respect to $\boldsymbol{A}$ is given by $\boldsymbol{GB}^\top$. Likewise, if we call the `bprop` method to request the gradient with respect to $\boldsymbol{B}$, then the matrix operation is responsible for implementing the `bprop` method and specifying that the desired gradient is given by $\boldsymbol{A}^\top\boldsymbol{G}$. The back-propagation algorithm itself does not need to know any differentiation rules. It only needs to call each operation's `bprop` rules with the right arguments. Formally, $\texttt{op.bprop}(\texttt{inputs}, \mathsf{X}, \mathsf{G})$ must return

$$\sum_i (\nabla_{\mathsf{X}} \texttt{op.f}(\texttt{inputs})_i)\, \mathsf{G}_i, \tag{6.54}$$

which is just an implementation of the chain rule as expressed in equation 6.47. Here, `inputs` is a list of inputs that are supplied to the operation, `op.f` is the mathematical function that the operation implements, X is the input whose gradient we wish to compute, and G is the gradient on the output of the operation.

The `op.bprop` method should always pretend that all of its inputs are distinct from each other, even if they are not. For example, if the `mul` operator is passed two copies of x to compute x^2, the `op.bprop` method should still return x as the derivative with respect to both inputs. The back-propagation algorithm will later add both of these arguments together to obtain $2x$, which is the correct total derivative on x.

Software implementations of back-propagation usually provide both the operations and their `bprop` methods, so that users of deep learning software libraries are able to back-propagate through graphs built using common operations like matrix multiplication, exponents, logarithms, and so on. Software engineers who build a new implementation of back-propagation or advanced users who need to add their own operation to an existing library must usually derive the `op.bprop` method for any new operations manually.

The back-propagation algorithm is formally described in algorithm 6.5.