terms of constructing a computational graph for the derivatives. Any subset of the graph may then be evaluated using specific numerical values at a later time. This allows us to avoid specifying exactly when each operation should be computed. Instead, a generic graph evaluation engine can evaluate every node as soon as its parents' values are available.

The description of the symbol-to-symbol based approach subsumes the symbol-to-number approach. The symbol-to-number approach can be understood as performing exactly the same computations as are done in the graph built by the symbol-to-symbol approach. The key difference is that the symbol-to-number approach does not expose the graph.

6.5.6 General Back-Propagation

The back-propagation algorithm is very simple. To compute the gradient of some scalar z with respect to one of its ancestors $\boldsymbol{x}$ in the graph, we begin by observing that the gradient with respect to z is given by $\frac{dz}{dz} = 1$. We can then compute the gradient with respect to each parent of z in the graph by multiplying the current gradient by the Jacobian of the operation that produced z. We continue multiplying by Jacobians traveling backwards through the graph in this way until we reach $\boldsymbol{x}$. For any node that may be reached by going backwards from z through two or more paths, we simply sum the gradients arriving from different paths at that node.

More formally, each node in the graph $\mathcal{G}$ corresponds to a variable. To achieve maximum generality, we describe this variable as being a tensor V. Tensor can in general have any number of dimensions. They subsume scalars, vectors, and matrices.

We assume that each variable V is associated with the following subroutines:

- `get_operation`(V): This returns the operation that computes V, represented by the edges coming into V in the computational graph. For example, there may be a Python or C++ class representing the matrix multiplication operation, and the `get_operation` function. Suppose we have a variable that is created by matrix multiplication, $\boldsymbol{C} = \boldsymbol{AB}$. Then `get_operation`($\mathsf{V}$) returns a pointer to an instance of the corresponding C++ class.

- `get_consumers`($\mathsf{V}, \mathcal{G}$): This returns the list of variables that are children of V in the computational graph $\mathcal{G}$.

- `get_inputs`($\mathsf{V}, \mathcal{G}$): This returns the list of variables that are parents of V in the computational graph $\mathcal{G}$.