Computing Derivatives

In order to train a network, we will need to compute derivatives of the loss between the actual and target outputs of the network, with respect to its parameters. To do so however, we will also need to compute derivatives for all intermediate variables computed by the network.

In the lectures we (will) have considered a column vector notation, where all vectors are column vectors. We present a series of derivative rules for column vectors.

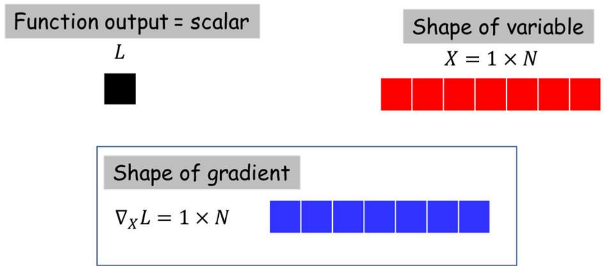
Python considers row vectors by default, so all our rules will be transposed, which leads to the following rules for derivatives.

All the equations below are in terms of *gradients*. We will represent the gradient of a variable Y with respect to variable X as $\nabla_X Y$.

We can now specify the following rules:

- The gradient of the loss (which is a scalar) with respect to a $1 \times N$ row vector is a $1 \times N$ row vector.

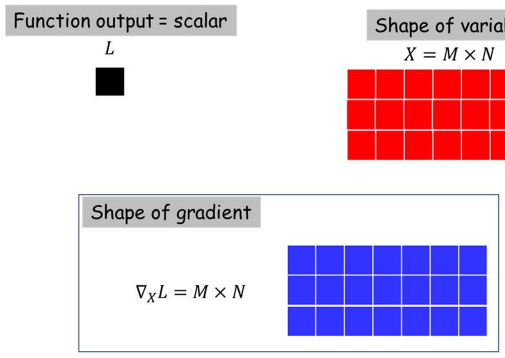
  Figure: Illustration of variable dimensions for $L = f(X)$ with scalar L and row vector X

  The i-th component of $\nabla_X L$ is the partial derivative $\frac{\partial L}{\partial X_i}$.

- The gradient of the loss with respect to an $M \times N$ matrix will also be an $M \times N$ matrix.

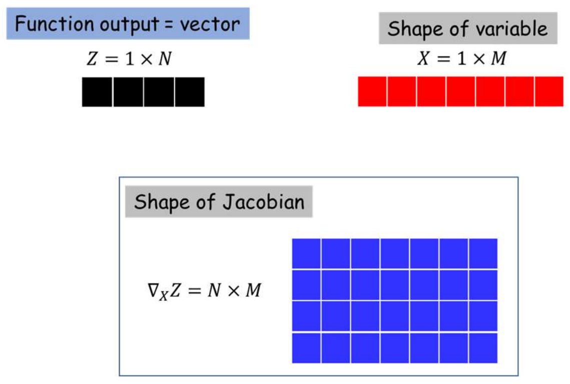
  Figure: Illustration of variable dimensions for $L = f(X)$ with scalar L and $M \times N$ matrix X

  The (i, j)th element of $\nabla_X L$ is the partial derivative $\frac{\partial L}{\partial X_{i,j}}$.

- The Jacobian of a $1 \times N$ row vector with respect to a $1 \times M$ row vector will be an $N \times M$ matrix.

Figure: Illustration of variable dimensions for $Z = f(X)$, where Z is an $1 \times N$ vector and X, is $1 \times M$



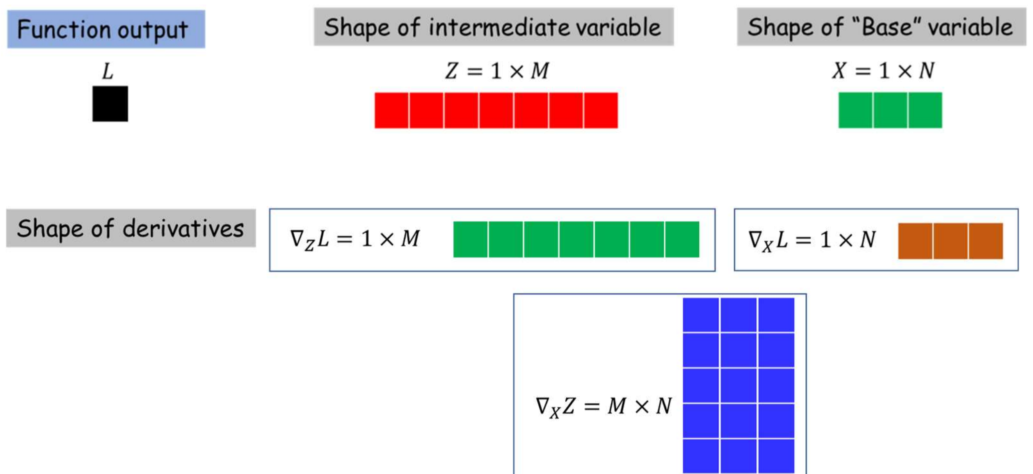
The (i, j)th element of $\nabla_X Z$ is the partial derivative $\frac{\partial Z_i}{\partial X_j}$. Note the indices: The numerator index in the partial derivative corresponds to the *first* (row) index in the Jacobian matrix $\nabla_X Z$, and the denominator index corresponds to the *second* (column) index.

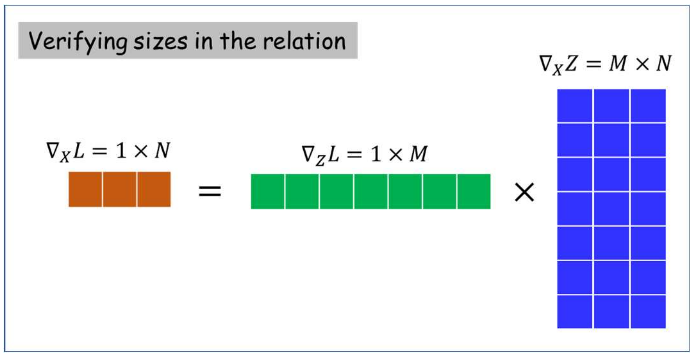
- In any application of the chain rule, the dimensions must match up. So, for instance, if we have $L = f(Z)$ where $Z = h(X)$, where L is a scalar (e.g. a loss), Z is an $1 \times M$ row vector and X is an $1 \times N$ row vector, then, by the rules given just above
  - $\nabla_X L$, the gradient of L w.r.t. X must be a $1 \times N$ row vector.
  - $\nabla_Z L$, the gradient of L w.r.t. Z must be a $1 \times M$ row vector
  - $\nabla_X Z$, the Jacobian of Z w.r.t. X must be an $M \times N$ matrix
  - The chain rule for the derivatives now is given by:

$$\nabla_X L = \nabla_Z L \, \nabla_X Z$$

The following figure illustrates the chain rule relation.

Figure: Illustration of how sizes must match up when we use the chain rule. The relation used is $L = f(Z)$ where $Z = G(X)$. L is a scalar, Z is a $1 \times M$ vector, and X is a $1 \times N$ vector. The objective is to compute $\nabla_X L$, the gradient of L w.r.t. X.


Function output
L
Shape of intermediate variable
$Z = 1 \times M$
Shape of "Base" variable
$X = 1 \times N$
Shape of derivatives
$\nabla_Z L = 1 \times M$
$\nabla_X L = 1 \times N$
$\nabla_X Z = M \times N$



Verifying sizes in the relation
$\nabla_X Z = M \times N$
$\nabla_X L = 1 \times N$
$\nabla_Z L = 1 \times M$
$=$
$\times$

The consequence of the above rules is that the shape of the derivative of the loss with respect to *any* network parameter or intermediate variable in the network will be the *transpose* of the shape of the parameter or variable.

Additional rules:

We will additionally use the following simple rules in computing derivatives:

a. For any computation of the kind $[a, b, c] = F(d, e, f)$, where the operation takes *in* variables d, e and f, and computes values a, b and c, the derivative computation will be *backward:*

$$\partial d,\ \partial e,\ \partial f = B(\partial a, \partial b, \partial c)$$

where ∂a, ∂b, ∂c, ∂d, ∂e and ∂f are the derivatives of the loss with respect to a, b, c, d, e and f respectively and B() is the function that computes the derivative for F().

Note the reverse in the order of variables: while computing derivatives, the derivatives w.r.t. $\partial a, \partial b, \partial c$, the *output* variables a, b, c of the "forward" function $F()$ are *input* to the "backward" function $B()$. The *output* of the backward function $B()$ are $\partial d, \partial e, \partial f$, the derivatives w.r.t d, e and f, which are the inputs to $F()$.

b. In order to compute the derivatives ∂d, ∂e and ∂f, you first need ∂a, ∂b and ∂c. So, if we have a sequence of operations:

$$[a, b, c] = F(d, e, f)$$
$$[u, v, w] = G(a, b, c)$$

then we must first compute derivatives for $\partial a, \partial b, \partial c$ from $\partial u, \partial v, \partial w$,and then use those to compute $\partial d, \partial e, \partial f$. So to compute the derivatives with respect to the earliest variables in the sequence of operations, we must compute them in *reverse* order, starting with the last operation, and then working our way backwards.

Now we are set to go.