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.

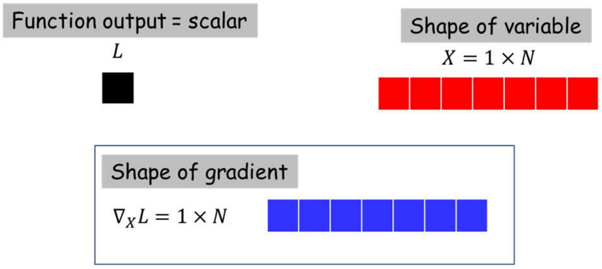
Python considers row vectors by default, so all our rules will be transposed, which leads to the following rules for derivatives.

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.

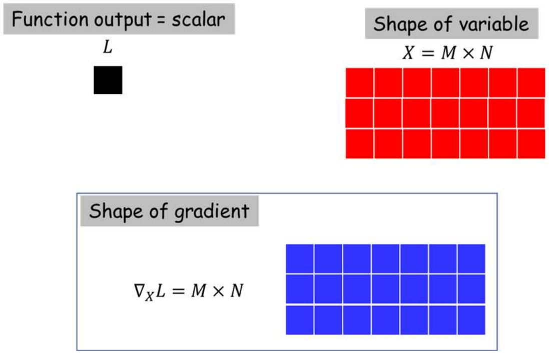
Figure: Illustration of variable dimensions for $L = f(X)$ with scalar L and row vector X



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.

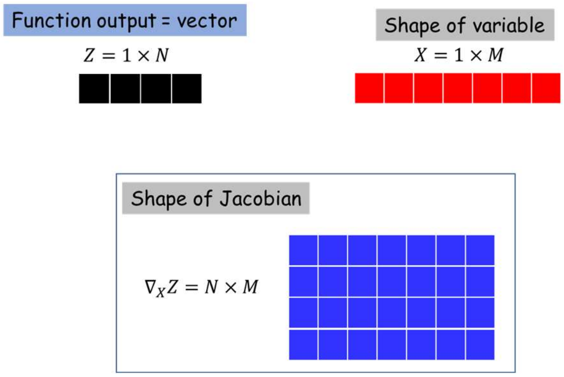
Figure: Illustration of variable dimensions for $L = f(X)$ with scalar L and $M \times N$ matrix X



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 $1 \times M$



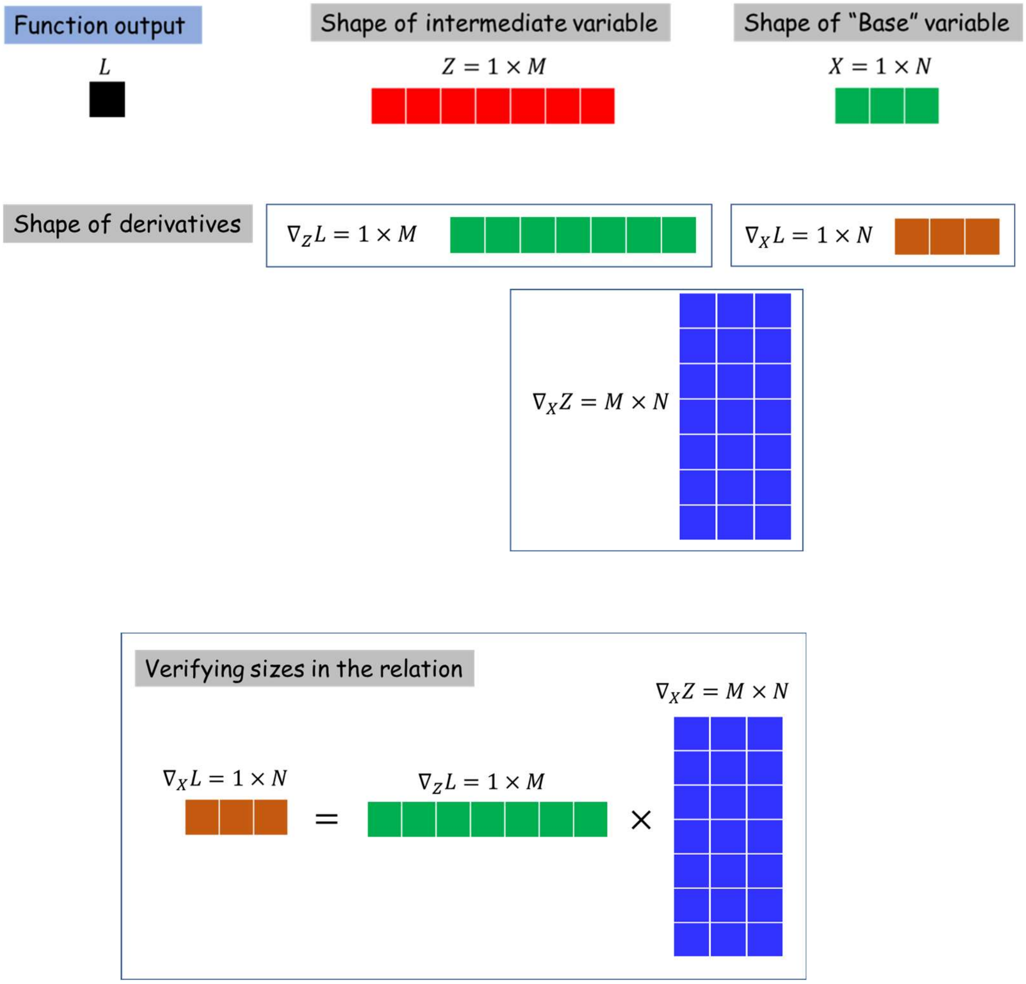
The (i, j)th element of $\nabla_X Z$ is the partial derivative $\frac{\partial Z_i}{\partial X_j}$. Note the indices: The numerator index in the partial derivative corresponds to the *first* (row) index in the Jacobian matrix $\nabla_X Z$, and the denominator index corresponds to the *second* (column) 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
$\nabla_X L = 1 \times N$
$=$
$\nabla_Z L = 1 \times M$
$\times$
$\nabla_X Z = M \times N$

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.

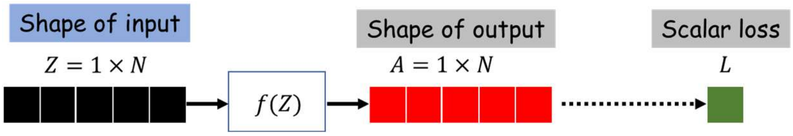
Applying it to an MLP

Let us apply the above rules to two of the main operations in an MLP: (a) Computing the affine function, aka the linear layer and (b) Applying the activation aka the activation layer.

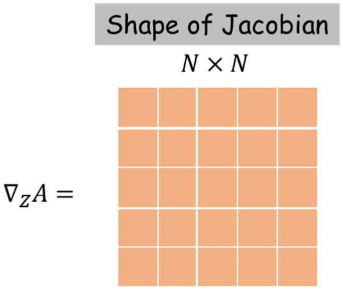
Backpropagating through an activation.

In forward computation, at an activation layer, a $1 \times N$ vector if affine values Z is put through an activation function $f()$ to get an activation output $A = f(Z)$. A will also be a $1 \times N$ vector. The activation output A is then propagated forward through the network, to compute the network output, from which the loss L is computed.

In terms of sizes, the computation will look as follows



From our above rules, the Jacobian of $A = f()$ with respect to Z will be an $N \times N$ matrix.



If the activation is a component-wise activation function like a RELU, sigmoid, tanh etc, then the Jacobian will be a diagonal matrix where the ith diagonal element is

$$(\nabla_Z A)_{i,i} = \frac{\mathrm{d}f(Z_i)}{\mathrm{d}Z_i}$$

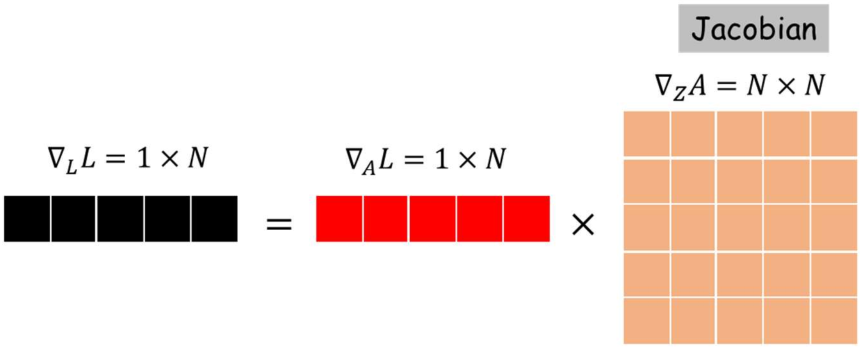
and the off diagonal elements are all 0, i.e. $(\nabla_Z A)_{i,j} = 0\ for\ i \neq j$

For “vector” activations such as the softmax, in which changing any component of the input changes all output components, the Jacobian will be a full matrix.

When we backpropagate loss gradients through an activation, we would already have obtained the derivative $\nabla_Z L$ which will be a $1 \times N$ vector. The chain rule then gives us the following

$$\nabla_Z L = \nabla_A L \, \nabla_Z A$$

We can verify the sizes for the above


Jacobian
$\nabla_Z A = N \times N$
$\nabla_L L = 1 \times N$
$=$
$\nabla_A L = 1 \times N$
$\times$

Vector activations:

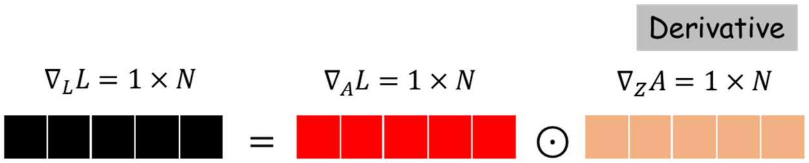
For vector activations like the softmax, the Jacobian is a full matrix and the above equation can be directly used.

Scalar activations:

For *scalar* component-wise activations such as RELU or tanh, the Jacobian is a diagonal matrix and constructing the full matrix and performing the above computation can be wasteful.

In this case it is simper to just consider the diagonal terms of the Jacobian, which can be arranged as a $1\ \times N$ row vector $\nabla_Z A = \left[\frac{dA_1}{dZ_1}, \frac{dA_2}{dZ_2}, \cdots, \frac{dA_N}{dZ_\mathrm{n}}\right]$. In this case the derivative will just be a component-wise multiplication:

$$\nabla_Z L = \nabla_A L \odot \nabla_Z A$$


Derivative
$\nabla_L L = 1 \times N$
$=$
$\nabla_A L = 1 \times N$
$\odot$
$\nabla_Z A = 1 \times N$

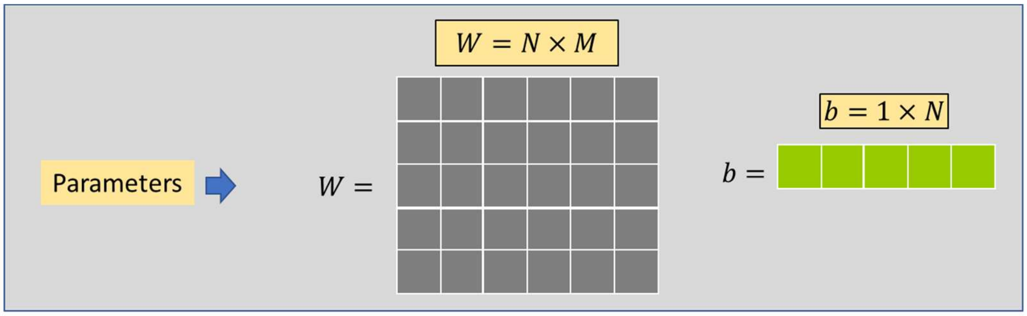
Backpropagating through a linear layer.

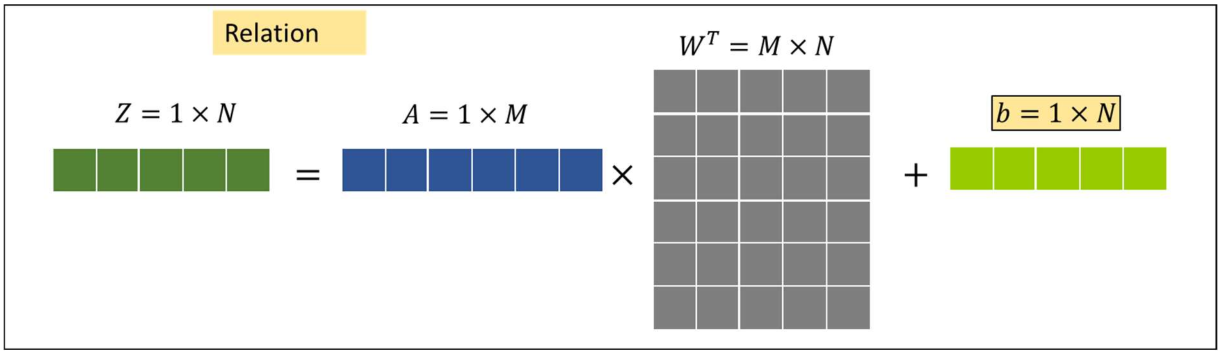
A "linear" layer is just an affine function and has the form

$$Z = AW^T + b$$

Where A is the output of the previous layer, W is the matrix of weights, and b is the vector of biases. If the previous layer has M neurons and the current layer has N neurons, then A is a $1 \times M$ row vector, Z is a $1 \times N$ row vector, W is a $N \times M$ matrix (with each row representing the incoming weights to a neuron), and b is a $1 \times M$ row vector.

In terms of shapes, the relation looks as follows.





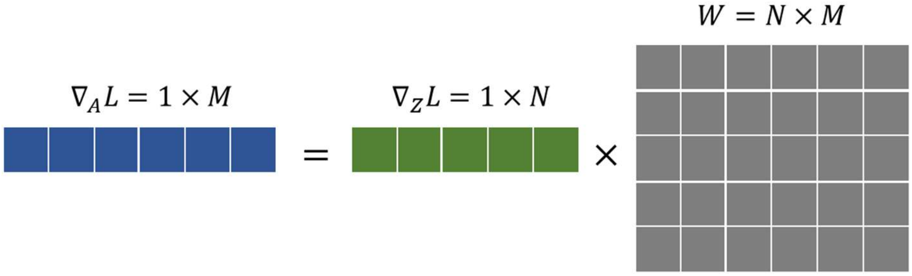
When propagating derivatives backward, we will assume we have $\nabla_Z L$, the gradient of the loss w.r.t. Z. This will be a $1 \times N$ row vector (same shape as Z).

By our definition, the gradient w.r.t. A must be a $1 \times M$ row vector (same shape as A), the gradient w.r.t. b must be a $1 \times N$ row vector (same shape as b), and the gradient w.r.t. W must be an $N \times M$ matrix (same shape as W).

This leads to the following relations. Please verify that the sizes match.

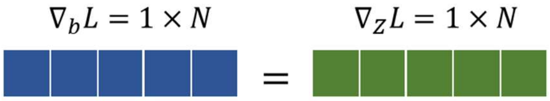
$$\nabla_A L = \nabla_Z L \cdot \nabla_A Z$$

$$\nabla_A L = \nabla_Z L \cdot \mathrm{W} \qquad (1)$$



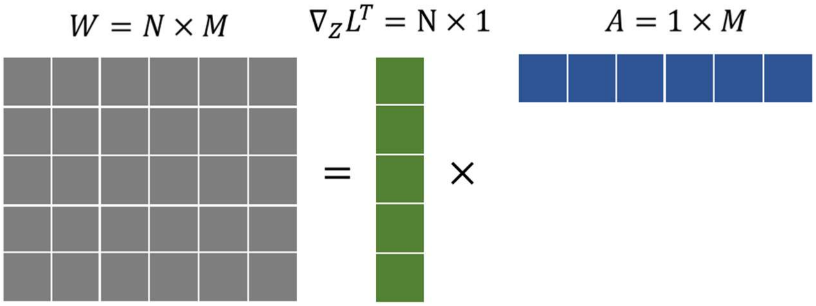
$$\nabla_b L = \nabla_Z L \ \cdot \nabla_{\mathrm{b}} Z$$

$$\nabla_b L = \nabla_Z L \tag{2}$$



$$\nabla_W L = \nabla_Z L^T \cdot \nabla_W Z$$

$$\nabla_W L = \nabla_Z L^T \cdot A \tag{3}$$



Now we are set to go.