Derivatives, Backpropagation, and Vectorization

Justin Johnson

September 6, 2017

1 Derivatives

1.1 Scalar Case

You are probably familiar with the concept of a derivative in the scalar case: given a function $f : \mathbb{R} \to \mathbb{R}$, the *derivative* of f at a point $x \in \mathbb{R}$ is defined as:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Derivatives are a way to measure *change*. In the scalar case, the derivative of the function f at the point x tells us how much the function f changes as the input x changes by a small amount ε:

$$f(x + \varepsilon) \approx f(x) + \varepsilon f'(x)$$

For ease of notation we will commonly assign a name to the output of f, say $y = f(x)$, and write $\frac{\partial y}{\partial x}$ for the derivative of y with respect to x. This notation emphasizes that $\frac{\partial y}{\partial x}$ is the *rate of change* between the variables x and y; concretely if x were to change by ε then y will change by approximately $\varepsilon \frac{\partial y}{\partial x}$. We can write this relationship as

$$x \to x + \Delta x \implies y \to \approx y + \frac{\partial y}{\partial x} \Delta x$$

You should read this as saying "changing x to $x + \Delta x$ implies that y will change to approximately $y + \Delta x \frac{\partial y}{\partial x}$". This notation is nonstandard, but I like it since it emphasizes the relationship between changes in x and changes in y.

The *chain rule* tells us how to compute the derivative of the compositon of functions. In the scalar case suppose that $f, g : \mathbb{R} \to \mathbb{R}$ and $y = f(x)$, $z = g(y)$; then we can also write $z = (g \circ f)(x)$, or draw the following *computational graph*:

$$x \xrightarrow{f} y \xrightarrow{g} z$$

The (scalar) chain rule tells us that

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$$

This equation makes intuitive sense. The derivatives $\frac{\partial z}{\partial y}$ and $\frac{\partial y}{\partial x}$ give:

$$x \to x + \Delta x \implies y \to\approx y + \frac{\partial y}{\partial x}\Delta x$$
$$y \to y + \Delta y \implies z \to\approx z + \frac{\partial z}{\partial y}\Delta y$$

Combining these two rules lets us compute the effect of x on z: if x changes by Δx then y will change by $\frac{\partial y}{\partial x}\Delta x$, so we have $\Delta y = \frac{\partial y}{\partial x}\Delta x$. If y changes by Δy then z will change by $\frac{\partial z}{\partial y}\Delta y = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}\Delta x$ which is exactly what the chain rule tells us.

1.2 Gradient: Vector in, scalar out

This same intuition carries over into the vector case. Now suppose that $f : \mathbb{R}^N \to \mathbb{R}$ takes a vector as input and produces a scalar. The derivative of f at the point $x \in \mathbb{R}^N$ is now called the *gradient*, and it is defined as:

$$\nabla_x f(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{\|h\|}$$

Now the gradient $\nabla_x f(x) \in \mathbb{R}^N$ is a vector, with the same intuition as the scalar case. If we set $y = f(x)$ then we have the relationship

$$x \to x + \Delta x \implies y \to\approx y + \frac{\partial y}{\partial x} \cdot \Delta x$$

The formula changes a bit from the scalar case to account for the fact that $x, \Delta x$, and $\frac{\partial y}{\partial x}$ are now *vectors* in $\mathbb{R}^N$ while y is a scalar. In particular when multiplying $\frac{\partial y}{\partial x}$ by Δx we use the *dot product*, which combines two vectors to give a scalar.

One nice outcome of this formula is that it gives meaning to the individual elements of the gradient $\frac{\partial y}{\partial x}$. Suppose that Δx is the ith basis vector, so that the ith coordinate of ε is 1 and all other coordinates of ε are 0. Then the dot product $\frac{\partial y}{\partial x} \cdot \Delta x$ is simply the ith coordinate of $\frac{\partial y}{\partial x}$; thus the ith coordinate of $\frac{\partial y}{\partial x}$ tells us the approximate amount by which y will change if we move x along the ith coordinate axis.

This means that we can also view the gradient $\frac{\partial y}{\partial x}$ as a vector of partial derivatives:

$$\frac{\partial y}{\partial x} = \left(\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, \ldots, \frac{\partial y}{\partial x_N}\right)$$

where x_i is the ith coordinate of the vector x, which is a scalar, so each partial derivative $\frac{\partial y}{\partial x_i}$ is also a scalar.

1.3 Jacobian: Vector in, Vector out

Now suppose that $f : \mathbb{R}^N \to \mathbb{R}^M$ takes a vector as input and produces a vector as output. Then the derivative of f at a point x, also called the *Jacobian*, is the $M \times N$ matrix of partial derivatives. If we again set $y = f(x)$ then we can write:

$$\frac{\partial y}{\partial x} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_N} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_M}{\partial x_1} & \cdots & \frac{\partial y_M}{\partial x_N} \end{pmatrix}$$

The Jacobian tells us the relationship between each element of x and each element of y: the (i, j)-th element of $\frac{\partial y}{\partial x}$ is equal to $\frac{\partial y_i}{\partial x_j}$, so it tells us the amount by which y_i will change if x_j is changed by a small amount.

Just as in the previous cases, the Jacobian tells us the relationship between changes in the input and changes in the output:

$$x \to x + \Delta x \implies y \to\approx y + \frac{\partial y}{\partial x}\Delta x$$

Here $\frac{\partial y}{\partial x}$ is a $M \times N$ matrix and Δx is an N-dimensional vector, so the product $\frac{\partial y}{\partial x}\Delta x$ is a matrix-vector multiplication resulting in an M-dimensional vector.

The chain rule can be extended to the vector case using Jacobian matrices. Suppose that $f : \mathbb{R}^N \to \mathbb{R}^M$ and $g : \mathbb{R}^M \to \mathbb{R}^K$. Let $x \in \mathbb{R}^N$, $y \in \mathbb{R}^M$, and $z \in \mathbb{R}^K$ with $y = f(x)$ and $z = g(y)$, so we have the same computational graph as the scalar case:

$$x \xrightarrow{f} y \xrightarrow{g} z$$

The chain rule also has the same form as the scalar case:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

However now each of these terms is a matrix: $\frac{\partial z}{\partial y}$ is a $K \times M$ matrix, $\frac{\partial y}{\partial x}$ is a $M \times N$ matrix, and $\frac{\partial z}{\partial x}$ is a $K \times N$ matrix; the multiplication of $\frac{\partial z}{\partial y}$ and $\frac{\partial y}{\partial x}$ is matrix multiplication.

1.4 Generalized Jacobian: Tensor in, Tensor out

Just as a vector is a one-dimensional list of numbers and a matrix is a two-dimensional grid of numbers, a *tensor* is a D-dimensional grid of numbers[1].

Many operations in deep learning accept tensors as inputs and produce tensors as outputs. For example an image is usually represented as a three-dimensional grid of numbers, where the three dimensions correspond to the height, width, and color channels (red, green, blue) of the image. We must therefore develop a derivative that is compatible with functions operating on general tensors.

Suppose now that $f : \mathbb{R}^{N_1 \times \cdots \times N_{D_x}} \to \mathbb{R}^{M_1 \times \cdots \times M_{D_y}}$. Then the input to f is a D_x-dimensional tensor of shape $N_1 \times \cdots \times N_{D_x}$, and the output of f is a D_y-dimensional tensor of shape $M_1 \times \cdots \times M_{D_y}$. If $y = f(x)$ then the derivative $\frac{\partial y}{\partial x}$ is a *generalized Jacobian*, which is an object with shape

$$(M_1 \times \cdots \times M_{D_y}) \times (N_1 \times \cdots \times N_{D_x})$$

Note that we have separated the dimensions of $\frac{\partial y}{\partial x}$ into two groups: the first group matches the dimensions of y and the second group matches the dimensions of x. With this grouping, we can think of the generalized Jacobian as generalization of a matrix, where each "row" has the same shape as y and each "column" has the same shape as x.

Now if we let $i \in \mathbb{Z}^{D_y}$ and $j \in \mathbb{Z}^{D_x}$ be vectors of integer indices, then we can write

$$\left(\frac{\partial y}{\partial x}\right)_{i,j} = \frac{\partial y_i}{\partial x_j}$$

In this equation note that y_i and x_j are scalars, so the derivative $\frac{\partial y_i}{\partial x_j}$ is also a scalar. Using this notation we see that like the standard Jacobian, the generalized Jacobian tells us the relative rates of change between all elements of x and all elements of y.

The generalized Jacobian gives the same relationship between inputs and outputs as before:

$$x \to x + \Delta x \implies y \to\approx y + \frac{\partial y}{\partial x}\Delta x$$

The difference is that now Δx is a tensor of shape $N_1 \times \cdots \times N_{D_x}$ and $\frac{\partial y}{\partial x}$ is a generalized matrix of shape $(M_1 \times \cdots \times M_{D_y}) \times (N_1 \times \cdots \times N_{D_x})$. The product $\frac{\partial y}{\partial x}\Delta x$ is therefore a *generalized matrix-vector multiply*, which results in a tensor of shape $M_1 \times \cdots \times M_{D_y}$.

The generalized matrix-vector multipy follows the same algebraic rules as a traditional matrix-vector multiply:

[1]The word *tensor* is used in different ways in different fields; you may have seen the term before in physics or abstract algebra. The machine learning definition of a tensor as a D-dimensional grid of numbers is closely related to the definitions of tensors in these other fields.

$$\left(\frac{\partial y}{\partial x}\Delta x\right)_j = \sum_i \left(\frac{\partial y}{\partial x}\right)_{i,j} (\Delta x)_i = \left(\frac{\partial y}{\partial x}\right)_{j,:} \cdot \Delta x$$

The only difference is that the indicies i and j are not scalars; instead they are vectors of indicies. In the equation above the term $\left(\frac{\partial y}{\partial x}\right)_{j,:}$ is the jth "row" of the generalized matrix $\frac{\partial y}{\partial x}$, which is a tensor with the same shape as x. We have also used the convention that the dot product between two tensors of the same shape is an elementwise product followed by a sum, identical to the dot product between vectors.

The chain rule also looks the same in the case of tensor-valued functions. Suppose that $y = f(x)$ and $z = g(y)$, where x and y have the same shapes as above and z has shape $K_1 \times \cdots \times K_{D_z}$. Now the chain rule looks the same as before:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$$

The difference is that now $\frac{\partial z}{\partial y}$ is a generalized matrix of shape $(K_1 \times \cdots \times K_{D_z}) \times (M_1 \times \cdots \times M_{D_y})$, and $\frac{\partial y}{\partial z}$ is a generalized matrix of shape $(M_1 \times \cdots \times M_{D_y}) \times (N_1 \times \cdots \times N_{D_x})$; the product $\frac{\partial z}{\partial y}\frac{\partial y}{\partial x}$ is a generalized matrix-matrix multiply, resulting in an object of shape $(K_1 \times \cdots \times K_{D_z}) \times (N_1 \times \cdots \times N_{D_x})$. Like the generalized matrix-vector multiply defined above, the generalized matrix-matrix multiply follows the same algebraic rules as the traditional matrix-matrix multiply:

$$\left(\frac{\partial z}{\partial x}\right)_{i,j} = \sum_k \left(\frac{\partial z}{\partial y}\right)_{i,k} \left(\frac{\partial y}{\partial x}\right)_{k,j} = \left(\frac{\partial z}{\partial y}\right)_{i,:} \cdot \left(\frac{\partial y}{\partial x}\right)_{:,j}$$

In this equation the indices i, j, k are vectors of indices, and the terms $\left(\frac{\partial z}{\partial y}\right)_{i,:}$ and $\left(\frac{\partial y}{\partial x}\right)_{:,j}$ are the ith "row" of $\frac{\partial z}{\partial y}$ and the jth "column" of $\frac{\partial y}{\partial x}$ respectively.

2 Backpropagation with Tensors

In the context of neural networks, a layer f is typically a function of (tensor) inputs x and weights w; the (tensor) output of the layer is then $y = f(x, w)$. The layer f is typically embedded in some large neural network with scalar loss L.

During backpropagation, we assume that we are given $\frac{\partial L}{\partial y}$ and our goal is to compute $\frac{\partial L}{\partial x}$ and $\frac{\partial L}{\partial w}$. By the chain rule we know that

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x} \qquad\qquad \frac{\partial L}{\partial w} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial w}$$

Therefore one way to proceed would be to form the (generalized) Jacobians $\frac{\partial y}{\partial x}$ and $\frac{\partial y}{\partial w}$ and use (generalized) matrix multiplication to compute $\frac{\partial L}{\partial x}$ and $\frac{\partial L}{\partial w}$.

However, there's a problem with this approach: the Jacobian matrices $\frac{\partial y}{\partial x}$ and $\frac{\partial y}{\partial w}$ are typically far too large to fit in memory.

As a concrete example, let's suppose that f is a linear layer that takes as input a minibatch of N vectors, each of dimension D, and produces a minibatch of N vectors, each of dimension M. Then x is a matrix of shape $N \times D$, w is a matrix of shape $D \times M$, and $y = f(x, w) = xw$ is a matrix of shape $N \times M$.

The Jacobian $\frac{\partial y}{\partial x}$ then has shape $(N \times M) \times (N \times D)$. In a typical neural network we might have $N = 64$ and $M = D = 4096$; then $\frac{\partial y}{\partial x}$ consists of $64 \cdot 4096 \cdot 64 \cdot 4096$ scalar values; this is more than 68 billion numbers; using 32-bit floating point, this Jacobian matrix will take 256 GB of memory to store. Therefore it is completely hopeless to try and explicitly store and manipulate the Jacobian matrix.

However it turns out that for most common neural network layers, we can derive expressions that compute the product $\frac{\partial y}{\partial x}\frac{\partial L}{\partial y}$ *without explicitly forming the Jacobian* $\frac{\partial y}{\partial x}$. Even better, we can typically derive this expression without even computing an explicit expression for the Jacobian $\frac{\partial y}{\partial x}$; in many cases we can work out a small case on paper and then infer the general formula.

Let's see how this works out for the case of the linear layer $f(x, w) = xw$. Set $N = 1$, $D = 2$, $M = 3$. Then we can explicitly write

$$y = \begin{pmatrix} y_{1,1} & y_{1,2} & y_{1,3} \end{pmatrix} = xw \tag{1}$$

$$= \begin{pmatrix} x_{1,1} & x_{1,2} \end{pmatrix} \begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} \\ w_{2,1} & w_{2,2} & w_{2,3} \end{pmatrix} \tag{2}$$

$$= \begin{pmatrix} x_{1,1}w_{1,1} + x_{1,2}w_{2,1} \\ x_{1,1}w_{1,2} + x_{1,2}w_{2,2} \\ x_{1,1}w_{1,3} + x_{1,2}w_{2,3} \end{pmatrix}^T \tag{3}$$

During backpropagation we assume that we have access to $\frac{\partial L}{\partial y}$ which technically has shape $(1) \times (N \times M)$; however for notational convenience we will instead think of it as a matrix of shape $N \times M$. Then we can write

$$\frac{\partial L}{\partial y} = \begin{pmatrix} dy_{1,1} & dy_{1,2} & dy_{1,3} \end{pmatrix}$$

Our goal now is to derive an expression for $\frac{\partial L}{\partial x}$ in terms of x, w, and $\frac{\partial L}{\partial y}$, without explicitly forming the entire Jacobian $\frac{\partial y}{\partial x}$. We know that $\frac{\partial L}{\partial x}$ will have shape $(1) \times (N \times D)$, but as is typical for representing gradients we instead view $\frac{\partial L}{\partial x}$ as a matrix of shape $N \times D$. We know that each element of $\frac{\partial L}{\partial x}$ is a scalar giving the partial derivatives of L with respect to the elements of x:

$$\frac{\partial L}{\partial x} = \begin{pmatrix} \frac{\partial L}{\partial x_{1,1}} & \frac{\partial L}{\partial x_{1,2}} \end{pmatrix}$$

Thinking one element at a time, the chain rule tells us that

$$\frac{\partial L}{\partial x_{1,1}} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x_{1,1}} \tag{4}$$

$$\frac{\partial L}{\partial x_{1,2}} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial x_{1,2}} \tag{5}$$

Viewing these derivatives as generalized matrices, $\frac{\partial L}{\partial y}$ has shape $(1)\times(N\times M)$ and $\frac{\partial y}{\partial x_{1,1}}$ has shape $(N\times M)\times(1)$; their product $\frac{\partial L}{\partial x_{1,1}}$ then has shape $(1)\times(1)$. If we instead view $\frac{\partial L}{\partial y}$ and $\frac{\partial y}{\partial x_{1,1}}$ as matrices of shape $N\times M$, then their generalized matrix product is simply the dot product $\frac{\partial L}{\partial y}\cdot\frac{\partial y}{\partial x_{1,1}}$.

Now we compute

$$\frac{\partial y}{\partial x_{1,1}} = \begin{pmatrix} \frac{\partial y_{1,1}}{\partial x_{1,1}} & \frac{\partial y_{1,2}}{\partial x_{1,1}} & \frac{\partial y_{1,3}}{\partial x_{1,1}} \end{pmatrix} = \begin{pmatrix} w_{1,1} & w_{1,2} & w_{1,3} \end{pmatrix} \tag{6}$$

$$\frac{\partial y}{\partial x_{1,2}} = \begin{pmatrix} \frac{\partial y_{1,1}}{\partial x_{1,2}} & \frac{\partial y_{1,2}}{\partial x_{1,2}} & \frac{\partial y_{1,3}}{\partial x_{1,2}} \end{pmatrix} = \begin{pmatrix} w_{2,1} & w_{2,2} & w_{2,3} \end{pmatrix} \tag{7}$$

where the final equality comes from taking the derivatives of Equation 3 with respect to $x_{1,1}$.

We can now combine these results and write

$$\frac{\partial L}{\partial x_{1,1}} = \frac{\partial L}{\partial y}\cdot\frac{\partial y}{\partial x_{1,1}} = dy_{1,1}w_{1,1} + dy_{1,2}w_{1,2} + dy_{1,3}w_{1,3} \tag{8}$$

$$\frac{\partial L}{\partial x_{1,2}} = \frac{\partial L}{\partial y}\cdot\frac{\partial y}{\partial x_{1,2}} = dy_{1,1}w_{2,1} + dy_{1,2}w_{2,2} + dy_{1,3}w_{2,3} \tag{9}$$

This gives us our final expression for $\frac{\partial L}{\partial x}$:

$$\frac{\partial L}{\partial x} = \begin{pmatrix} \frac{\partial L}{\partial x_{1,1}} & \frac{\partial L}{\partial x_{1,2}} \end{pmatrix} \tag{10}$$

$$= \begin{pmatrix} dy_{1,1}w_{1,1} + dy_{1,2}w_{1,2} + dy_{1,3}w_{1,3} \\ dy_{1,1}w_{2,1} + dy_{1,2}w_{2,2} + dy_{1,3}w_{2,3} \end{pmatrix}^T \tag{11}$$

$$= \frac{\partial L}{\partial y}x^T \tag{12}$$

This final result $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial y}x^T$ is very interesting because it allows us to efficiently compute $\frac{\partial L}{\partial x}$ without explicitly forming the Jacobian $\frac{\partial y}{\partial x}$. We have only derived this formula for the specific case of $N = 1, D = 2, M = 3$ but it in fact holds in general.

By a similar thought process we can derive a similar expression for $\frac{\partial L}{\partial w}$ without explicitly forming the Jacobian $\frac{\partial y}{\partial w}$. You should try and work through this as an exercise.