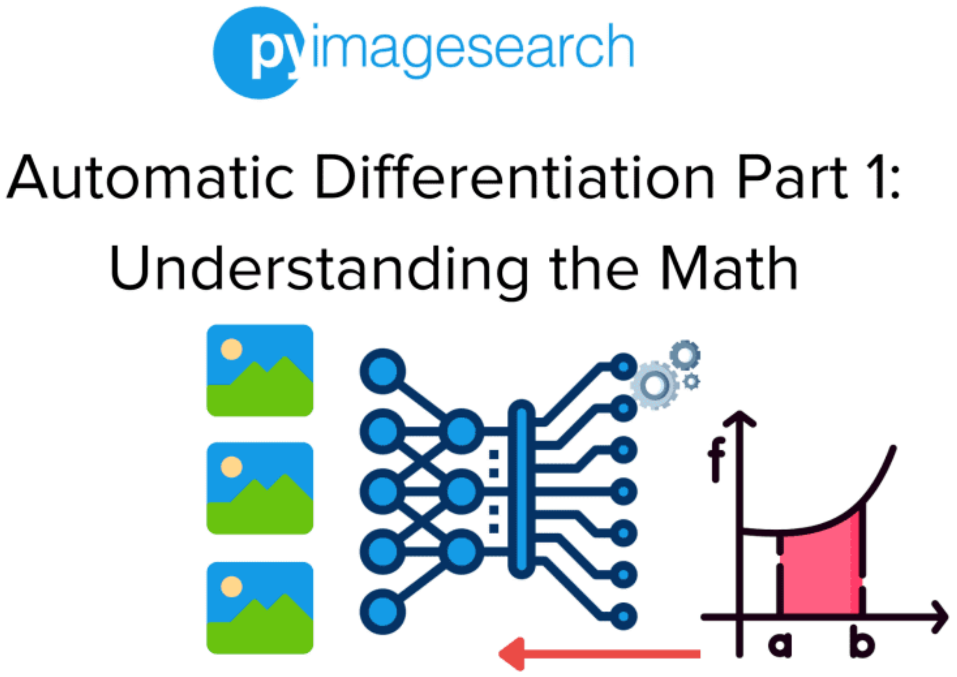
Automatic Differentiation Part 1: Understanding the Math

Table of Contents

In this tutorial, you will learn **the math behind automatic differentiation needed for backpropagation**.



This lesson is the 1st of a 2-part series on **Autodiff 101 — Understanding Automatic Differentiation from Scratch**:

1. ***Automatic Differentiation Part 1: Understanding the Math*** **(this tutorial)**
2. *Automatic Differentiation Part 2: Implementation Using Micrograd*

To learn about automatic differentiation, *just keep reading.*

Automatic Differentiation Part 1: Understanding the Math

Imagine you are trekking down a hill. It is dark, and there are a lot of bumps and turns. You have no way of knowing how to reach the center. Now imagine every time you progress, you have to pause, take out the topological map of the hill and calculate your direction and speed for the next set. Sounds painfully less fun, right?

If you have been a reader of our tutorials, you would know what that analogy refers to. The hill is your loss landscape, the topological map is the set of rules for multivariate calculus, and you are the parameters of the neural network. The objective is to reach the global minimum.

And that brings us to the question:

> *Why do we use a Deep Learning Framework today?*

The first thing that pops into the mind is **automatic differentiation**. We write the forward pass, and that is it; no need to worry about the backward pass. Every operator is automatically differentiated and is waiting to be used in an optimization algorithm (like stochastic gradient descent).

Today in this tutorial, we will walk through the valleys of automatic differentiation.

Introduction

In this section, we will lay out the foundation necessary for understanding
autodiff
`autodiff`.

Jacobian

Let’s consider a function

$F : \mathbb{R}^n \rightarrow \mathbb{R}$
. is a multivariate function that simultaneously depends on multiple variables. Here the multiple variables can be

$x = \{x_1, x_2, \ldots, x_n\}$
. The output of the function is a **scalar value**. This can be considered as a neural network that takes an image and outputs the probability of a dog’s presence in the image.

Note**:** Let us recall that in a neural network, we compute gradients with respect to the parameters (weights and biases) and not the inputs (the image). Thus the domain of the function is the parameters and not the inputs, which helps keep the gradient computation accessible. We need to now think of everything we do in this tutorial from the perspective of making it **simple** and **efficient** to obtain the gradients with respect to the weights and biases (*parameters*). This is illustrated in **Figure 1**.

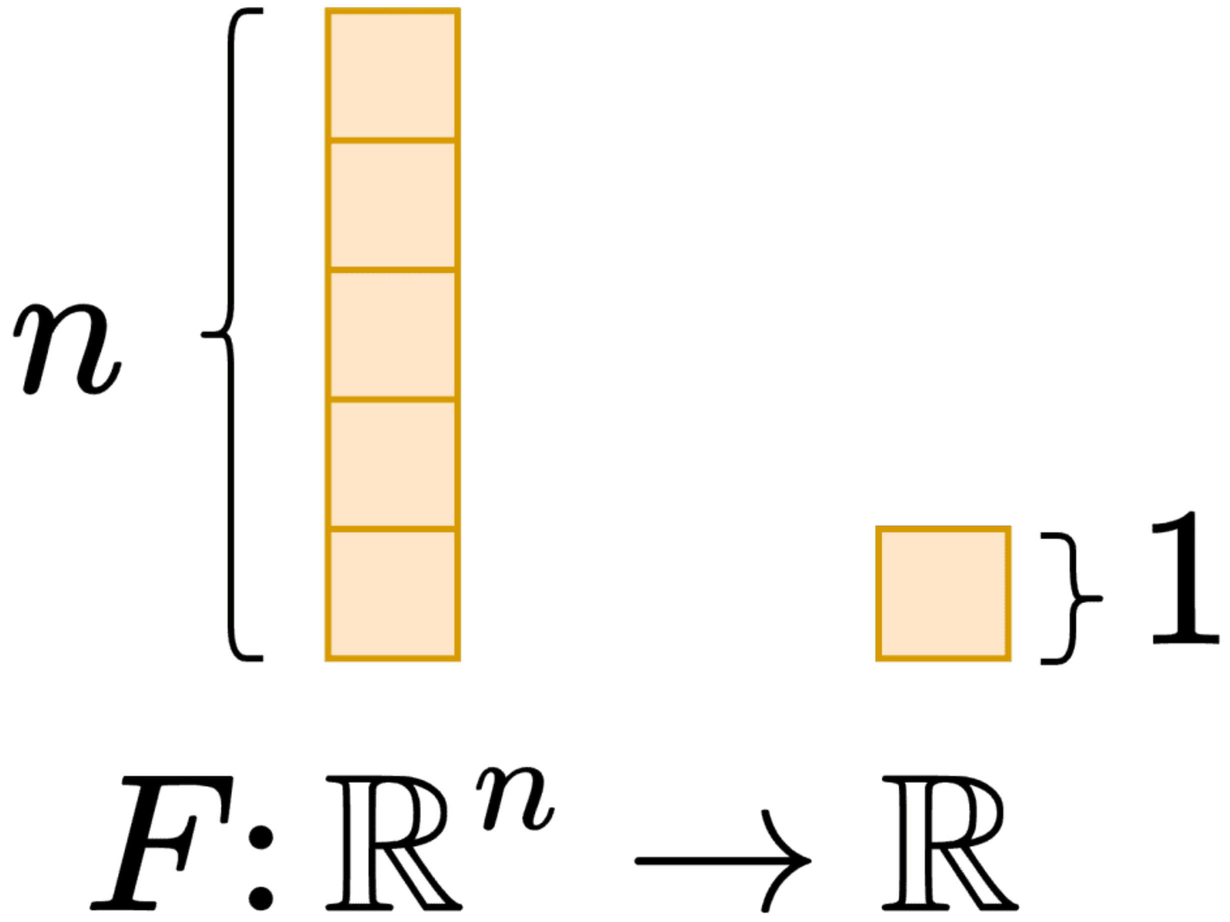


Figure 1: *Domain of the function from the perspective of a neural network (source: image by the authors).*

A neural network is a composition of many sublayers. So let's consider our function
$F(x)$
as a composition of multiple functions (primitive operations).

$$F(x) \;=\; D \circ C \circ B \circ A$$

The function
$F(x)$
is composed of four primitive functions, namely

$D, C, B, \text{ and } A$
. For anyone new to composition, we can call
$F(x)$
to be a function where

$D(C(B(A(x))))$
is equal to
$F(x)$
.

The next step would be to find the gradient of
$F(x)$

. However, before diving into the gradients of the function, let us revisit Jacobian matrices. It turns out that the derivatives of a multivariate function are a Jacobian matrix consisting of partial derivatives of the function w.r.t. all the variables upon which it depends.

Consider two multivariate functions,
u
and
v
, which depend on the variables
x
and
y
. The Jacobian would look like this:

$$\frac{\partial(u,v)}{\partial x, y} = \begin{bmatrix} \frac{\partial u}{\partial x} & \frac{\partial u}{\partial y} \\ \frac{\partial v}{\partial x} & \frac{\partial v}{\partial y} \end{bmatrix}$$

Now let’s compute the Jacobian of our function
$F(x)$
. We need to note here that the function depends of variables

$x = \{x_1, x_2, \ldots, x_n\}$
, and outputs a scalar value. This means that the Jacobian will be a row vector.

$$F'(x) = \frac{\partial y}{\partial x} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \ldots & \frac{\partial y}{\partial x_n} \end{bmatrix}$$

Chain Rule

Remember how our function
$F(x)$
is composed of many primitive functions? The derivative of such a composed function is done with the help of the chain rule. To help our way into the chain rule, let us first write down the composition and then define the intermediate values.

$F(x) = D(C(B(A(x))))$
is composed of:

- $y = D(c)$
- $c = C(b)$
- $b = B(a)$
- $a = A(x)$

Now that the composition is spelled out, let’s first get the derivatives of the intermediate values.

- $D'(c) = \frac{\partial y}{\partial c}$
- $C'(b) = \frac{\partial c}{\partial b}$

- $B'(a) = \frac{\partial b}{\partial a}$
- $A'(x) = \frac{\partial a}{\partial x}$

Now with the help of the chain rule, we derive the derivative of the function $F(x)$.

$$F'(x) \;=\; \frac{\partial y}{\partial c}\frac{\partial c}{\partial b}\frac{\partial b}{\partial a}\frac{\partial a}{\partial x}$$

Mix the Jacobian and Chain Rule

After knowing about the Jacobian and the Chain Rule, let us visualize the two together. Shown in **Figure 2**.

$$F'(x) \;=\; \frac{\partial y}{\partial x} \;=\; \begin{bmatrix} \frac{\partial y}{\partial x_1} & \dots & \frac{\partial y}{\partial x_n} \end{bmatrix}$$

$$F'(x) \;=\; \frac{\partial y}{\partial c}\frac{\partial c}{\partial b}\frac{\partial b}{\partial a}\frac{\partial a}{\partial x}$$

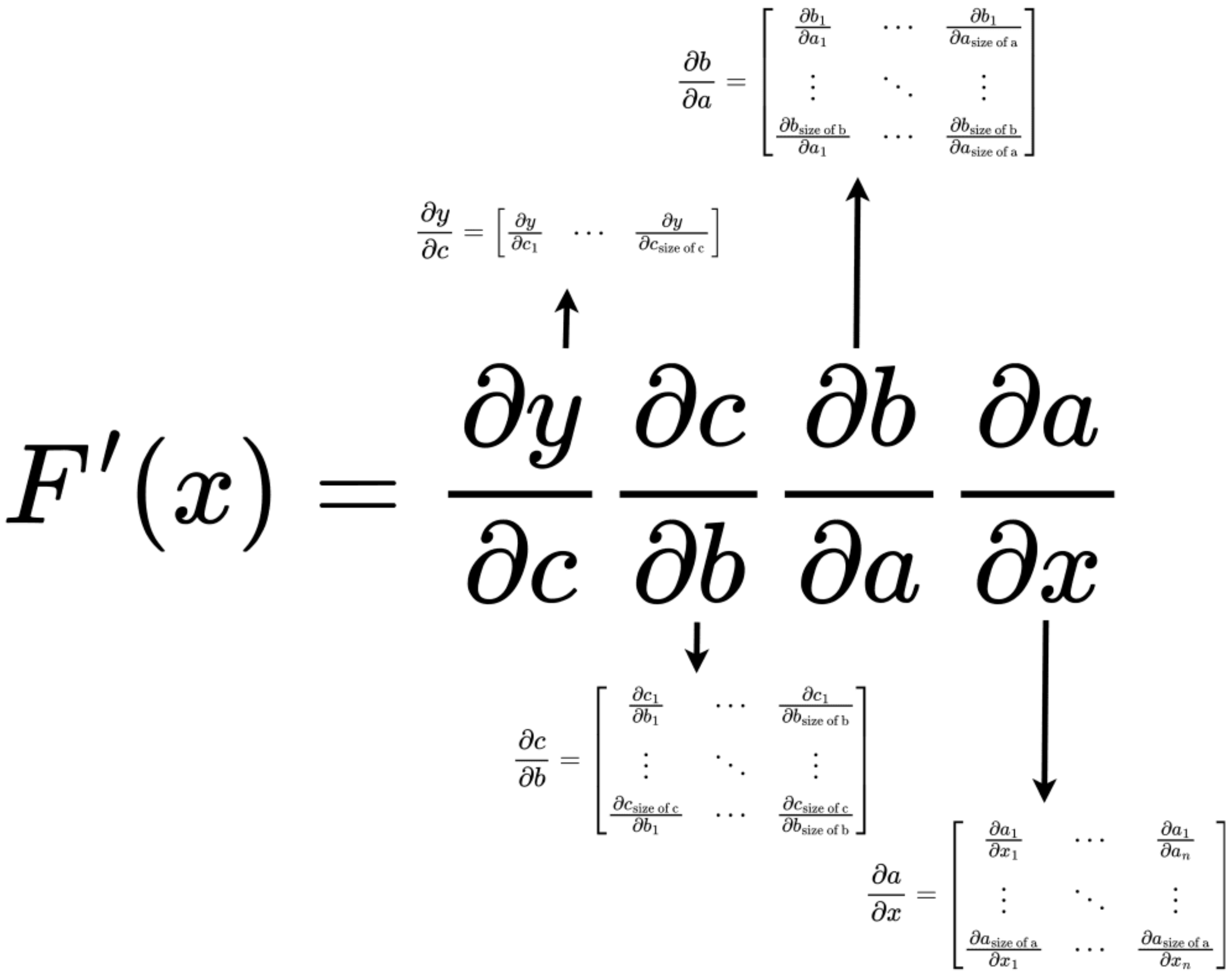

$\frac{\partial b}{\partial a} = \begin{bmatrix} \frac{\partial b_1}{\partial a_1} & \dots & \frac{\partial b_1}{\partial a_{\text{size of a}}} \\ \vdots & \ddots & \vdots \\ \frac{\partial b_{\text{size of b}}}{\partial a_1} & \dots & \frac{\partial b_{\text{size of b}}}{\partial a_{\text{size of a}}} \end{bmatrix}$
$\frac{\partial y}{\partial c} = \begin{bmatrix} \frac{\partial y}{\partial c_1} & \dots & \frac{\partial y}{\partial c_{\text{size of c}}} \end{bmatrix}$
$F'(x) = \frac{\partial y}{\partial c}\frac{\partial c}{\partial b}\frac{\partial b}{\partial a}\frac{\partial a}{\partial x}$
$\frac{\partial c}{\partial b} = \begin{bmatrix} \frac{\partial c_1}{\partial b_1} & \dots & \frac{\partial c_1}{\partial b_{\text{size of b}}} \\ \vdots & \ddots & \vdots \\ \frac{\partial c_{\text{size of c}}}{\partial b_1} & \dots & \frac{\partial c_{\text{size of c}}}{\partial b_{\text{size of b}}} \end{bmatrix}$
$\frac{\partial a}{\partial x} = \begin{bmatrix} \frac{\partial a_1}{\partial x_1} & \dots & \frac{\partial a_1}{\partial a_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial a_{\text{size of a}}}{\partial x_1} & \dots & \frac{\partial a_{\text{size of a}}}{\partial x_n} \end{bmatrix}$

Figure 2: *Jacobian and chain rule together (source: image by the authors).*

The derivative of our function
$F(x)$
is just the matrix multiplication of the Jacobian matrices of the intermediate terms.

Now, this is where we ask the question:

> *Does it matter the order in which we do the matrix multiplication?*

Forward and Reverse Accumulations

In this section, we try to understand the answer to the question of ordering the Jacobian matrix multiplication.

There are two extremes in which we could order the multiplications: the forward accumulation and the reverse accumulation.

Forward Accumulation

If we order the multiplication from right to left in the same order in which the function
$F(x)$
was evaluated, the process is called forward accumulation. The best way to think about the ordering is to place brackets in the equation, as shown in **Figure 3**.

$$F'(x) \;=\; \frac{\partial y}{\partial c}\left(\frac{\partial c}{\partial b}\left(\frac{\partial b}{\partial a}\frac{\partial a}{\partial x}\right)\right)$$

$$F'(x) = \frac{\partial y}{\partial c}\left(\frac{\partial c}{\partial b}\left(\underbrace{\frac{\partial b}{\partial a}\frac{\partial a}{\partial x}}_{\downarrow}\right)\right)$$

$$\frac{\partial b}{\partial x} = \begin{bmatrix} \frac{\partial b_1}{\partial x_1} & \cdots & \frac{\partial b_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial b_{\text{size of b}}}{\partial x_1} & \cdots & \frac{\partial b_{\text{size of b}}}{\partial x_n} \end{bmatrix}$$

Figure 3: *Forward accumulation of gradients (source: image by the authors).*

With the function

$F : \mathbb{R}^n \to \mathbb{R}$
, the forward accumulation process is matrix multiplication in all the steps. This is more FLOPs.

Note: Forward accumulation is beneficial when we want to get the derivative of a function

$F : \mathbb{R} \to \mathbb{R}^n$
.

Another way to understand forwarding accumulation is to think of a Jacobian-Vector Product (JVP). Consider a Jacobian
$F'(x)$
and a vector . The Jacobian-Vector Product would look to be
$F'(x)v$

$$F'(x)v \;=\; \frac{\partial y}{\partial c}\left(\frac{\partial c}{\partial b}\left(\frac{\partial b}{\partial a}\left(\frac{\partial a}{\partial x}v\right)\right)\right)$$

This is done for us to have matrix-vector multiplication at all the stages (which makes the process more efficient).

➤ **Question:** If we have a Jacobian-Vector Product, how can we obtain the Jacobian from it?

➤ **Answer:** We pass a one-hot vector and get each column of the Jacobian one at a time.

So we can think of forwarding accumulation as a process in which we build the Jacobian per column.

Reverse Accumulation

Suppose we order the multiplication from left to right, in the opposite direction to which the function was evaluated. In that case, the process is called reverse accumulation. The diagram of the process is illustrated in **Figure 4**.

$$F'(x) \;=\; \left(\left(\frac{\partial y}{\partial c}\frac{\partial c}{\partial b}\right)\frac{\partial b}{\partial a}\right)\frac{\partial a}{\partial x}$$

$$F'(x) = ((\underbrace{\frac{\partial y}{\partial c}\frac{\partial c}{\partial b}}_{\downarrow})\frac{\partial b}{\partial a})\frac{\partial a}{\partial x}$$

$$\frac{\partial y}{\partial b} = \begin{bmatrix} \frac{\partial y}{\partial x_1} & \cdots & \frac{\partial y}{\partial b_{\text{size of b}}} \end{bmatrix}$$

Figure 4: *Reverse accumulation of gradients (source: image by the authors).*

As it turns out, with reverse accumulation deriving the derivative of a function

$F : \mathbb{R}^n \rightarrow \mathbb{R}$
is a vector to matrix multiplication at all steps. This means that for the particular function, reverse accumulation has lesser FLOPs than forwarding accumulation.

Another way to understand forwarding accumulation is to think of a Vector-Jacobian Product (VJP). Consider a Jacobian
$F'(x)$
and a vector . The Vector-Jacobian Product would look to be

$v^T F'(x)$

$$v^T F'(x) \;=\; \left(\left(\left(v^T \frac{\partial y}{\partial c}\right) \frac{\partial c}{\partial b}\right) \frac{\partial b}{\partial a}\right) \frac{\partial a}{\partial x}$$

This allows us to have vector-matrix multiplication at all stages (which makes the process more efficient).

➤ **Question:** If we have a Vector-Jacobian Product, how can we obtain the Jacobian from it?

➤ **Answer:** We pass a one-hot vector and get each row of the Jacobian one at a time.

So we can think of reverse accumulation as a process in which we build the Jacobian per row.

Now, if we consider our previously mentioned function
$F(x)$
, we know that the Jacobian
$F'(x)$
is a row vector. Therefore, if we apply the reverse accumulation process, which means the Vector-Jacobian Product, we can obtain the row vector in one shot. On the other hand, if we apply the forward accumulation process, the Jacobian-Vector Product, we will obtain a single element as a column, and we would need to iterate to build the entire row.

This is why reverse accumulation is used more often in the Neural Network literature.

Summary

In this tutorial, we studied the math of automatic differentiation and how it is applied to the parameters of a Neural Network. The next tutorial will expand on this and see how we can implement automatic differentiation using a python package. The implementation will involve a step-by-step walkthrough of creating a python package and using it to train a neural network.

Did you enjoy a math-heavy tutorial on the fundamentals of automatic differentiation? Let us know.

Twitter: @PyImageSearch

References

Citation Information