Journal of Applied and Advanced Research 2024, 9: 15-25
Doi: 10.21839/jaar.2024.v9.9169

Research Article

A mathematical primer to classical deep learning

Enow Takang Achuo Albert*, Ngalle Hermine Bille, Ngonkeu Mangaptche Eddy Leonard

Department of Plant Biology, Faculty of Science, University of Yaoundé I, P.O. Box 812, Yaoundé, Center Region, Cameroon

Received: May 02, 2024; Revised: September 16, 2024; Accepted: September 17, 2024; Published: September 23, 2024
*Corresponding Author: Enow Takang Achuo Albert (E-mail: albert.enow@facsciences-uy1.cm)

ABSTRACT

This manuscript synthesizes the statistical foundations of classical deep learning by integrating insights from eight seminal works. It covers matrix calculus, neuron layers, weight and bias indexing, cost functions, differentiation of neuron operations, activation functions, bias functions, gradient descent, and backpropagation algorithms. The synthesis aims to provide a comprehensive understanding of the mathematical and statistical principles underpinning deep learning models, facilitating their application and further development in various domains.

Key words: Deep Learning, Matrix Calculus, Activation Functions, Gradient Descent, Backpropagation

INTRODUCTION

Preamble

The following works together form the basis for this communication – (Parr & Howard, 2018) on matrix calculus, (Cabello, 2022) on neuron layers and weight and bias indexing, (Pick & Cole, 2008) on cost functions, (Chen *et al.*, 2018) on differentiating neuron operations, (Kılıçarslan *et al.*, 2021) on activation functions, (James *et al.*, 2013) on bias functions, (Du *et al.*, 2017) on gradient descent functions and (Alber *et al.*, 2018; Parr & Howard, 2018) on back propagation algorithms.

Consider the following inexhaustive declaration of notations: m: size of the training set. n: number of input variables $(x_1, x_2, x_3, \cdots x_n)$. x_i: single input variable. L: number of layers in a neural network. l or ℓ : a specific layer in the network. w^l: weights (w) of layer l. ∇ : *nabla*. Symbol for the derivative of a function. A neural network (NN) can be thought of as a super function which harbors composite functions and has weights and biases as parameters. The input to the NN is a vector of all the variables in one training example. The major interest resides in calculating the cost and trying, as much as possible, to minimize it. With gradients, we display the partial derivatives of a function which transforms a vector into a scalar. Consider the function below

$$f(x,y) = x^2 + cos(y)$$

The partial derivatives with respect to both x and y (respectively) would be

$$\frac{\partial f}{\partial x} = 2x$$

$$\frac{\partial f}{\partial y} = -\sin(y)$$

The representation of the scalar output of *f(x,y)* would be

$$\begin{bmatrix} x \\ y \end{bmatrix} \xrightarrow[f(x,y)]{} (S)$$

A gradient describes how the partial derivatives of *f(x,y)* can be displayed

$$\begin{bmatrix} \partial f / \partial x \\ \partial f / \partial y \end{bmatrix} = \begin{bmatrix} 2x \\ -\sin(y) \end{bmatrix}$$

A *Jacobian* returns a vector when supplied with an input vector. The returned vector can be of same or different shape as the input vector. Consider that instead of returning a scalar, *f(x,y)* (redefined) returns a vector

$$f(x,y) = \begin{bmatrix} 2x + y^3 \\ e^y - 13x \end{bmatrix}$$

Computing the partial derivatives of the redefined *f(x,y)* becomes more complicated, since, in a manner of expressing, two 'functions' are now involved. They are

$$f_1 = 2x + y^3$$
$$f_2 = e^y - 13x$$

Now, the partial derivatives of each of the functions with respect to each of the variables can be defined

$$\frac{\partial f_1}{\partial x} = 2, \frac{\partial f_1}{\partial y} = 3y^2$$

and

$$\frac{\partial f_2}{\partial x} = -13, \frac{\partial f_2}{\partial y} = e^y$$

Similar to the gradient, the expression of the Jacobian would depend on the arrangement of the partial derivatives

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix}$$

Before examining the Jacobian chain rule, it would be helpful to begin by examining the scalar chain rule in a new way. Normally, if a function is expressed as

$$f(x) = \sin(x^2)$$

Then the derivative of *f (x)* (defined above) would be the product of the derivative of the inner function (x^2) and the derivative of the outer function (sin (x^2))

$$\nabla f(x) = 2x cos(x^2)$$

But there is an algorithmic way which can help plot any function. In this context, it requires splitting *f (x)* into two functions, *g* and *f*, defined as

$$g = x^2$$

and

$$f = \sin(g)$$

So, to methodically find the derivative of *f* with respect to *x*, the product of the derivative of *f* with respect to *g* and of *g* with respect to *x* is computed, as expressed below

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$$

So, to find the derivatives of very nested functions, it suffices to define intermediate functions, compute the product of the derivatives of those intermediate functions, then substitute the original information which was stored in each function definition. This is a new, algorithmic way of looking at the scalar chain rule. It can help to better examine vector-to-vector (Jacobian) functions. Consider the example function below

$$f(a,b) = \begin{bmatrix} \sin(a^2+b) \\ \ln(b^3) \end{bmatrix}$$

The expressions in $f(a,b)$ would, in and of themselves, generally require the chain rule for their derivatives. It is possible to set intermediate functions for the function expressions, then convert the setup to a vector of intermediate functions

$$g = \begin{bmatrix} a^2+b \\ b^3 \end{bmatrix}$$

and

$$f = \begin{bmatrix} \sin(g_1) \\ \ln(g_2) \end{bmatrix}$$

Now, both Jacobians can be expressed as

$$\frac{\partial \vec{g}}{\partial \vec{a}} = \begin{bmatrix} 2a & 1 \\ 0 & 3b^2 \end{bmatrix}$$

and

$$\frac{\partial \vec{f}}{\partial \vec{g}} = \begin{bmatrix} cos(g_1) & 0 \\ 0 & \frac{1}{g_2} \end{bmatrix}$$

Recall that in the scalar chain rule (as earlier expressed)

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}$$

Therefore

$$\frac{\partial \vec{f}}{\partial \vec{a}} = \frac{\partial \vec{f}}{\partial \vec{g}}\frac{\partial \vec{g}}{\partial \vec{a}}$$

or

$$\frac{\partial \vec{f}}{\partial \vec{a}} = \begin{bmatrix} \cos(g_1) & 0 \\ 0 & \frac{1}{g_2} \end{bmatrix}\begin{bmatrix} 2a & 1 \\ 0 & 3b^2 \end{bmatrix}$$

Which is

$$\frac{\partial \vec{f}}{\partial \vec{a}} = \begin{bmatrix} 2a\cos(g_1) & \cos(g_1) \\ 0 & \frac{3b^2}{g_2} \end{bmatrix}$$

Finally, substituting g_1 and g_2 with the stored expressions results in

$$\frac{\partial \vec{f}}{\partial \vec{a}} = \begin{bmatrix} 2a\cos(a^2+b) & \cos(a^2+b) \\ 0 & \frac{3b^2}{b^3} \end{bmatrix}$$

Which can be further simplified as

$$\frac{\partial \vec{f}}{\partial \vec{a}} = \begin{bmatrix} 2a\cos(a^2+b) & \cos(a^2+b) \\ 0 & \frac{3}{b} \end{bmatrix}$$

Forward propagation

The neuron function

Consider that x_1, x_2, x_3, x_4, x_5 are the inputs to a single neuron, *N*. These inputs are connected to the neuron by weights (w_1, w_2, w_3, w_4, w_5). The neuron would compute the weighted sum of the inputs, described as

$$\sigma\left(\sum_{i=1}^{n} x_i w_i + b\right)$$

Where *σ* is the sigmoid activation function, *n* is the total number of weights, and *b* is the bias. is outside the summation. A neural network with no nonlinear activation function can be proven to be equal to linear regression. The above summation can also be described more understandably, using dot products.

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} = x_1 w_1 + x_2 w_2 \cdots x_n w_n$$

With the above understanding, the weighted sum of the inputs can be simplified as

$$\sigma\left(x^T w + b\right)$$

If $x^T w + b$ is considered to be *z*, then the activation *a* for the neuron *N* would be

$$z = x^T w + b$$
$$a = \sigma(z)$$

Therefore, *a* is a vector to scalar function. The two-step representation method above would come handy with further differentiations.

Indexing weights and biases

The standard representation for weights in an NN is

$$w_{jk}^{l}$$

Where *l* is a specific layer in the network, *j* is the number of the neuron in which the weight enters, and *k* is the node in layer *l*–1. Biases are usually the same for each node in a layer. Generally, the bias for a given layer is expressed as b^l.

A layer in a neural network

A layer in a neural network is made up of nodes. Each node takes in a vector containing all of the inputs, multiplies them (inputs) by their respective weights, sum the results into a (computed) weighted sum, then adds the weighted sum to its own bias. at the end, each node churns out a scalar. All of the scalars then get assembled to form a vector, which becomes the input into the next layer (feed forward propagation). The weighted sum of inputs for an entire layer can therefore be expressed as

$$\sigma\left(W^l a^{l-1} + b^l\right)$$

Derivatives of neural networks and gradient descent

Motivation and cost functions

For obtaining the best outputs, it is important to reduce the RMSE (or any efficiency measure employed) as much as possible. This would entail changing the weights and biases of the NN often. To do this, one has to loop back through the network (back propagation) and note how each weight and bias affects the total cost (error). Using the information obtained, the weights and bias can be tweaked accordingly, reducing the cost and obtaining a better answer. This is the entire goal of back propagation. One of the most popular cost functions is the MSE. Here, each NN output is compared to the actual output. The difference is squared and the mean of all squared differences is computed. The MSE cost function (where *m* is the number of inputs, *y* is the ground truth for each input and $\hat{y}$ is the predicted ground truth, output by the NN) is expressed as

$$\frac{1}{2m}\sum_{i=1}^{m}\left(y_i - \hat{y}_i\right)^2$$

Using the Jacobian laws, the operations occurring in a single neuron would be further derived.

Differentiating a neuron's operations

Derivative of a binary element-wise operation

A binary element-wise operation is a function *f* (different from the total function) which takes in two vectors ($\vec{v}, \vec{w}$) and returns a single vector ($\vec{b}$), as shown below

$$f\left(\vec{v}, \vec{w}\right) = \vec{b}$$

The operations needed to be done on $\vec{v}$ and $\vec{w}$ need to be element-wise (as the name suggests). If the operation is a multiplication, it is referred to as a *Hadamard product*

$$F\left(\vec{v}, \vec{w}\right) = f\left(\vec{v}\right) \odot g\left(\vec{w}\right)$$

The total function *F* and the operator $\odot$ have to be element wise, but the functions *f* and *g* don't have to be. The Jacobian derivative of involves two functions, one with respect to the elements of $\vec{v}$, and the other with respect to the elements of $\vec{w}$

$$\frac{\partial F}{\partial \vec{v}} \text{ and } \frac{\partial F}{\partial \vec{w}}$$

Therefore

$$F\left(\vec{v}, \vec{w}\right) = \begin{bmatrix} f_1\left(\vec{v}\right) \odot g_1\left(\vec{w}\right) \\ f_2\left(\vec{v}\right) \odot g_2\left(\vec{w}\right) \\ \vdots \\ f_n\left(\vec{v}\right) \odot g_n\left(\vec{w}\right) \end{bmatrix}$$

And

$$F_1\left(\vec{v},\vec{w}\right)=f_1\left(\vec{v}\right)\odot g_1\left(\vec{w}\right)$$
$$F_2\left(\vec{v},\vec{w}\right)=f_2\left(\vec{v}\right)\odot g_2\left(\vec{w}\right)$$
$$\vdots$$
$$F_n\left(\vec{v},\vec{w}\right)=f_n\left(\vec{v}\right)\odot g_n\left(\vec{w}\right)$$

It is noteworthy that since *f* and *g* don't have to be element-wise, $\vec{v}$ and $\vec{w}$ don't need to be indexed.

$$\frac{\partial F}{\partial \vec{v}}=$$
$$\begin{bmatrix} \frac{\partial}{\partial \vec{v}_1} f_1\left(\vec{v}\right)\odot g_1\left(\vec{w}\right) & \frac{\partial}{\partial \vec{v}_2} f_1\left(\vec{v}\right)\odot g_1\left(\vec{w}\right) & \cdots \\ \frac{\partial}{\partial \vec{v}_n} f_1\left(\vec{v}\right)\odot g_1\left(\vec{w}\right) & & \\ \frac{\partial}{\partial \vec{v}_1} f_2\left(\vec{v}\right)\odot g_2\left(\vec{w}\right) & \frac{\partial}{\partial \vec{v}_2} f_2\left(\vec{v}\right)\odot g_2\left(\vec{w}\right) & \cdots \\ \frac{\partial}{\partial \vec{v}_n} f_2\left(\vec{v}\right)\odot g_2\left(\vec{w}\right) & & \\ & \vdots & \\ \frac{\partial}{\partial \vec{v}_1} f_n\left(\vec{v}\right)\odot g_n\left(\vec{w}\right) & \frac{\partial}{\partial \vec{v}_2} f_n\left(\vec{v}\right)\odot g_n\left(\vec{w}\right) & \cdots \end{bmatrix}$$
$$\frac{\partial}{\partial \vec{v}_n} f_n\left(\vec{v}\right)\odot g_n\left(\vec{w}\right)$$

Looking at $\frac{\partial F}{\partial \vec{v}}$ above, and noting that all non-diagonal derivatives would result in nil, since they have no relationship with the indexed *F*s (F_1,F_2 , F_n), $\frac{\partial F}{\partial \vec{v}}$ can be re-expressed as

$$\frac{\partial F}{\partial \vec{v}}=\begin{bmatrix} \frac{\partial}{\partial \vec{v}_1} f_1\left(\vec{v}\right)\odot g_1\left(\vec{w}\right) & 0 & \cdots & 0 \\ 0 & \frac{\partial}{\partial \vec{v}_2} f_2\left(\vec{v}\right)\odot g_2\left(\vec{w}\right) & \cdots & 0 \\ & \vdots & & \\ 0 & 0 & \cdots & \frac{\partial}{\partial \vec{v}_n} f_n\left(\vec{v}\right)\odot g_n\left(\vec{w}\right) \end{bmatrix}$$

Extrapolating, it is conclusive that for all element-wise functions, the Jacobian would be diagonal. When

$$diag\left(a,b,c\right)=\begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix},$$

Then in the context of the afore-expressed Jacobian,

$$diag\left(a,b,c\right)=diag\begin{pmatrix} \frac{\partial}{\partial \vec{v}_1} f_1\left(\vec{v}\right)\odot g_1\left(\vec{w}\right), \frac{\partial}{\partial \vec{v}_2} f_2\left(\vec{v}\right)\odot \\ g_2\left(\vec{w}\right), \frac{\partial}{\partial \vec{v}_n} f_n\left(\vec{v}\right)\odot g_n\left(\vec{w}\right) \end{pmatrix}$$

Derivative of a Hadamard product

A Hadamard product is an element-wise multiplication of two vectors.

$$F\left(\vec{v},\vec{w}\right)=\begin{bmatrix} \vec{v}_1 \otimes \vec{w}_1 \\ \vec{v}_2 \otimes \vec{w}_2 \\ \vdots \\ \vec{v}_n \otimes \vec{w}_n \end{bmatrix}$$

$\frac{\partial F}{\partial \vec{v}}$ would now result in

$$\frac{\partial F}{\partial \vec{v}}=\begin{bmatrix} \frac{\partial F_1}{\partial \vec{v}_1} & \frac{\partial F_1}{\partial \vec{v}_2} & \cdots & \frac{\partial F_1}{\partial \vec{v}_n} \\ \frac{\partial F_2}{\partial \vec{v}_1} & \frac{\partial F_2}{\partial \vec{v}_2} & \cdots & \frac{\partial F_2}{\partial \vec{v}_n} \\ & \vdots & & \\ \frac{\partial F_n}{\partial \vec{v}_1} & \frac{\partial F_n}{\partial \vec{v}_2} & \cdots & \frac{\partial F_n}{\partial \vec{v}_n} \end{bmatrix}$$

Since all non-diagonal derivatives would result in nil, $\frac{\partial F}{\partial \vec{v}}$ will now be re-presented as

$$\frac{\partial F}{\partial \vec{v}}=\begin{bmatrix} \frac{\partial F_1}{\partial \vec{v}_1} & 0 & \cdots & 0 \\ 0 & \frac{\partial F_2}{\partial \vec{v}_2} & \cdots & 0 \\ & \vdots & & \\ 0 & 0 & \cdots & \frac{\partial F_n}{\partial \vec{v}_n} \end{bmatrix}$$

Further, defining

$$W_1 = \frac{\partial F_1}{\partial \vec{v}_1} = v_1,$$
$$W_2 = \frac{\partial F_2}{\partial \vec{v}_2} = v_2,$$
$$W_n = \frac{\partial F_n}{\partial \vec{v}_n} = v_n,$$

$\frac{\partial F}{\partial \vec{v}}$ finally becomes

$$\frac{\partial F}{\partial \vec{v}} = \begin{bmatrix} V_1 & 0 & \dots & 0 \\ 0 & V_2 & \dots & 0 \\ & \vdots & & \\ 0 & 0 & \dots & V_n \end{bmatrix}$$

Derivative of a scalar expansion

This is the derivative of multiplying a vector by a scalar. The expression below demonstrates the multiplication of a vector by a scalar.

$$2\begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} 2v_1 \\ 2v_2 \\ \vdots \\ 2v_n \end{bmatrix}$$

A more expressive way or re-presenting the above expression is by broadcasting the scalar (transforming it into a vector of same dimensions as the other vector), then performing an element-wise multiplication

$$\begin{bmatrix} 2 \\ 2 \\ \vdots \\ 2 \end{bmatrix} \otimes \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} 2v_1 \\ 2v_2 \\ \vdots \\ 2v_n \end{bmatrix}$$

More generally, the interest is to determine the derivative of the *z* with respect to the vector elements. Let

$$F\left(\vec{v}, x\right) = F\left(\vec{v}\right) \odot g(x)$$

where

$$g(x) = \vec{1}x$$

It is worth noting that the multiplication of *x* by the ones vector ($\vec{1}$) is an act of broadcasting *x* itself. $F\left(\vec{v}, x\right)$ becomes re-expressed as

$$F\left(\vec{v}, x\right) = \begin{bmatrix} f_1\left(\vec{v}\right) \odot g_1(x) \\ f_2\left(\vec{v}\right) \odot g_2(x) \\ \vdots \\ f_n\left(\vec{v}\right) \odot g_n(x) \end{bmatrix}$$

The Jacobian with respect to the elements of is expressed as $\vec{v}$

$$\frac{\partial F}{\partial \vec{v}} = \begin{bmatrix} \frac{\partial f_1}{\partial v_1} & \frac{\partial f_1}{\partial v_2} & \dots & \frac{\partial f_1}{\partial v_n} \\ \frac{\partial f_2}{\partial v_1} & \frac{\partial f_2}{\partial v_2} & \dots & \frac{\partial f_2}{\partial v_n} \\ & \vdots & & \\ \frac{\partial f_n}{\partial v_1} & \frac{\partial f_n}{\partial v_2} & \dots & \frac{\partial f_n}{\partial v_n} \end{bmatrix}$$

Since all non-diagonal derivatives would result in nil, $\frac{\partial F}{\partial \vec{v}}$ will once again be re-presented as

$$\frac{\partial F}{\partial \vec{v}} = \begin{bmatrix} \frac{\partial f_1}{\partial v_1} & 0 & \dots & 0 \\ 0 & \frac{\partial f_2}{\partial v_2} & \dots & 0 \\ & \vdots & & \\ 0 & 0 & \dots & \frac{\partial f_n}{\partial v_n} \end{bmatrix}$$

x is unique in the sense that it is a scalar.

Since it is just a single number, it has no indexes. Interestingly, the derivative with respect to *x* is a gradient, and not a Jacobian

$$\triangleleft F_x = \begin{bmatrix} \frac{\partial f_1}{\partial x} \\ \frac{\partial f_2}{\partial x} \\ \vdots \\ \frac{\partial f_n}{\partial x} \end{bmatrix}$$

By applying rules of scalar calculus

$$\triangleleft F_x = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

Derivative of a neuron's activation

The derivative of a neuron with respect to the weights and the bias is examined by understanding how the activation changes with respect to changes in the weights and bias (ignoring the cost function at this point). The activation is

$$a = \sigma\left(W^T x + b\right)$$

If

$$z = W^T x + b$$

then

$$\frac{\partial a}{\partial W} = \frac{\partial a}{\partial z}\frac{\partial z}{\partial W}$$

and

$$\frac{\partial a}{\partial b} = \frac{\partial a}{\partial z}\frac{\partial z}{\partial b}$$

Representing the activation as a Hadamard product changes the expression to:

$$a = \sigma\left(sum\left(W \otimes x\right) + b\right)$$

Letting

$$H = W \otimes x$$

and

$$S\left(H\right) = sum\left(W \otimes x\right),$$

then $\frac{\partial a}{\partial W}$ becomes

$$\frac{\partial a}{\partial W} = \frac{\partial a}{\partial z}\frac{\partial z}{\partial S}\frac{\partial S}{\partial H}\frac{\partial H}{\partial W}$$

and $\frac{\partial a}{\partial b}$ remains

$$\frac{\partial a}{\partial b} = \frac{\partial a}{\partial z}\frac{\partial z}{\partial b}$$

Now, $\frac{\partial S}{\partial H}$ is

$$\frac{\partial S}{\partial H} = 1^T$$

and $\frac{\partial H}{\partial W}$ is

$$\frac{\partial H}{\partial W} = diag\left(x_1 \quad x_2 \quad \cdots \quad x_n\right)$$

Also, $\frac{\partial z}{\partial S}$ is

$$\frac{\partial z}{\partial S} = 1$$

$\frac{\partial S}{\partial H}\frac{\partial H}{\partial W}$ is

$$\frac{\partial S}{\partial H}\frac{\partial H}{\partial W} = \left[x\right]^T$$

therefore

$$\frac{\partial a}{\partial W} = \frac{\partial a}{\partial z}\left[x\right]^T$$

At this point, it is important to note that as opposed to sigmoid, which is a concrete activation function,

$$sigmoid = \frac{1}{1 + e^{-(wx+b)}}$$

ReLU on the other hand is

$$\max\left(0, z\right), z = sum\left(W \otimes x\right) + b$$

The slope of ReLU at $z > 0$ is exactly 1. ReLU is not continues the whole way through, since it is indifferentiable at $z = 0$. Therefore, the graph of ReLU is a discontinuous piecewise function. The differential of ReLU is

$$\begin{cases} 0 & if \quad z \le 0 \\ z & if \quad z > 0 \end{cases}$$

After further substitutions,

$$\frac{\partial a}{\partial W} = \begin{cases} \vec{0}^T & if \quad W^T x + b \le 0 \\ \left[x\right]^T & if \quad W^T x + b > 0 \end{cases}$$

Derivative of the cost for a simple neural network

The cost was not considered while finding the derivative of the neuron's activation function. Here, its derivative shall be presented. Recall in I.2.5.3 that the cost function (MSE)was expressed as follows

$$\frac{1}{2m}\sum_{i=1}^{m}\left(y_i - \hat{y}_i\right)^2$$

It was previously expressed (with a no-change-adding modification) that

$$\frac{\partial a}{\partial W} = \begin{cases} \vec{0}^T & if \quad W^T x + b \le 0 \\ \left[x\right]^T & if \quad W^T x + b > 0 \end{cases} \text{ and}$$

$$\frac{\partial a}{\partial b} = \begin{cases} 0 & if \quad W^T x + b \le 0 \\ 1 & if \quad W^T x + b > 0 \end{cases} = \frac{\partial a}{\partial z}\frac{\partial z}{\partial b}$$

For the cost, C, the goal is to express both $\frac{\partial C}{\partial W}$ and $\frac{\partial C}{\partial b}$.

$\frac{\partial C}{\partial W}$ is expressed as

$$\frac{\partial C}{\partial W}=\frac{\partial C}{\partial a}\frac{\partial a}{\partial W} \text{ (and } \frac{\partial a}{\partial W}=\begin{cases}\vec{0}^{T} & if \quad W^{T}x+b\le 0\\ [x]^{T} & if \quad W^{T}x+b>0\end{cases})$$

$\frac{\partial C}{\partial b}$ is expressed as

$$\frac{\partial C}{\partial b}=\frac{\partial C}{\partial a}\frac{\partial a}{\partial b} \text{ (and } \frac{\partial a}{\partial b}=\begin{cases}0 & if \quad W^{T}x+b\le 0\\ 1 & if \quad W^{T}x+b>0\end{cases}=\frac{\partial a}{\partial z}\frac{\partial z}{\partial b})$$

To express $\frac{\partial C}{\partial a}$, $\frac{1}{2m}\sum_{i=1}^{m}\left(y_i-\hat{y}_i\right)^2$ has to be differentiated, where $\hat{y}=activation$ (so to speak).

Consider X to be the expression for the training examples. If $\hat{y}=activation=a^{L}$, then the cost can be re-expressed as

$$\frac{1}{2m}\sum_{i=1}^{m}\left(y-a^{L}\right)^2$$

Attributing $y-a^{L}$ to v, the cost can further be re-expressed as

$$\frac{1}{2m}\sum_{i=1}^{m}(v)^2$$

So,

$$\frac{\partial C}{\partial W}=\frac{\partial C}{\partial a}\frac{\partial a}{\partial W}=\frac{\partial C}{\partial v}\frac{\partial v}{\partial a}\frac{\partial a}{\partial W}$$

(and $\frac{\partial a}{\partial W}=\begin{cases}\vec{0}^{T} & if \quad W^{T}x+b\le 0\\ [x]^{T} & if \quad W^{T}x+b>0\end{cases}$)

or (intuitively),

$$\frac{\partial C}{\partial W}=\frac{\partial C}{\partial a}\frac{\partial a}{\partial W}=\frac{\partial C}{\partial v}\frac{\partial v}{\partial a}\frac{\partial a}{\partial W}=\frac{\partial C}{\partial v}\frac{\partial v}{\partial W}$$

Continuing,

$$\frac{\partial v}{\partial W}=\frac{\partial\left(y-a^{L}\right)}{\partial W}=\frac{\partial -aL}{\partial W}=-\frac{\partial aL}{\partial W}=\frac{\partial v}{\partial a}$$

And so,

$$\frac{\partial v}{\partial W}=-\frac{\partial a}{\partial W}$$

Knowing that the derivative of a sum is same as the sum of the derivatives,

$$\underbrace{\frac{\partial}{\partial W}\frac{1}{2m}\sum_{i=1}^{m}(v)^2}_{derivative-of-the-sum}=\underbrace{\frac{1}{2m}\sum_{i=1}^{m}\frac{\partial}{\partial W}(v)^2}_{sum-of-the-derivatives}$$

Continuing,

$$\frac{1}{2m}\sum_{i=1}^{m}\frac{\partial}{\partial W}(v)^2=\frac{1}{2m}\sum_{i=1}^{m}2v\frac{\partial v}{\partial W}$$

And when the twos cancel each other

$$\frac{1}{2m}\sum_{i=1}^{m}2v\frac{\partial v}{\partial W}=\frac{1}{m}\sum_{i=1}^{m}v\frac{\partial v}{\partial W}$$

Recalling that

$$\frac{\partial v}{\partial W}=-\frac{\partial a}{\partial W} \text{ (and } \frac{\partial a}{\partial W}=\begin{cases}\vec{0}^{T} & if \quad W^{T}x+b\le 0\\ [x]^{T} & if \quad W^{T}x+b>0\end{cases})$$

$\frac{1}{m}\sum_{i=1}^{m}v\frac{\partial v}{\partial W}$ becomes

$$\frac{1}{m}\sum_{i=1}^{m}v\begin{cases}\vec{0}^{T} & if \quad W^{T}x+b\le 0\\ -[x]^{T} & if \quad W^{T}x+b>0\end{cases}$$

which can also be represented as

$$\frac{1}{m}\sum_{i=1}^{m}\begin{cases}\vec{0}^{T} & if \quad W^{T}x+b\le 0\\ -v[x]^{T} & if \quad W^{T}x+b>0\end{cases}$$

Also recalling that $\hat{y}=activation=a^{L}$ and $v=y-a^{L}$, $\frac{1}{m}\sum_{i=1}^{m}\begin{cases}\vec{0}^{T} & if \quad W^{T}x+b\le 0\\ -v[x]^{T} & if \quad W^{T}x+b>0\end{cases}$ becomes

$$\frac{1}{m}\sum_{i=1}^{m}\begin{cases}\vec{0}^{T} & if \quad W^{T}x+b\le 0\\ -\left(y-\hat{y}\right)[x]^{T} & if \quad W^{T}x+b>0\end{cases}$$

Recalling that

$$z=W^{T}x+b$$

and ReLU is defined as

$$\max(0,z)$$

and

$$\hat{y}=\max(0,z),$$

$\frac{1}{m}\sum_{i=1}^{m}\begin{cases}\vec{0}^{T} & if \quad W^{T}x+b\le 0\\ -\left(y-\hat{y}\right)[x]^{T} & if \quad W^{T}x+b>0\end{cases}$ can be re-expressed as:

$$\frac{1}{m}\sum_{i=1}^{m}\begin{cases}\vec{0}^{T} & if \quad W^{T}x+b\le 0\\ -\left(y-\max\left(0,W^{T}x+b\right)\right)[x]^{T} & if \quad W^{T}x+b>0\end{cases}$$

It is observable from the expression above that the *max* function is redundant, since the first part of the piece-wise expression would handle a case where $W^T x + b = 0$. So, the expression would make more sense as

$$\frac{1}{m}\sum_{i=1}^{m}\begin{cases} \vec{0}^{T} & if \quad W^T x + b \le 0 \\ -\left(y - \left(W^T x + b\right)\right)[x]^T & if \quad W^T x + b > 0 \end{cases}$$

which is rearrangeable as

$$\frac{1}{m}\sum_{i=1}^{m}\begin{cases} \vec{0}^{T} & if \quad W^T x + b \le 0 \\ \left(W^T x + b - y\right)[x]^T & if \quad W^T x + b > 0 \end{cases}$$

Finally, the summation and the fraction can be brought into the piece-wise expression to obtain

$$\begin{cases} \vec{0}^{T} & if \quad W^T x + b \le 0 \\ \frac{1}{m}\sum_{i=1}^{m} \left(W^T x + b - y\right)[x]^T & if \quad W^T x + b > 0 \end{cases}$$

and so

$$\frac{\partial C}{\partial W} = \begin{cases} \vec{0}^{T} & if \quad W^T x + b \le 0 \\ \frac{1}{m}\sum_{i=1}^{m} \left(W^T x + b - y\right)[x]^T & if \quad W^T x + b > 0 \end{cases}$$

Understanding the derivatives of the cost with respect to the weights

From the expression above, what is needed is a Jacobian (or derivative, or gradient) which can be fed into gradient descent to find the minima of the cost (or loss). Consider the following equation for the error term defined below (obtained from the second part of the piece-wise expression above)

$$e_i = W^T x + b - y$$

The first part of the second part of the piece-wise expression can then be rewritten as

$$\frac{1}{m}\sum_{i=1}^{m} e_i [x]^T$$

In the case of a single example, the above expression is rewritten as

$$e\,[x]^T$$

Expanding the expression above results in

$$e[x]^T = \begin{bmatrix} ex_1 \\ ex_2 \\ \vdots \\ ex_n \end{bmatrix}$$

Given that

$$\frac{\partial C}{\partial \vec{W}} = e[x]^T,$$

$$\frac{\partial C}{\partial \vec{W}} = \begin{bmatrix} ex_1 \\ ex_2 \\ \vdots \\ ex_n \end{bmatrix}$$

So,

$$\frac{\partial C}{\partial \vec{W}} = \begin{bmatrix} ex_1 & \frac{\partial C}{\partial \vec{W}_1} \\ ex_2 & \frac{\partial C}{\partial \vec{W}_2} \\ \vdots & \vdots \\ ex_n & \frac{\partial C}{\partial \vec{W}_n} \end{bmatrix}$$

It has now been demonstrated that the whole cost function is one big vector to scalar problem.

Finally, in the case of multiple inputs

$$\frac{1}{m}\sum_{i=1}^{m} e_i [x]^T = \begin{bmatrix} e_1x_1 \\ e_1x_2 \\ \vdots \\ e_1x_n \end{bmatrix} + \begin{bmatrix} e_2x_1 \\ e_2x_2 \\ \vdots \\ e_2x_n \end{bmatrix} + \dots \begin{bmatrix} e_mx_1 \\ e_mx_2 \\ \vdots \\ e_mx_n \end{bmatrix}$$

Also representing it as one big vector

$$\frac{1}{m}\sum_{i=1}^{m} e_i [x]^T = \begin{bmatrix} e_1x_1 + & e_2x_1 + & \cdots & + e_mx_1 \\ e_1x_2 + & e_2x_2 + & \cdots & + e_mx_2 \\ & & \vdots & \\ e_1x_n + & e_2x_n + & \cdots & + e_mx_n \end{bmatrix}$$

Differentiating the bias

At this point, the task is to find the derivative of the cost with respect to the bias. Recalling:

$$C = \frac{1}{2m}\sum_{i=1}^{m}\left(y - a^L\right)^2$$

and setting

$$v = y - a^L,$$

Then using the chain rule, $\frac{\partial C}{\partial b}$ is expressed as

$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial v}\frac{\partial v}{\partial a^L}\frac{\partial a^L}{\partial b}$$

Already, $\frac{\partial a^L}{\partial b}$ is known

$$\begin{cases} 0 & if \quad W^T x + b \le 0 \\ 1 & if \quad W^T x + b > 0 \end{cases} = \frac{\partial a}{\partial z}\frac{\partial z}{\partial b}$$

Also,

$$\frac{\partial v}{\partial a^L} = -1$$

Expressing C in terms of v,

$$C = \frac{1}{2m}\sum_{i=1}^{m}(v)^2$$

and

$$\frac{\partial v}{\partial a^L}\frac{\partial a^L}{\partial b} = -1 \bullet \begin{cases} 0 & if \quad W^T x + b \le 0 \\ 1 & if \quad W^T x + b > 0 \end{cases} = \frac{\partial a}{\partial z}\frac{\partial z}{\partial b}$$

which becomes

$$\frac{\partial v}{\partial a^L}\frac{\partial a^L}{\partial b} = \begin{cases} 0 & if \quad W^T x + b \le 0 \\ -1 & if \quad W^T x + b > 0 \end{cases} = \frac{\partial a}{\partial z}\frac{\partial z}{\partial b}$$

What remains is the derivative of the cost function with respect to v. Knowing that the derivative of the sum is equal to the sum of the derivatives,

$$\frac{\partial C}{\partial v} = \frac{\partial}{\partial v}\frac{1}{2m}\sum_{i=1}^{m}(v)^2 = \frac{1}{2m}\sum_{i=1}^{m}\frac{\partial}{\partial v}(v)^2$$

Then using the chain rule,

$$\frac{\partial}{\partial v}(v)^2 = \frac{\partial v^2}{\partial v}\frac{\partial v}{\partial b}$$

and so

$$\frac{\partial C}{\partial v} = \frac{\partial}{\partial v}\frac{1}{2m}\sum_{i=1}^{m}(v)^2 = \frac{1}{2m}\sum_{i=1}^{m}\frac{\partial}{\partial v}(v)^2 = \frac{1}{2m}\sum_{i=1}^{m}\left(\frac{\partial v^2}{\partial v}\frac{\partial v}{\partial b}\right)$$

Applying the power rule on $\frac{\partial v^2}{\partial v}$,

$$\frac{1}{2m}\sum_{i=1}^{m}\left(\frac{\partial v^2}{\partial v}\frac{\partial v}{\partial b}\right) = \frac{1}{2m}\sum_{i=1}^{m}2v\frac{\partial v}{\partial b}$$

After canceling out the twos,

$$\frac{\partial C}{\partial v} = \frac{1}{m}\sum_{i=1}^{m}v\frac{\partial v}{\partial b}$$

Noting that

$$\frac{\partial v}{\partial a^L}\frac{\partial a^L}{\partial b} = \begin{cases} 0 & if \quad W^T x + b \le 0 \\ -1 & if \quad W^T x + b > 0 \end{cases}$$

simplifies to

$$\frac{\partial v}{\partial b},$$

then

$$\frac{\partial C}{\partial v} = \frac{1}{m}\sum_{i=1}^{m}v \bullet \begin{cases} 0 & if \quad W^T x + b \le 0 \\ -1 & if \quad W^T x + b > 0 \end{cases}$$

which is rearranged as

$$\frac{\partial C}{\partial v} = \frac{1}{m}\sum_{i=1}^{m}\begin{cases} 0 & if \quad W^T x + b \le 0 \\ -v & if \quad W^T x + b > 0 \end{cases}$$

Further, recalling that

$$v = y - a^L$$

and substituting it in $\frac{\partial C}{\partial v}$,

$$\frac{\partial C}{\partial v} = \frac{1}{m}\sum_{i=1}^{m}\begin{cases} 0 & if \quad W^T x + b \le 0 \\ -\left(y - a^L\right) & if \quad W^T x + b > 0 \end{cases}$$

Which can then be rearranged as

$$\frac{\partial C}{\partial v} = \frac{1}{m}\sum_{i=1}^{m}\begin{cases} 0 & if \quad W^T x + b \le 0 \\ a^L - y & if \quad W^T x + b > 0 \end{cases}$$

Also recalling that

$$a^L = W^T x + b$$

and substituting it in $\frac{\partial C}{\partial v}$,

$$\frac{\partial C}{\partial v} = \frac{1}{m}\sum_{i=1}^{m}\begin{cases} 0 & if \quad W^T x + b \le 0 \\ \left(W^T x + b - y\right) & if \quad W^T x + b > 0 \end{cases}$$

Rearranging $\frac{\partial C}{\partial v}$,

$$\frac{\partial C}{\partial v} = \begin{cases} 0 & if \quad W^T x + b \le 0 \\ \frac{1}{m}\sum_{i=1}^{m}\left(W^T x + b - y\right) & if \quad W^T x + b > 0 \end{cases}$$

Finally, recalling that

$$\frac{\partial C}{\partial b} = \frac{\partial C}{\partial v}\frac{\partial v}{\partial a^L}\frac{\partial a^L}{\partial b} = \frac{\partial C}{\partial v}\frac{\partial v}{\partial b}$$

Then $\frac{\partial C}{\partial b}$ is

$$\frac{\partial C}{\partial b} = \begin{cases} 0 & if \quad W^T x + b \le 0 \\ \frac{1}{m}\sum_{i=1}^{m}\left(W^T x + b - y\right) & if \quad W^T x + b > 0 \end{cases}$$

$$\bullet \begin{cases} 0 & if \quad W^T x + b \le 0 \\ -1 & if \quad W^T x + b > 0 \end{cases}$$

Gradient descent algorithm

The derivative of the cost function with respect to both the weights and the biases is

$$\nabla_{w,b} C = \begin{bmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial b_1} \\ \vdots \\ \frac{\partial C}{\partial w_n} \end{bmatrix}$$

The gradient points in the direction of steepest descent of the cost function. Let θ represent all weights and biases in a NN. Gradient descent is an iterative algorithm, which seeks to minimize the cost with every iteration. Initially, random weights are set. This causes the cost to be very high. θ would then be updated with

$$\theta - \alpha \nabla_{w,b} C$$

where α is a scalar, called the *learning rate*. Notice that adding $-\alpha \nabla_{w,b} C$ means the weights and biases would be decreased, therefore reducing the cost in turn. The element-wise operation is represented as

$$\begin{bmatrix} W_1 - \alpha \frac{\partial C}{\partial w_1} \\ W_2 - \alpha \frac{\partial C}{\partial w_2} \\ \vdots \\ b_n - \alpha \frac{\partial C}{\partial w_n} \end{bmatrix}$$

The coefficient (or learning rate, α) plays a critical role by determining how quickly step-downs in the gradient occur. A large learning rate may cause the optimization process to miss the point of best cost and stop at a point of worse cost. A very small learning rate however, may lead to the point of best cost, but it may be very expensive, computationally (may take a lot of time).

Finding the derivatives of an entire layer

Consider a three-layered neural network with three inputs x_1, x_2, x_3, three weight matrices W_1, W_2, W_3, three activation layers a^1, a^2, a^3 and three biases b_1, b_2, b_3. The derivative of the cost with respect to the weight matrices is expressed as

$$\frac{\partial C}{\partial w_1} = \frac{\partial a^1}{\partial w_1} \frac{\partial a^2}{\partial a^1} \frac{\partial a^3}{\partial a^2} \frac{\partial C}{\partial a^3}$$

$$\frac{\partial C}{\partial w_2} = \frac{\partial a^2}{\partial w_2} \frac{\partial a^3}{\partial a^2} \frac{\partial C}{\partial a^3}$$

$$\frac{\partial C}{\partial w_3} = \frac{\partial a^3}{\partial w_3} \frac{\partial C}{\partial a^3}$$

Back propagation

Error of a node

Consider a multi-layer NN. Also, consider the activation for the second node in the layer as

$$z_2^1 = \left(W^T x + b\right)$$

Suppose an infinitesimal addition is made to the node, expressed as

$$z_2^1 + \Delta = \left(W^T x + b + \Delta\right)$$

The error of the j^{th} node of layer ℓ can be defined as

$$\delta_j^\ell = \frac{\partial C}{\partial z_j^\ell}$$

The four equations of back propagation

The error of all nodes in the last layer

The error of all nodes in the last layer is expressed as

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L}$$

which is also

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'\left(z_j^L\right)$$

The above equation is component-wise for the error of the last node. In back propagation, it is more apt to use the matrix-based version for the entire layer.

$$\delta^L = \nabla_a C \odot \sigma'\left(z^L\right)$$

The error of any node

As per the equation below, if the error of the nodes in layer $\ell + 1$ are known, then they can be used to find the error of the nodes in layer ℓ.

$$\delta^\ell = \left(\left(W^{\ell+1}\right)^T \delta^{\ell+1}\right) \odot \sigma'\left(z^\ell\right)$$

The derivative of the cost with respect to any bias

For any layer, the derivative of the cost with respect to the bias of that layer is expressed as

$$\frac{\partial C}{\partial b_\ell} = \delta^\ell$$

Recall that

$$\delta^{\ell} = \left(\left(W^{\ell+1} \right)^{T} \delta^{\ell+1} \right) \odot \sigma' \left(z^{\ell} \right)$$

Therefore, $\frac{\partial C}{\partial b_{\ell}}$ can also be expressed as

$$\left(\left(W^{\ell+1} \right)^{T} \delta^{\ell+1} \right) \odot \sigma' \left(z^{\ell} \right)$$

The derivative of the cost with respect to any weight

The derivative of the cost with respect to any weight is expressed as

$$\frac{\partial C}{\partial W_{jk}^{\ell}} = a_k^{\ell-1} \bullet \delta_j^{\ell}$$

In the vectorized form, the equation can be generalized to the level of an entire layer, and re-written as

$$\frac{\partial C}{\partial W^{\ell}} = \delta^{\ell} \left(a^{\ell-1} \right)^{T}$$

CONCLUSION

This synthesis of the statistical foundations of classical deep learning highlights the critical mathematical and statistical principles that drive the effectiveness of deep learning models. By understanding these foundational elements, researchers and practitioners can enhance the development and application of deep learning techniques, leading to more robust and efficient models.

REFERENCES

Alber, M., Bello, I., Zoph, B., Kindermans, P.-J., Ramachandran, P., & Le, Q. (2018). *Backprop Evolution. arXiv,* arXiv:1808.02822. https://doi.org/10.48550/arXiv.1808.02822

Cabello, J. G. (2022). Mathematical Neural Networks. *Axioms, 11*(2), 80. https://doi.org/10.3390/axioms11020080

Chen, R. T. Q., Rubanova, Y., Bettencourt, J., & Duvenaud, D. K. (2018). Neural Ordinary Differential Equations. *arXiv,* arXiv:1806.07366. https://doi.org/10.48550/arXiv.1806.07366

Du, S. S., Jin, C., Lee, J. D., Jordan, M. I., Poczos, B., & Singh, A. (2017). *Gradient Descent Can Take Exponential Time to Escape Saddle Points. arXiv,* arXiv:1705.10412. https://doi.org/10.48550/arXiv.1705.10412

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning: With Applications in R.* New York, US: Springer Science & Business Media. https://doi.org/10.1007/978-1-0716-1418-1

Kılıçarslan, S., Adem, K., & Çelik, M. (2021). An overview of the activation functions used in deep learning algorithms. *Journal of New Results in Science, 10*(3), 3. https://doi.org/10.54187/jnrs.1011739

Parr, T., & Howard, J. (2018). *The Matrix Calculus You Need For Deep Learning. arXiv,* arXiv:1802.01528. https://doi.org/10.48550/arXiv.1802.01528

Pick, A. J., & Cole, D. J. (2008). A Mathematical Model of Driver Steering Control Including Neuromuscular Dynamics. *Journal of Dynamic Systems, Measurement, and Control, 130*(3), 031004. https://doi.org/10.1115/1.2837452