NNs

Prerequisites

Basic Linear algebra
Multivariable Calculus
→ Differential
→ Jacobian's / gradients
→ Basic ML knowledge

Agenda

Multivariable Calculus Review
Neurons as a function
Jacobians & NNs
Gradient descent
Back propogation
BP as Matrix Calculus

Basic Notation

Define m = size of training set

Define n = # of input variables $x_1, x_2, \ldots$ $\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$

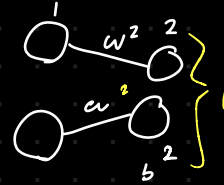
Define ^nL = amount of layers in NN

Define l = specific layer

w^l = weight (l) l = layer weights going into

b^l - bias for layer l

eg.



Define x_i = single input variable

Big picture

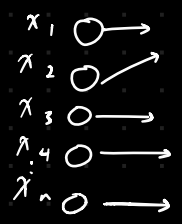
NNs are functions (linear functions of weights & biases)

Big calculus problem (linear approximations)

→ $\frac{\partial cost}{\partial w}$ (want this)

Find how much each weight & bias contributes to the cost

with respect to every weight and bias



$$f(x,y) = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix} = \begin{bmatrix} 2x + y^3 \\ e^y - 13x \end{bmatrix}$$

→ Breaks up into 2 scalar functions

$\mathbb{R}^2 \to \mathbb{R}^2$

① $f_1 = 2x + y^3 \to \frac{\partial f_1}{\partial x} = 2 \quad \frac{\partial f_1}{\partial y} = 3y^2$

② $f_2 = e^y - 13x \to \frac{\partial f_2}{\partial x} = -13 \quad \frac{\partial f_2}{\partial y} = e^y$

small functions composing one large function

create Jacobian

denoted also as: $\nabla f_1, \nabla f_2$

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_2}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial x} \end{bmatrix} = \begin{pmatrix} 2 & 3y^2 \\ -13 & e^y \end{pmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

"Transposed" defined rigorously from original function

Matrix Calculation Review

Gradients, Jacobians, Jacobian Chain Rule

Vector function

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

"Gradient" is a horizontal vector of partial derivatives

$$\begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x \\ -\sin(y) \end{bmatrix} = \nabla f_{(x,y)}$$

Notation: upside down triangle

Triangle for change

"Gradient of $f(x,y)$"

Gradients:

vector to scalar

$\mathbb{R}^n \to \mathbb{R}^1$

$x = 2, \; y = 0$

$\begin{bmatrix} x \\ y \end{bmatrix} \xrightarrow{f(x,y)} (5)$ → 5 for Scalar

eg. $\begin{bmatrix} 2 \\ 0 \end{bmatrix} = (5)$

$\mathbb{R}^2 \quad \mathbb{R}^1$

Jacobian:

$\mathbb{R}^n \to \mathbb{R}^m$

partial derivatives,

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

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

Vector to another Vector

since we can vary the input size and size of the Jacobian.

From last page

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

$\mathbb{R}^2 \Longrightarrow \mathbb{R}^2$

$$\left.\begin{array}{ll} \frac{\partial f_1}{\partial x} = 2 & \frac{\partial f_1}{\partial y} = -3y^2 \\ \frac{\partial f_2}{\partial x} = e^x & \frac{\partial f_2}{\partial y} = -13 \end{array}\right\} = J = \begin{bmatrix} 2 & -3y^2 \\ e^x & -13 \end{bmatrix}^T = \begin{bmatrix} \nabla f_1^T \\ \nabla f_2^T \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix}$$

Transpose the matrix to "factor" out x & y.

$f_1 = 2x - y^3$

$f_2 = e^x - 13y$

Scalar Chain Rule

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

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

Let $g = x^2$

$f = \sin(g)$

technical breakdown

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x} = 2x\cos g \quad \text{(substitute back)} \quad = 2x\cos(x^2)$$

Now, extend to: $f = g(h(c(a(x))))$ as a general concept: use product of partial derivatives

also from 3Blue1Brown chain rule video

① How does g change as $\begin{bmatrix} x \\ y \end{bmatrix}$ change?

$$\vec{x} = \begin{pmatrix} x \\ y \end{pmatrix} \quad \frac{\partial \vec{g}}{\partial \vec{x}} = \begin{bmatrix} 2x & 1 \\ 0 & 3y^2 \end{bmatrix}$$

② How does f change as (g_1, g_2) change

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

③ Apply the Scalar chain rule

Vector notation

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

Matrix multiplication

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

What if we cannot use chain rule?

$$f(x,y) = \begin{pmatrix} \sin(x^2+y) \\ \cos(y^3) \\ xy^3 \end{pmatrix}$$

$$g = \begin{pmatrix} x^2+y \\ y^3 \\ 1 \end{pmatrix}$$

leave as is if no intermediate

put a 1 when you are not using chain rule also / no intermediate function

---

Jacobian chain Rule

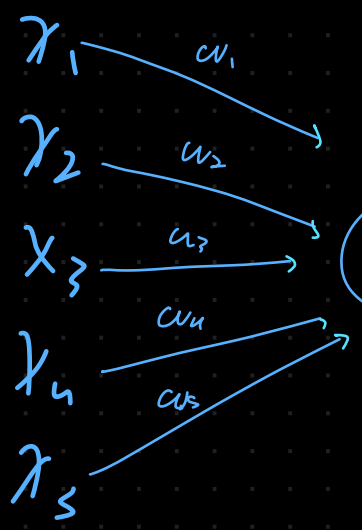
Define the following:

$$f(x,y) = \begin{pmatrix} \sin(x^2+y) \\ \ln(y^3) \end{pmatrix}$$

$$g = \begin{pmatrix} x^2+y \\ y^3 \end{pmatrix} \begin{matrix} g_1 \\ g_2 \end{matrix} \quad f = \begin{pmatrix} \sin(g_1) \\ \ln(g_2) \end{pmatrix}$$

Inside functions — outside functions

Neurons



① $w_1 x_1 + w_2 x_2 \dots$

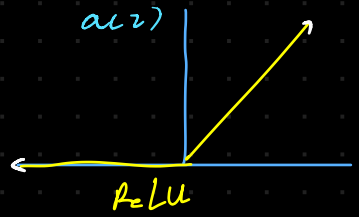
Inputs to one single neuron

② $a\left(\sum_{i=1}^{n} x_i w_i + b\right)$

Scalar

activation function

$a(z)$



★ A neural network with no non-linear activation function is linear regression

③ We define

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}^T \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ \vdots \\ w_n \end{bmatrix} \rightarrow a(\underbrace{\vec{x}^T \vec{w} + b}_{z})$$

and then say

$z = (\vec{x}^T \vec{w} + b)$

$a = a(z)$

Notations

denote as a

Notation for layers

Denote w^{l}_{jk} (think of it as the lines between neurons)

l = layer weight is going into

j = # of the neuron of the layer l

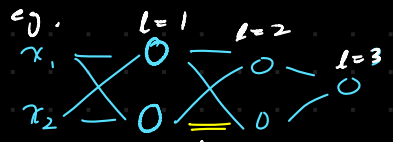
k = # of the neuron in layer $l-1$

b^{l}_{j} or b^{l}

↑ Can also show up

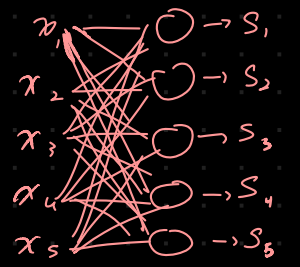
↑ often each layer has one bias

e.g.



w^{2}_{22} (going into layer 2, into neuron #2, from neuron #2)

For one layer: Define scalars

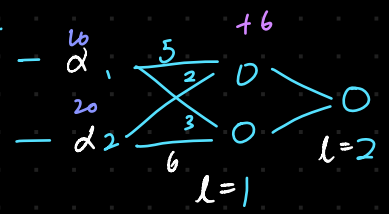


$\rightarrow \vec{a}^{\,l}$ (activation layer) — l for which layer

denote

(a for activation) input goes into

eg.



w^l_{jk} construct the weights matrix

$$w^1 = \begin{bmatrix} w^1_{(1,1)} & w^1_{(1,2)} \\ w^1_{(2,1)} & w^1_{(2,2)} \end{bmatrix} = \begin{bmatrix} 5 & 2 \\ 3 & 6 \end{bmatrix} \quad \begin{matrix} j \\ \\ \end{matrix} \quad \overset{k}{\text{(top)}}$$

$$\sigma(w^l \cdot a^{l-1} + b^l)$$

activation of the previous layer

Calculates the "weighted" sum of the input

$$= \begin{bmatrix} 3 & 2 \\ 3 & 6 \end{bmatrix} \begin{bmatrix} 10 \\ 20 \end{bmatrix} + \begin{bmatrix} b_1 \\ 1 \end{bmatrix}$$

b_1

$$z = \begin{bmatrix} 90 \\ 150 \end{bmatrix} + \begin{bmatrix} 6 \\ 6 \end{bmatrix} = \begin{bmatrix} 96 \\ 156 \end{bmatrix}$$

$$\sigma(z) = \sigma\left(\begin{bmatrix} 96 \\ 156 \end{bmatrix}\right)$$

From single neuron to full layer:

Single node

$\sigma(\vec{x}^T \vec{w} + b)$

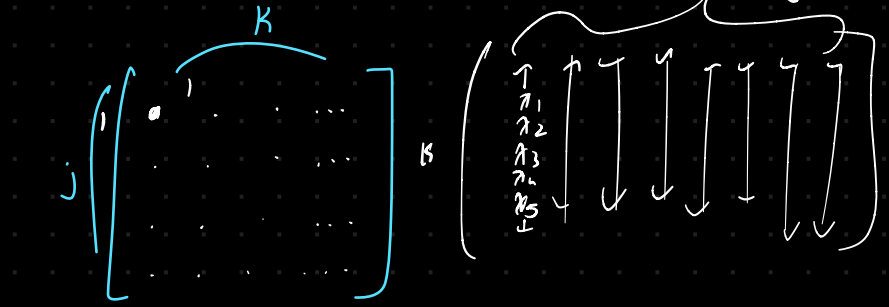
Entire layer

w^l

w^l_{jk} is equivalent to:

Define # of neurons in layer $= k = j$ (usually square)

eg. Coming from the 1st neuron in layer $l-1$, and going into the 1st neuron in layer l



Comes from the kth neuron and goes into the jth neuron

Activation Function Calculation

General: $a_n = \sigma(w^{l+1} a^n + b^l)$

$a_0 \Rightarrow \sigma(w^1 a^0 + b^1)$

Vector → vector — output is a vector

from a_0,

$a_1 = \sigma(w^2 a^1 + b^2)$

$a_2 \rightarrow \dots$

$z = \begin{bmatrix} 96 \\ 156 \end{bmatrix}$

$\sigma(z) = \sigma\left(\begin{bmatrix} 96 \\ 156 \end{bmatrix}\right) = \begin{bmatrix} 96 \\ 156 \end{bmatrix}$

ReLU: $z=0$, $z=z$

Sigmoid: 0, 1

feed forward through an NN

input through the network

forward

backward

$\sigma(z) = \begin{bmatrix} 96 \\ 156 \end{bmatrix}$ stays the same

Jacobians & NNs

Minimize the cost of the output (error)

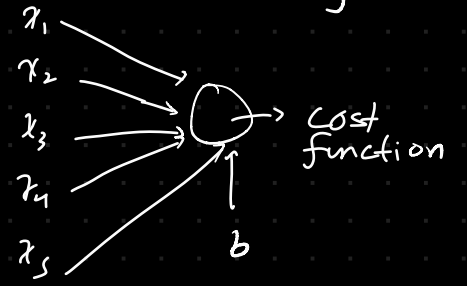
Mean squared error

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

↓ ↓ ↓ answer we got, real answer

$\frac{1}{2}$ does not matter (for the power rule)

m is the # of training examples



Jacobians of Operations

Binary element-wise operations

vector to vector

$F(\vec{v},\vec{w}) \to \vec{z}$

① $\begin{bmatrix}\vec{v}\end{bmatrix} \begin{matrix}\otimes\\ \otimes\\ \otimes\end{matrix} \begin{pmatrix}\vec{w}\end{pmatrix} = \begin{bmatrix}\vdots\end{bmatrix}$

element wise operators (can be any thing, keep it general)

Hadamard product ($\otimes$) is a binary element-wise operation.

③ $J = \frac{\partial \vec{F}}{\partial \vec{v}} = \begin{matrix}F_1\\ F_2\\ \vdots\\ F_n\end{matrix}$

$$\begin{bmatrix} v_1 \otimes w_1 \\ v_2 \otimes w_2 \\ v_3 \otimes w_3 \\ \vdots \\ v_n \otimes v_n \end{bmatrix} = \begin{pmatrix} b \end{pmatrix}$$

② $F(\vec{v},\vec{w})$ (general)

$= (f(\vec{v})) \circ (g(\vec{w})) =$ "operations on the functions before"

$$\begin{bmatrix} f_1(\vec{v}) \circ g_1(\vec{w}) \\ f_2(\vec{v}) \circ g_2(\vec{w}) \\ \vdots \\ f_n(\vec{v}) \circ g_n(\vec{w}) \end{bmatrix} \begin{matrix} F_1 \\ F_2 \\ \vdots \\ F_n \end{matrix}$$

first element of v_0 second element of w

Can get 2 Jacobians one for each vector

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

In the most general form (if $F(\vec{v})$ has multiple element wise types of operations)

General representation

incase f_1 and g_1 don't change the vector

$$\begin{bmatrix} \frac{\partial}{\partial v_1} f_1(\vec{v}) \circ g_1(\vec{w}) & \frac{\partial}{\partial v_2} f_1(\vec{v}) \circ g_1(\vec{w}) & \cdots & \frac{\partial}{\partial v_n} f_1(\vec{v}) \circ g_1(\vec{w}) \\ \frac{\partial}{\partial v_1} f_2(\vec{v}) \circ g_2(\vec{w}) & \frac{\partial}{\partial v_2} f_2(\vec{v}) \circ g_2(\vec{w}) & \cdots & \frac{\partial}{\partial v_n} f_2(\vec{v}) \circ g_2(\vec{w}) \\ \frac{\partial}{\partial v_1} f_n(\vec{v}) \circ g_n(\vec{w}) & \frac{\partial}{\partial v_2} f_n(\vec{v}) \circ g_n(\vec{w}) & \cdots & \frac{\partial}{\partial v_n} f_n(\vec{v}) \circ g_n(\vec{w}) \end{bmatrix}$$

If $f(\vec{v}) = \vec{v}$ and $g(\vec{w}) = \vec{w}$

$$\begin{bmatrix} \frac{\partial}{\partial v_1}(\vec{v}) \circ (\vec{w}) & 0 & \cdots & 0 \\ 0 & \frac{\partial}{\partial v_2}(\vec{v}) \circ (\vec{w}) & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{\partial}{\partial v_n}(\vec{v} \cdot \vec{w}) \end{bmatrix}$$

↑ scalar

* For all element wise functions, the Jacobian will be a diagonal matrix

↳ $\begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & c \end{pmatrix}$ → diag (a, b, c)

Derivatives of Neuron Operators (Hadamond product)

→ Hadamard Product

$$F(\vec{v},\vec{w}) = \begin{bmatrix} f_1(\vec{v}) \otimes g_1(\vec{w}) \\ f_2(\vec{v}) \otimes g_2(\vec{w}) \\ \vdots \\ f_n(\vec{v}) \otimes g_n(\vec{w}) \end{bmatrix} = \begin{bmatrix} v_1 \otimes w_1 \\ v_2 \otimes w_2 \\ \vdots \\ v_n \otimes w_n \end{bmatrix}$$ ⇒ much better description

$$\frac{\partial F}{\partial \vec{v}} = \begin{matrix} F_1 \\ F_2 \\ \vdots \\ F_n \end{matrix} \begin{bmatrix} \frac{\partial F_1}{\partial v_1} & \frac{\partial F_1}{\partial v_2} & \cdots & \frac{\partial F_1}{\partial v_n} \\ \frac{\partial F_2}{\partial v_1} & \frac{\partial F_2}{\partial v_2} & \cdots & \frac{\partial F_2}{\partial v_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial v_1} & \frac{\partial F_n}{\partial v_2} & \cdots & \frac{\partial F_n}{\partial v_n} \end{bmatrix} = \begin{bmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & & \vdots \\ \vdots & & \ddots & \\ 0 & 0 & \cdots & w_n \end{bmatrix}$$

$\frac{\partial F}{\partial \vec{v}}$ ↓ with respect to $\vec{v}$

$\frac{d}{dw}(xw) = x$ (easier easy to understand)

Derivative of a Scalar expansion

$$2\begin{bmatrix} \\ \vec{v} \\ \\ 1 \end{bmatrix} = \begin{bmatrix} 2v_1 \\ 2v_2 \\ 2v_3 \\ \vdots \\ v_n \end{bmatrix}$$

$F(\vec{v}, x) = f(\vec{v}) \circ g(x)$

defining for $g(\vec{v}) = \vec{1} \cdot x$ (expands into a vector)

The act of multiplying x by the ones vector is an act of broadcasting itself (expands to a diag matrix or a Jacobian)

$$\begin{bmatrix} 2 \\ 2 \\ \vdots \\ 2 \end{bmatrix} \circ \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} f_1(\vec{v}) \circ g_1(\vec{x}) \\ f_2(\vec{v}) \circ g_2(\vec{x}) \\ \vdots \\ f_n(\vec{v}) \circ g_n(\vec{x}) \end{bmatrix} \begin{matrix} F_1 \\ F_2 \\ \vdots \\ F_n \end{matrix}$$

don't forget, this is the Scalar vector.

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

When we get the Jacobian, we set a gradient (x is a Scalar)

Gradient with respect to x

$$\nabla_x F = \begin{bmatrix} \frac{\partial f_1}{\partial x} \\ \frac{\partial f_2}{\partial x} \\ \vdots \\ \frac{\partial f_n}{\partial x} \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

Derivative of a Sum

g is element wise

e.g. $\vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \sum_{i=1}^{n} (g(\vec{v}_i))_i = \begin{bmatrix} \text{---} S \text{---} \end{bmatrix}$ for S to denote the sum

$$= \begin{bmatrix} \frac{\partial S}{\partial v_1} & \frac{\partial S}{\partial v_2} & \cdots & \frac{\partial S}{\partial v_n} \end{bmatrix}$$

$$J = \begin{bmatrix} \frac{\partial}{\partial v_1} \sum_{i=1}^{n} g_i(v) & \frac{\partial}{\partial v_2} \sum_{i=1}^{n} g_i(v) & \cdots & \frac{\partial}{\partial v_n} \sum g(v) \end{bmatrix}$$

★ The derivative of a sum is the sum of the derivatives

$$J = \begin{bmatrix} \sum_{i=1}^{n} \frac{\partial}{\partial v_1} g_i(v) & \sum_{i=1}^{n} \frac{\partial}{\partial v_2} g_i(v) & \cdots & \sum_{i=1}^{n} \frac{\partial}{\partial v_n} g_i(v) \end{bmatrix}$$

Easy to generalize

If $g(\vec{v}) = 2\vec{v}$ → $[2\ 2\ 2\ \dots\ 2]$

Consider $g(\vec{v}) = \vec{v}$

$$J = \begin{bmatrix} \sum_{i=1}^{n} \frac{\partial}{\partial v_1}(v_i) & \sum_{i=1}^{n} \frac{\partial}{\partial v_2}(v_i) & \cdots & \sum_{i=1}^{n} \frac{\partial}{\partial v_n}(v_i) \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & \dots & 1 \end{bmatrix}$$

1 + 0 + 0 + ... 0 + 1 + 0 + ... 0 + 0 + ... + 1

Basically a horizontal line of what g does if it does anything

Derivative of a neuron

activation
$a = \sigma(\vec{w}^T \vec{x} + \vec{b})$ dot product = Hadamard product

(the part $\vec{w}^T \vec{x} + \vec{b}$ is labelled z)

we want:

$$\frac{\partial \vec{a}}{\partial \vec{w}} = \frac{\partial \vec{a}}{\partial \vec{z}} \frac{\partial \vec{z}}{\partial \vec{w}}$$

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

Define an activation like:

$$a = \sigma(\text{Sum}(w \otimes x) + b)$$

($w \otimes x$ is H, the Hadamard product; $\text{Sum}(H)$ is $S(H)$)

By Jacobian convention, $H = w \otimes x$

① $$\frac{\partial H}{\partial w} = \begin{bmatrix} x_1 & \cdots & 0 \\ \vdots & x_2 & \vdots \\ 0 & \cdots & x_n \end{bmatrix}$$
$\text{diag}(x_1 \ldots x_n)$

② $z = S(H) + b$
$\frac{\partial z}{\partial H} = 1$
for $S(H)$
$$\frac{\partial s}{\partial H} = (1\ 1\ 1\ 1\ 1\ 1]$$

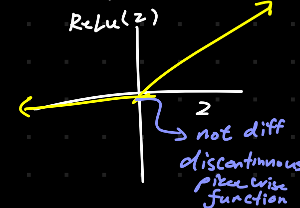
③ $z = S - b$
$$\frac{\partial z}{\partial s} = 1$$
(scalar)

④ Derivative of ReLU wrt the activation

$$\frac{\partial a}{\partial w} = \frac{\partial a}{\partial z}[\vec{x}^T]$$
↓ Simplifies after all 3 substitutions

Need to find this

$\max(0, z)$ of 2 numbers
($z = \Sigma(\vec{w}\,\vec{x}) + b$)



, but in practice, does not matter

To solve, we split up into piecewise functions

taking the derivative

$$\frac{\partial a}{\partial w} = \max \begin{cases} z \le 0,\ 0 \to \vec{0} \times \frac{\partial z}{\partial w} = \vec{0} = [\vec{0}] \\ z > 0,\ z \to 1 \times \frac{\partial z}{\partial w} = \frac{\partial z}{\partial w} = [x^T] \end{cases}$$

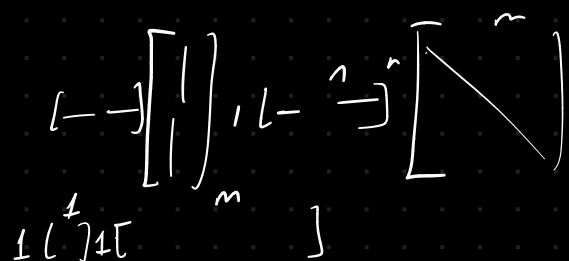
↓ also putting all derivatives together

For some w, $\frac{\partial a}{\partial w}$ gives the best approximation for the instant rate of change of the activation, wrt that specific w

$$\frac{\partial a}{\partial w} = \frac{\partial a}{\partial z}\ \frac{\partial z}{\partial s}\ \frac{\partial s}{\partial H}\ \frac{\partial H}{\partial w}$$
(④ ③ ② ①)

$\frac{\partial a}{\partial z}$ → either 0 or 1

1^T → previous slide



our final answer is:

$$\frac{\partial a}{\partial w} = [0\ 0\ 0\ \ldots\ 0]$$
or
$$\frac{\partial a}{\partial w} = (x_1, x_2, x_3 \ldots x_n]$$

depending on $\vec{w}^T\vec{x} + \vec{b}$

★ The derivative of the cost wrt a weight $\vec{w}$ is directly proportional to the input $\vec{x}$

$\sigma(\Sigma(w \otimes x) + b)$
(the part $\Sigma(w \otimes x) + b$ is z)

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

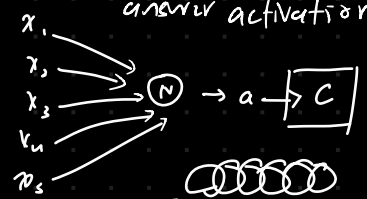
$$\frac{\partial a}{\partial b} = \begin{cases} 0 \times 1 = \vec{0} \text{ if } z \le 0 & = [\vec{0}] \\ \vec{1} \times \vec{1} = [\vec{1}] \text{ if } z > 0 \end{cases}$$

(important so that the vector operations make sense dimensionally)

The Gradient of the Loss Function

Cost function

MSE: $\frac{1}{2m}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$ ← answer, ↓ activation a^L



we want to find:

$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial a}\frac{\partial a}{\partial w}$ (chain rule)

$\frac{\partial C}{\partial b}$ ↑ know this already

Recall that

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

$\frac{\partial a}{\partial b} = \begin{cases} [0]^T & \text{if } w^T x + b \le 0 \\ [1]^T & \text{if } w^T x + b > 0 \end{cases}$

one training example $\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}$

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

several training examples (m)

$\begin{bmatrix} x_{11} & x_{21} & \cdots & x_{n1} \\ x_{12} & x_{22} & & \\ \vdots & \vdots & \ddots & \\ x_{1n} & x_{2n} & \cdots & x_{nn} \end{bmatrix}$

correct answer

Define and Say: $v = y - a^L$

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

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

$\frac{\partial C}{\partial w} = \frac{\partial C}{\partial v}\frac{\partial v}{\partial w}$ ← we have $\frac{\partial a}{\partial w}$

① $\frac{\partial v}{\partial w} = \frac{\partial}{\partial w}(y - a^L)$ — don't forget that L is not actually a constant in the equation

Find $\frac{\partial C}{\partial v}$ $\frac{\partial v}{\partial w} = -(1)\left(\frac{\partial a}{\partial w}\right)$

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

Take out the constant $= \frac{1}{2m}\left(\sum_{i=1}^{m}\frac{\partial}{\partial w}(v^2)\right)$ $\left(\frac{dv^2}{dw} = \frac{dv^2}{dv}\frac{dv}{dw}\right)$ (known)

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

$= \frac{1}{m}\sum_{i=1}^{m}(v)\left(\frac{\partial v}{\partial w}\right)$ $\left(\frac{\partial v}{\partial w} = -\frac{\partial a}{\partial w}\right)$

what does this mean? ↓ looking at a piecewise → make the options negative

$= \frac{1}{m}\sum_{i=1}^{m}\begin{cases} [\vec{0}]^T & \text{if } \dots \\ -v\frac{\partial z}{\partial w} & \text{if } \dots \end{cases}$ ✱ DOES not actually change the piecewise

Very important to understand that it is just attached to the result of the piecewise

Subbing in the variables, $= \frac{1}{m}\sum_{i=1}^{m}\begin{cases} [0]^T & \text{if } \dots \\ -(y_i - a^L)[\vec{x}^T] \end{cases}$

From ReLU,

$\sigma(w^T x + b) \to \max(0, w^t x + b)$

$\Rightarrow \frac{1}{m}\sum_{i=1}^{m}\begin{cases} [0]^T & \text{if } \dots \\ -\{y_i - (\max(0, \vec{w}^t\vec{x} + b))\}[\vec{x}^T] \end{cases}$

if $w^t x + b > 0$ ← always the case if it went down this path. max function is redundant

$= -(y_i - (w^t x + b))[\vec{x}^T]$

$\frac{\partial C}{\partial w} = \frac{1}{m}\sum_{i=1}^{m}(\vec{w}^t\vec{x} + \vec{b} - \vec{y}_i)[\vec{x}^t]$

$= \text{scalar} \quad 1 \quad [\ ] \quad 1 \quad [\ \ \ ] \quad n$

$\frac{1}{m}\begin{cases} \vec{0}_t \\ \sum_{i=1}^{m}(w^t x + b - y)(x^t) \end{cases}$

what did we find out?

want a Jacobian for back propagation

$$\frac{\partial C}{\partial w} = \frac{1}{n}\begin{cases} \vec{0}^T & \text{if } w^t x + b \le 0 \\ \sum_{i=1}^{n} (w^t x + b - y_i)(x^t) & \text{if } w^t x + b > 0 \end{cases}$$

$(w^t x + b - y_i)$: e_i (called error)

Want:

$$\begin{bmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \frac{\partial C}{\partial w_3} \\ \vdots \\ \frac{\partial C}{\partial w_n} \end{bmatrix}^T$$

w_n total weights

$$\Rightarrow \boxed{\frac{1}{n}\sum_{i=1}^{n} e_i [x]}$$

where $e_i = w^t \vec{x} - y_i$ (scalar)

Consider $n=1$ and 5 inputs, transposing our $e_i[x]^T$ so it becomes a column,

$$\frac{\partial C}{\partial \vec{w}} = \begin{bmatrix} e x_1 \\ e x_2 \\ \vdots \\ e x_5 \end{bmatrix} \begin{matrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \vdots \\ \frac{\partial C}{\partial w_3} \end{matrix}$$

how C changes wrt w_1 ...

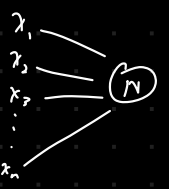
how C changes wrt w_2 ...

when you change w_1 by some amount, the corresponding change is ...

consider $\frac{1}{n}\sum_{i=1}^{n} e_i x^T$

$$= \frac{1}{n}\left(e_1 x^T + e_2 x^T + e_3 x^T \dots e_5 x^T\right)$$

$$\frac{1}{n}\left(\begin{bmatrix} e_1 x_1 \\ e_2 x_1 \\ \vdots \\ e_5 x_1 \end{bmatrix} + \begin{bmatrix} e_1 x_1 \\ e_2 x_1 \\ \vdots \\ e_5 x_2 \end{bmatrix} \dots + \begin{bmatrix} e_5 x_1 \\ e_5 x_2 \\ \vdots \\ e_5 x_5 \end{bmatrix}\right)$$



Consider a general case

$$\frac{1}{n}\begin{bmatrix} e_1 x_1 + e_2 x_1 \dots & e_5 x_1 \\ e_1 x_2 + e_2 x_2 \dots & e_5 x_2 \\ e_1 x_3 + e_2 x_3 \dots & e_5 x_3 \\ e_1 x_4 + e_2 x_4 \dots & e_5 x_4 \\ e_1 x_5 + e_2 x_5 \dots & e_5 x_5 \end{bmatrix}$$

random averages

$$\Rightarrow \begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ g_4 \\ g_5 \end{bmatrix} = \frac{\partial C}{\partial \vec{w}}$$

derivative of the cost with respect to all w's, averaged out over all training examples

Derivative of the bias

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

$v = y - a^L$

From $\frac{\partial C}{\partial b} = \frac{\partial C}{\partial v}\frac{\partial v}{\partial a^L}\frac{\partial a^L}{\partial b}$ (-1) ✓

$$\frac{\partial v}{\partial b} = -1 \begin{cases} [\vec{0}]^T & \text{if } w^t x + b \le 0 \\ 1 & \text{if } w^t x + b > 0 \end{cases}$$

* Scalar *

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

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

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

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

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

$$= (-1)\begin{cases} 0 & \text{if } \dots \\ 1 & \text{if } \dots \end{cases}$$

$$= \frac{1}{m}\sum_{i=1}^{m} v \begin{cases} 0 & \text{if } \dots \\ -1 & \text{if } \dots \end{cases} \quad \left(\begin{array}{l}\text{plug into} \\ \text{our piecewise}\end{array}\right)$$

$$= \frac{1}{m}\sum_{i=1}^{m}\begin{cases} 0 & \text{if } \dots \\ -(y - a^L) & \text{if } \dots \end{cases}$$

↓ max redundancy again, can remove

$$= \frac{1}{m}\sum_{i=1}^{m}\left\{(w^t x + b - y)\right.$$

$$= \begin{cases} 0 & \text{if } \dots \quad w^t x \le 0 \\ \frac{1}{m}\sum_{i=1}^{m}\underbrace{(w^t x + b - y)}_{c_i} & \text{if } \dots \quad w^t x + b > 0 \end{cases}$$

Now generally, $\frac{\partial C}{\partial b} = \frac{1}{m}\sum_{i=1}^{m} c_i$

$$\boxed{\frac{\partial C}{\partial b} = \frac{1}{m}\left[c_1 + c_2 + \dots c_n\right]}$$

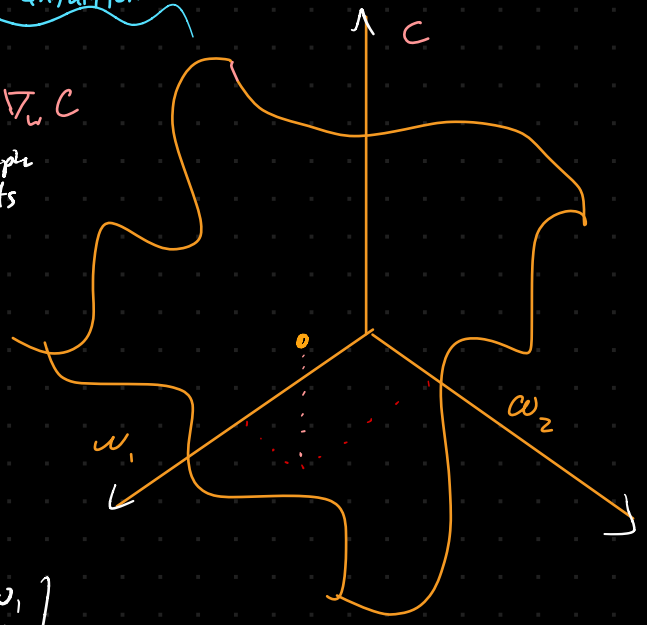
Just a scalar

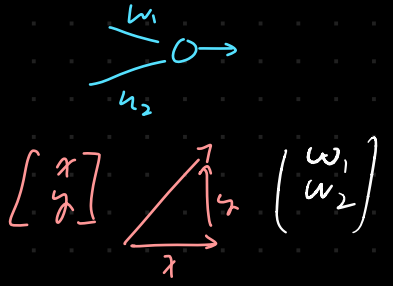
derivative of the cost wrt to the bias, averaged over all training examples.

Gradient Descent Intuition

$$\frac{\partial C}{\partial w} = \begin{bmatrix} e_i x_1 \\ e_i x_2 \end{bmatrix} \nabla_w C$$

one training example
2 inputs, 2 weights





The gradient of a function points in the direction of steepest ascent

Vectors can be translated

The value of the input decides the derivative

want to change the weights of the corresponding input that causes the greatest effect on the cost function

$\begin{bmatrix} e_i x_1 \\ e_i x_2 \end{bmatrix}$ will point in the direction of the weights that have the most impact (evaluate $e_i x_1$ first)

$\nabla_w C$ goes in a higher direction of cost — error is magnified

$-\nabla_w C$ tells us how to most quickly decrease the cost

The vector points in the direction that have the most impact (cost activation is x_i error is magnified by x_i)

This we want to do gradient descent in the opposite direction

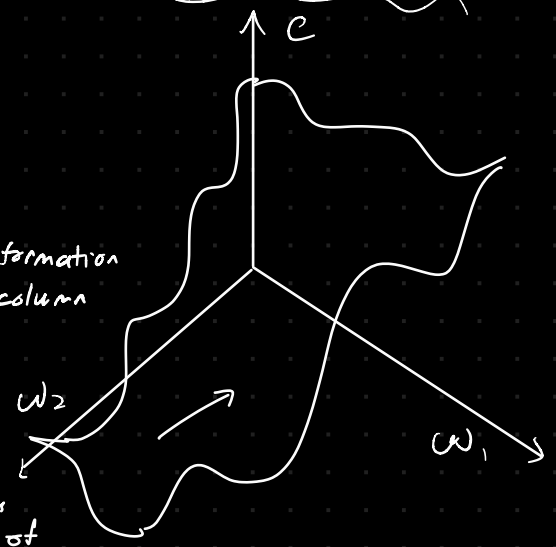
All machine learning models optimize to point in the direction of steepest descent

Gradient Descent & Stochastic Gradient Descent

$$\begin{bmatrix} \frac{\partial C}{\partial w_1} \\ \frac{\partial C}{\partial w_2} \\ \frac{\partial C}{\partial w_3} \\ \vdots \\ \frac{\partial C}{\partial b_1} \\ \frac{\partial C}{\partial w_n} \end{bmatrix}$$

$\nabla_{w,b} C$

put all information in one column vector.



The gradient points in the direction of steepest ascent

θ (defined to be all weights and biases)

Initially starts as randomly assigned

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

↓ learning rate (step size)

$$\begin{bmatrix} w_1 - \alpha \frac{\partial C}{\partial w_1} \\ w_2 - \alpha \frac{\partial C}{\partial w_2} \\ \vdots \\ w_n - \alpha \frac{\partial C}{\partial w_n} \\ b_1 - \alpha \frac{\partial C}{\partial b_1} \\ b_2 - \alpha \frac{\partial C}{\partial b_2} \\ \vdots \\ b_n - \alpha \frac{\partial C}{\partial b_n} \end{bmatrix}$$

α (learning rate) depends on research

Training batch: m
↳ Can be very computationally expensive.

$$\begin{bmatrix} | \\ \vec{q} \\ | \end{bmatrix}$$ over m examples

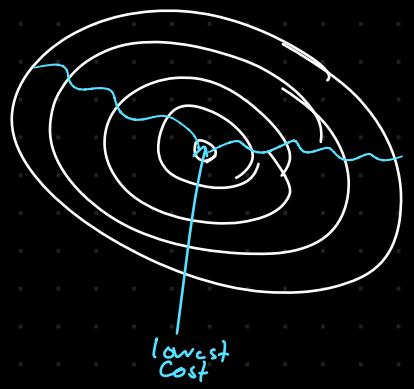
"Mini-batch" Gradient descent:

SGD → batching smaller datasets and then, running gradient descent on each batch

Shuffle the data well. Batch should represent data as a whole.

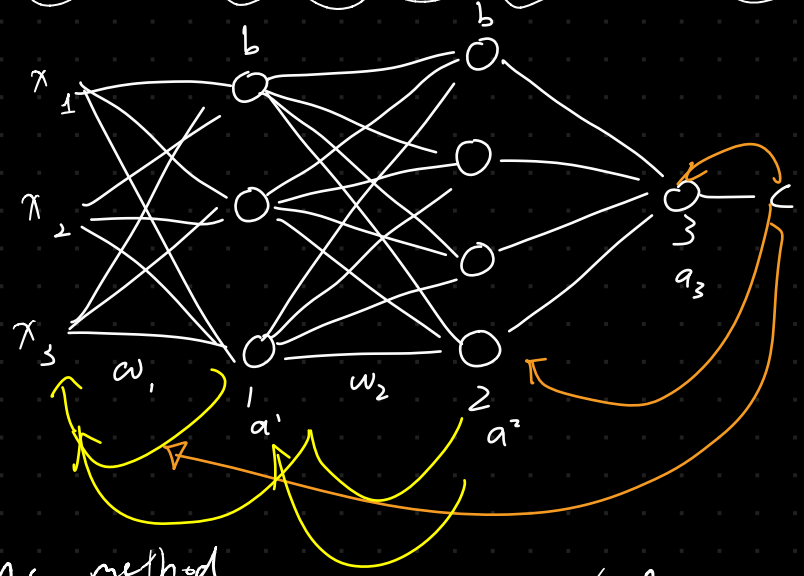
$\{x_1\}$ $\{x_2\}$

↓ SGD ↓ SGD



then, you can take as many steps, as you have batches instead of one step over all examples.

Finding derivatives of entire layers (why it is also different from one neuron)



Back propagation - saves efficiency by not having to go back and forth

Accumulate our "error"

one method

$$\frac{\partial C}{\partial w_1} = \left(\frac{\partial a^1}{\partial w_1}\right)\left(\frac{\partial a^2}{\partial a^1}\right)\left(\frac{\partial a^3}{\partial a^2}\right)\left(\frac{\partial C}{\partial a^3}\right)$$

$$\left(\frac{\text{first layer}}{\text{weights}}\right)\left(\frac{\text{second layer}}{\text{first layer}}\right)\left(\frac{\text{third layer}}{\text{second layer}}\right)\left(\frac{\text{cost}}{\text{third layer}}\right)$$

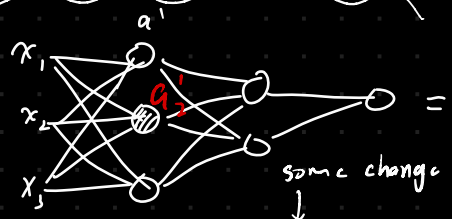
$$\frac{\partial C}{\partial w_2} = \frac{\partial a^2}{\partial w^2}\frac{\partial a^3}{\partial a^2}\frac{\partial C}{\partial a^3}$$

Doing these calculations to compute the derivatives going forward is resource-intensive
Not often done, but possible

error of node j, of layer l

δ_j^l

The error of a node



Activation of node 2 of layer 1, before ReLU

$(z^1_2 + \Delta)$ is the "total error"

If we add Δ to a node, how does the cost function change?

General form error of a node (impact of a node on the cost)

$$\delta^l_j = \frac{\partial C}{\partial z^l_j}$$

$$\delta^1_2 = \frac{\partial C}{\partial z^1_2}$$

→ if close to 0, this means that changing the value of z has little effect on the cost. Vice versa.

How much "error" is being caused by a node

"Bang for your buck"

1st equation – error of last node a^L

$$a^L = \sigma(z^L)$$

$$\delta^l_j = \frac{\partial C}{\partial z^l_j}$$

$$\delta^1_j = \left(\frac{\partial C}{\partial a^1_j}\right)\left(\frac{\partial a^1_j}{\partial z^1_j}\right)$$ chain rule again

$$\delta^1_j = \frac{\partial C}{\partial a^1_j} \cdot \sigma'(z^1_j)$$ general form

Now, do the entire last layer:

$\nabla_{a^l} C$ (gradient of cost wrt a^l)

$$\begin{bmatrix} \frac{\partial C}{\partial a^l} \\ \frac{\partial C}{\partial a^l} \\ \frac{\partial C}{\partial a^l} \\ \vdots \end{bmatrix} \otimes \begin{bmatrix} \sigma'(z_1) \\ \sigma'(z_2) \\ \sigma'(z_3) \\ \vdots \end{bmatrix}$$

element wise ↑ derivative ↓

$$\delta^l = \nabla_{a^l} C \otimes \sigma'(z^L)$$

element-wise operation

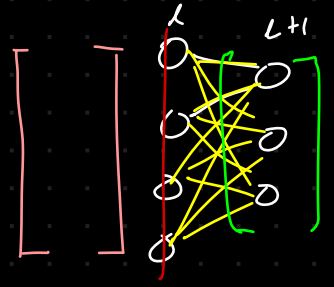
2nd equation: error of any node: $\delta^l = ((w^{l+1})^T \otimes \delta^{l+1}) \otimes \sigma'(z^l)$

= weights going into layer $l+1$

↓ weight matrix connecting $l, l+1$

↘ total error of $l+1$ layer

derivatives of the activations

$\delta^l =$ 

why does this work:

If we know errors of layer $l+1$

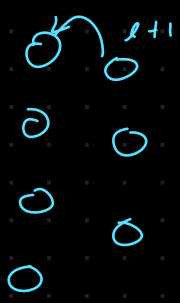
$(w^{l+1})^T$ is like going back and getting that error

eg. Find $\frac{\partial C}{\partial z_1}$, given $\frac{\partial C}{\partial z_2}$

z_1 z_2

$\sigma(w_1 a_0 + b_1) + \sigma(w_2 a_1 + b_2)$

given



$w = \begin{bmatrix} & \\ & \end{bmatrix}$ (K columns, j rows)

$w^T = \begin{bmatrix} & \\ & \end{bmatrix}$ (j columns, K rows)

Reversing through the network

using the chain rule:

$$\frac{\partial C}{z_1} = \frac{\partial z_2}{\partial z_1} \frac{\partial C}{\partial z_2}$$

$$\frac{\partial C}{z_1} = \frac{\partial a_1}{\partial z_1} \frac{\partial z_2}{\partial a_1} \frac{\partial C}{\partial z_2}$$

$$\sigma'(z_1) \otimes w_2^T \otimes \delta^2$$

$z_1 = w_2 a_1$

$\frac{\partial z_1}{\partial a_1} = w_2^T$ ← Transposed

Generally, derivative of matrix-vector product wrt the vector is,

$\frac{\partial Ab}{\partial b} = A^T$

works for any layer

Equation 3 – derivative of cost wrt any bias

$$\frac{\partial C}{\partial b^l} = \delta^l$$

$$\begin{bmatrix} 4 \\ 4 \\ 4 \\ 4 \\ 4 \end{bmatrix} \quad \begin{matrix} 0 \\ 0 \\ 0 \\ 0 \end{matrix} = \begin{bmatrix} \quad \end{bmatrix}$$

eg. $\sigma(w_1 a_0 + b_1) \Rightarrow \sigma(w_2 a_1 + b_2)$

z_1; a_1, a_2

given δ^1 (error of z_1)

$$\delta^1 = \frac{\partial C}{\partial z_1}$$

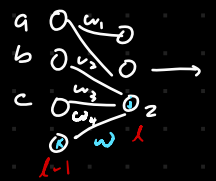
$$\frac{\partial C}{\partial b_1} = \left(\frac{\partial C}{\partial z_1}\right)\left(\frac{\partial z_1}{\partial b_1}\right)$$

$\frac{\partial C}{\partial z_1} = \delta^1$, $\frac{\partial z_1}{\partial b_1} = 1$

$$\boxed{\frac{\partial C}{\partial b_1} = \delta^1}$$

Equation 4 – find the derivative of the cost wrt any weight

$$\frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \quad (\text{Scalar})$$



have $\frac{\partial C}{\partial z^l_j}$ δ

know $\frac{\partial C}{\partial w^l_{jk}} = \frac{\partial C}{\partial z^l_j} \frac{\partial z^l_j}{\partial w^l_{jk}}$

(Taking derivative)

$z^l_j = (a w_1 + b w_2 + c w_3 + k w + b)$

dot product

$z^l_j = (a w_1 + b w_2 + c w_3 + k w + b)$

$\frac{\partial z^l_j}{\partial w} = k$

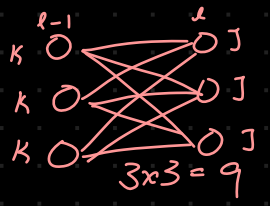
what is k? activation of node $l-1$ from node k

Vectorized form:

Example:

from kth, into jth

error of the node its coming from in l



$$w_l = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix} \begin{matrix} K \\ \\ J \end{matrix}$$

(3, 3)

$$\frac{\partial C}{\partial w_l} \begin{bmatrix} \frac{\partial C}{\partial w_{11}} & \frac{\partial C}{\partial w_{12}} & \frac{\partial C}{\partial w_{13}} \\ \vdots & \vdots & \vdots \end{bmatrix}$$

$$\frac{\partial C}{\partial w_2} = \begin{bmatrix} a^1_1 \delta^2_1 & a^1_2 \delta^2_1 & a^1_3 \delta^2_1 \\ a^1_1 \delta^2_2 & a^1_2 \delta^2_2 & a^1_3 \delta^2_2 \\ a^1_1 \delta_3 & a^1_2 \delta_3 & a^1_3 \delta_3 \end{bmatrix}$$

Calculate like:

$$\delta^2 = \begin{bmatrix} \delta^2_1 \\ \delta^2_2 \\ \delta^2_3 \end{bmatrix}^3 \qquad a_1 = \begin{bmatrix} a^1_1 \\ a^1_2 \\ a^1_3 \end{bmatrix}^1$$

$$\boxed{(\vec{\delta}^2)(\vec{a}_1^{\,T}) = \frac{\partial C}{\partial w_l}}$$

by matrix vector multiplication

(n, 1) (1, m)

n m

0 0

0 0

0 0

0 0

Tying everything together

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

(e_i)

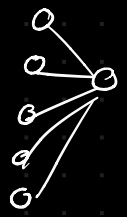
$$\frac{\partial C}{\partial w} \begin{cases} \vec{0} & \\ \frac{1}{M}\sum_{i=1}^{m} (w^T x + b - y)\, x^T & \end{cases}$$

(e_i)

$$\frac{\partial C}{\partial b} = e^i = \frac{\partial C}{\partial b} = \delta^l$$

$$\frac{\partial C}{\partial w} = e_i x^T = \frac{\partial C}{\partial w^l} = \delta^l (a^{l-1})^T$$

These are equivalent



e^i

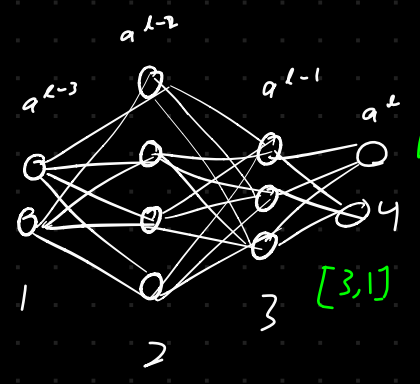
$e_i x^T$ are only wrt one node

BP Equations

1. $\delta^L = \nabla_a C \circ \sigma'(z^L)$
2. $\delta^l = ((w^{l+1})^T \delta^{l+1}) \circ \sigma'(z^l)$
3. $\frac{\partial C}{\partial b^l} = \delta^l$
4. $\frac{\partial C}{\partial w^l} = \delta^l (a^{l-1})^T$

---

① Feed forward and store z^l and a^l values of all l layers



$[1] \; j \; [w_{11} \; w_{12} \; w_{13}]$ (k)

$\downarrow$

$[1,3]$

$\downarrow$

$[3,1]$

0 Keeps dimension

② compute the cost

③ use eq 1 to calculate δ^L

④ use eq 2 to compute all δ^l terms

⑤ use eq 3 to find bias derivatives $\frac{\partial C}{\partial b^l}$

⑥ use eq 4 to find weight derivatives $\frac{\partial C}{\partial w^l}$

$$\nabla_{w,b} C = \begin{bmatrix} \\ \\ \end{bmatrix}$$

unroll all matrices / flatten

$$\begin{bmatrix} a & b \\ c & d \\ e & f \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix}$$

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

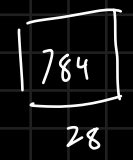
forward pass →

back propagation ←

back propagation ←

C ∪

start

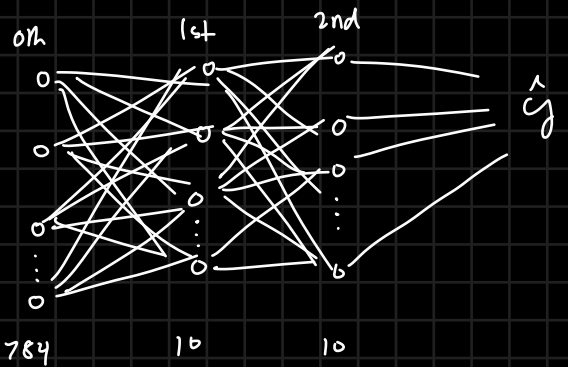


n training images

$dz' =$

$x = \begin{bmatrix} x^1 & x^2 \cdots & x^m \end{bmatrix} \Rightarrow$ 0, 1, 2, 3, ... 9

10 classes



① Forward propogation

$A^0 = X$ (first layer)

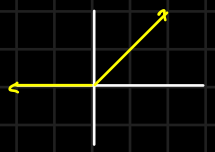
$Z^1 = W^1 A^0 + b^1$

$10 \times m \quad 10 \times 784 \quad 784 \times m \quad 10 \times m$

Apply activation function (otherwise layers are just linear combinations of previous layers)

Use ReLU (Z^1)

adds complexity rather than a linear model



$Z^2 = W^2 A^1 + b^2$

$A^2 = \text{Softmax}(A^2)$ $\quad \frac{e^{z_i}}{\sum_{i=1}^{n} e^{z_i}}$

our outputs are converted to a probability between 0 and 1