CALCULUS FOR DATA SCIENCE

A workshop by Shpresim Sadiku
Institute of Mathematics, Technische Universität Berlin

DS3
Data Science Summer School

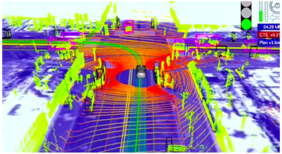
Hertie School
Data Science Lab

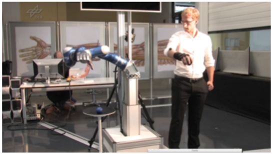
Lecturer

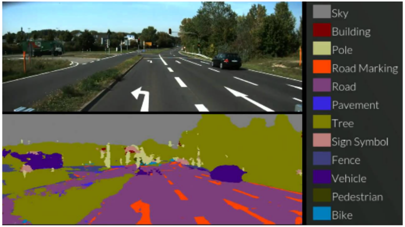
- Shpresim Sadiku
- sadiku@zib.de
- www.shpresimsadiku.com
- PhD Candidate in Mathematics at TU Berlin
- Scientific Assistant at Zuse Institute Berlin
- Passionate about Artificial Intelligence
- I love to travel and lift weights

Why Data Science?

Self-driving cars and robotics

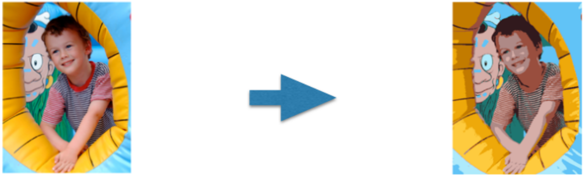




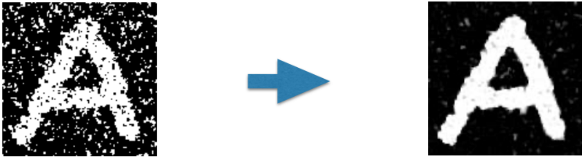


Typical problems in Data Science

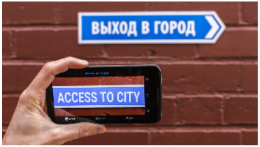
- Image Compression

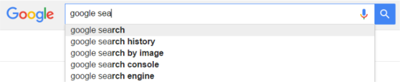


- Noise Reduction



Natural Language Processing





Prerequisites for Data Science

Mathematical background in

- Linear Algebra (August 16)
- Calculus (Today)
- Statistics and Probability Theory (August 18)

Outline

- Variables and Functions
- Limits
- Derivatives
- Integrals
- Gradient Descent
- Matrix Calculus
- The Hessian
- Least Squares
- Eigenvalues as Optimization
- The Perceptron Algorithm
- Perceptron via gradient descent
- Gradients of a Neural Networks
- Numerical gradient computation
- Backpropagation algorithm
  - Chain rule and multivariate chain rule
  - Backpropagation through example
  - Formalization of backpropagation
  - Vanishing gradients
  - Choice of nonlinear activation functions
  - Automatic differentiation

Numbers

- ***Natural numbers*** - $1, 2, 3, 4, 5...$
- ***Whole numbers*** - introduce 0 for numbers greater than 9 such as $10, 1000, 1090$
- ***Integers*** $..., -2, -1, 0, 1, 2, ...$
- ***Rational numbers*** - any number that can be expressed as a fraction $\frac{2}{3}, \frac{687}{100}, 2$
  - Note all finite decimals and integers are also rational
- ***Irrational numbers*** - cannot be expressed as a fraction $\pi, \sqrt{2}, e$
  - Infinite number of decimal digits (3.14159265358979323846...)
  - Prove that $\sqrt{2}$ is irrational (!)
- ***Real numbers*** - rational and irrational numbers
- ***Complex and imaginary numbers*** - encountered when taking square root of a negative number
  - In data science for e.g. matrix decomposition

Order of Operations

1. Parentheses
2. Exponents
3. Multiplication
4. Division
5. Addition
6. Subtraction

$$
\begin{aligned}
2 \times \frac{(3+2)^2}{5} - 4 \\
2 \times \frac{(5)^2}{5} - 4 \\
2 \times \frac{25}{5} - 4 \\
\frac{50}{5} - 4 \\
10 - 4 \\
6
\end{aligned}
$$

Variables and Functions

- A ***variable*** is a named placeholder for an unspecified or unknown number
  - Denoted by α, β, θ
- Can represent any real number, can do math operations with it

- ***Functions*** define relationships between two or more variables
- Take *input variables*, plug them into an expression, and result in an *output variable*

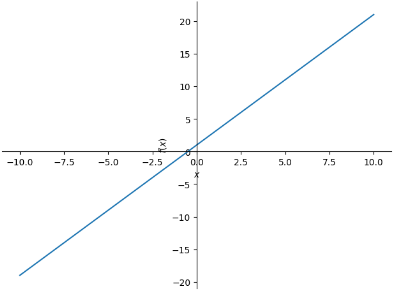
$y = 2x + 1$

x	2x+1	y
0	2(0)+1	1
1	2(1)+1	3
2	2(2)+1	5
3	2(3)+1	7

- Can also be expressed as $f(x) = 2x + 1$

Continuous Functions

- Making steps of x infinitely small then $y = 2x + 1$ is a ***continuous function***
    - For every possible value of x there is a value of y



Exercises

- Plot $f(x) = x^2 + 1$
- Plot $f(x, y) = 2x + 3y$

Logarithms

- ***Logarithm*** is a math function that finds a power for a specific number and base
  - Applications in measuring earthquakes, managing volume on your stereo
  - Used in logistic regression
- E.g. $2^x = 8$ or $x = \log_2 8 = 3$
- In general $a^x = b \iff \log_a b = x$
  - Default base in earthquake measurements is 10
  - Default base in data science and Python is e

  Properties
  - $\log(a \times b) = \log(a) + \log(b)$
  - $\log(\frac{a}{b}) = \log(a) - \log(b)$
  - $\log(a^n) = n \times \log(a)$
  - $\log(1) = 0$
  - $\log(x^{-1}) = \log(\frac{1}{x}) = -\log(x)$

Euler's Number e

- e is resulting value of $(1+\frac{1}{n})^n$ as n gets bigger and bigger

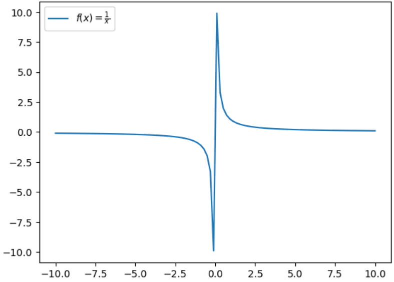
$$\left(1+\frac{1}{100}\right)^{100} = 2.79481382942$$
$$\left(1+\frac{1}{1000}\right)^{1000} = 2.71692393224$$
$$\left(1+\frac{1}{10000}\right)^{10000} = 2.71814592682$$
$$\left(1+\frac{1}{10000000}\right)^{10000000} = 2.71828169413$$

- As n gets larger it converges approximately on 2.71828 which gives e

Limits

- e - increasing input variable the output keeps approaching a value but never reaches it
- As x increases forever, $f(x)$ gets closer to 0 but never reaches it

$$\lim_{x \to \infty} \frac{1}{x} = 0$$



- $\lim_{n \to \infty} \left(1 + \frac{1}{n}\right)^n = e = 2.71828169413$

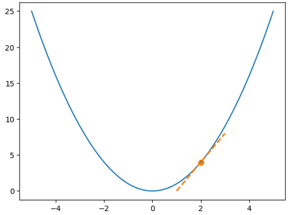
Derivatives

- ***Derivative*** - gives the slope of a function
  - Measures the rate of change at any point in a function
  - Derivatives are used in ML algorithms, e.g. gradient descent
- When slope is 0, we are at the minimum or maximum of an output variable

- $f(x) = x^2$
- Measure steepness at any point in curve, visualize with a tangent line
- $x = 2$ and $x = 2.1$
- $f(x) = 4$ and $f(x) = 4.41$
- Calculate slope m between two points

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{4.41 - 4.0}{2.1 - 2.0} = 4.1$$

- If $x_2 = 2.00001$ then $m = 4.00004$ very close to actual slope of 4



Derivatives

- Exponential function like $f(x) = x^2$ - derivative will make exponent a multiplier and decrement exponent by 1

$$\frac{d}{dx} f(x) = \frac{d}{dx} x^2 = 2x$$
$$\frac{d}{dx} f(2) = 2(2) = 4$$

- Use Python library SymPy to calculate derivatives
- Formal definition

$$f(x)^{'} = \lim_{s \to 0} \frac{(x+s)^2 - x^2}{(x+s) - x}$$

  - $\lim_{s \to \infty} \frac{(2+s)^2 - 2^2}{(2+s) - 2} = 4$

Partial Derivatives

- Slopes wrt multiple variables in several directions
- For each given variable, assume other variables are constant
- $f(x, y) = 2x^3 + 3y^3$

$$\frac{d}{dx} 2x^3 + 3y^3 = 6x^2$$
$$\frac{d}{dy} 2x^3 + 3y^3 = 9y^2$$

- For (x, y) values $(1, 2)$, slope wrt x is $6(1) = 6$ and wrt y is $9(2)^2 = 36$
- Forever approaching step size s to 0 but never reaching it (otherwise no line), we converge on a slope of 4

The Chain Rule

$$y = x^2 + 1, \quad z = y^3 - 2$$

1. Substitute first function y into second function z

$$\begin{aligned} z &= (x^2+1)^3 - 2 \\ \frac{dz}{dx}((x^2+1)^3 - 2) &= 6x(x^2+1)^2 \end{aligned}$$

2. Take derivatives of y and z separately, then multiply them

$$\frac{dy}{dx}(x^2+1) = 2x$$

$$\frac{dz}{dy}(y^3-2) = 3y^1$$

$$\frac{dz}{dx} = (2x)(3y^2) = 6xy^2$$

   - Substitute y

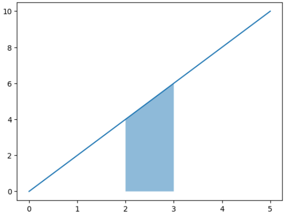
$$\frac{dz}{dx} = 6xy^2 = 6x(x^2+1)^2$$

- The ***chain rule***

$$\frac{dz}{dx} = \frac{dz}{dy} \times \frac{dy}{dx}$$

Integrals

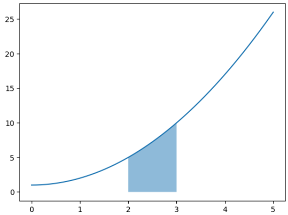
- Opposite of derivative is integral
- Finds area under the curve for a given range
- Area for a range under a straight line is easy



- $f(x) = 2x$
- Measure area under the line between 2 and 3
- Area of a trapezoid $\frac{(4+6)}{2} \times 1 = 5$

Integrals

- What if the function is more difficult?
- E.g. $f(x) = x^2 + 1$
- Curviness does not give a clean geometric formula to find the area
- Pack five rectangles of equal length under the curve, where height of each one extends from x-axis to where midpoint touches the curve
- Rectangle area - length $\times$ width
- The more rectangles the better the approximation
  - Increase/decrease smth toward infinity to approach an actual value



Integral approximation in Python

```
def approximate_integral(a, b, n, f):
  delta_x = (b - a) / n
  total_sum = 0

  for i in range(1, n + 1):
    midpoint = 0.5 * (2 * a + delta_x * (2 * i - 1))
    total_sum += f(midpoint)

  return total_sum * delta_x

def my_function(x):
  return x**2 + 1

area = approximate_integral(a=2, b=3, n=5, f=my_function)

print(area) # prints 7.330000000000002
```

- What happens if we use 1000 rectangles? What about 1000000?
  - We get more precision - 7.333333250000001 and 7.333333333333075

$\hookrightarrow$ Converging to 7.333 (if a rational number its likely 22/3)

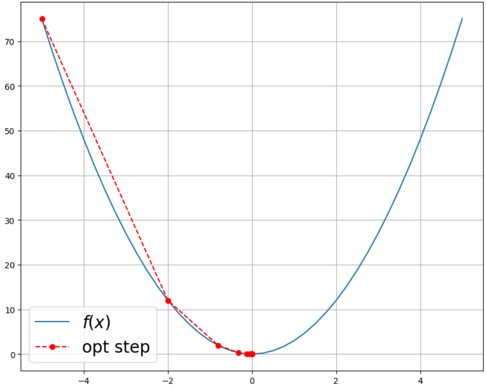
- Use SymPy to perform integration

Gradient Descent

- Used heavily to solve optimization problems

$$\min_{x \in \mathcal{X}} f(x)$$

  where the domain $\mathcal{X}$ is a convex set
- Update rule $x^{t+1} = x^t - \tau \nabla f(x^t)$, for a learning rate $\tau > 0$



Gradient Descent Exercise

Exercise

Given $f(x_1, x_2) = 0.5x_1^2 + x_2^2 + 2x_1 + x_2 + \cos(\sin\sqrt{\pi})$

- Compute the minimum (x_1^*, x_2^*) of (x_1, x_2) analytically
- Perform two steps of gradient descent on $f(x_1, x_2)$ starting from point $(x_1^{(0)}, x_2^{(0)}) = (0, 0)$ with learning rate $\tau = 1$
- Will the gradient descent procedure ever converge to the true minimum (x_1^*, x_2^*)?

Solution

- $\nabla f(x_1, x_2) = \begin{bmatrix} x_1 + 2 \\ 2x_2 + 1 \end{bmatrix} \overset{!}{=} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \implies \begin{bmatrix} x_1^* \\ x_2^* \end{bmatrix} = \begin{bmatrix} -2 \\ -1/2 \end{bmatrix}$
- 1^{st} update $\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \end{bmatrix} - \tau \begin{bmatrix} x_1^{(0)} + 2 \\ 2x_2^{(0)} + 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \tau \begin{bmatrix} 0 + 2 \\ 2 \cdot 0 + 1 \end{bmatrix} = \begin{bmatrix} -2 \\ -1 \end{bmatrix}$

  2^{nd} update $\begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \end{bmatrix} = \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} - \tau \begin{bmatrix} x_1^{(1)} + 2 \\ 2x_2^{(1)} + 1 \end{bmatrix} = \begin{bmatrix} -2 \\ -1 \end{bmatrix} - 1 \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$
- 3^{rd} update $\begin{bmatrix} x_1^{(3)} \\ x_2^{(3)} \end{bmatrix} = \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \end{bmatrix} - \tau \begin{bmatrix} x_1^{(2)} + 2 \\ 2x_2^{(2)} + 1 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \end{bmatrix} - 1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ -1 \end{bmatrix} = \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix}$

  $\hookrightarrow$ Stuck between $x^{(1)}$ and $x^{(2)}$ forever. Decrease learning rate (adaptive step size).

Gradient Descent Exercise

Exercise

Given $f(x_1, x_2) = 0.5x_1^2 + x_2^2 + 2x_1 + x_2 + \cos(\sin\sqrt{\pi})$

- Compute the minimum (x_1^*, x_2^*) of (x_1, x_2) analytically
- Perform two steps of gradient descent on $f(x_1, x_2)$ starting from point $(x_1^{(0)}, x_2^{(0)}) = (0, 0)$ with learning rate $\tau = 1$
- Will the gradient descent procedure ever converge to the true minimum (x_1^*, x_2^*)?

Solution

- $\nabla f(x_1, x_2) = \begin{bmatrix} x_1 + 2 \\ 2x_2 + 1 \end{bmatrix} \overset{!}{=} \begin{bmatrix} 0 \\ 0 \end{bmatrix} \implies \begin{bmatrix} x_1^* \\ x_2^* \end{bmatrix} = \begin{bmatrix} -2 \\ -1/2 \end{bmatrix}$
- 1^{st} update $\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \begin{bmatrix} x_1^{(0)} \\ x_2^{(0)} \end{bmatrix} - \tau \begin{bmatrix} x_1^{(0)} + 2 \\ 2x_2^{(0)} + 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \tau \begin{bmatrix} 0 + 2 \\ 2 \cdot 0 + 1 \end{bmatrix} = \begin{bmatrix} -2 \\ -1 \end{bmatrix}$

  2^{nd} update $\begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \end{bmatrix} = \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} - \tau \begin{bmatrix} x_1^{(1)} + 2 \\ 2x_2^{(1)} + 1 \end{bmatrix} = \begin{bmatrix} -2 \\ -1 \end{bmatrix} - 1 \begin{bmatrix} 0 \\ -1 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \end{bmatrix}$
- 3^{rd} update $\begin{bmatrix} x_1^{(3)} \\ x_2^{(3)} \end{bmatrix} = \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \end{bmatrix} - \tau \begin{bmatrix} x_1^{(2)} + 2 \\ 2x_2^{(2)} + 1 \end{bmatrix} = \begin{bmatrix} -2 \\ 0 \end{bmatrix} - 1 \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} -2 \\ -1 \end{bmatrix} = \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix}$

  $\hookrightarrow$ Stuck between $x^{(1)}$ and $x^{(2)}$ forever. Decrease learning rate (adaptive step size).

Matrix Calculus

- $f : \mathbb{R}^{m \times n} \to \mathbb{R}$
- ***Gradient*** of f (w.r.t. $A \in \mathbb{R}^{m \times n}$) is the matrix of partial derivatives

$$\nabla_A f(A) \in \mathbb{R}^{m \times n} = \begin{bmatrix} \frac{\partial f(A)}{\partial A_{11}} & \frac{\partial f(A)}{\partial A_{12}} & \cdots & \frac{\partial f(A)}{\partial A_{1n}} \\ \frac{\partial f(A)}{\partial A_{21}} & \frac{\partial f(A)}{\partial A_{22}} & \cdots & \frac{\partial f(A)}{\partial A_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(A)}{\partial A_{m1}} & \frac{\partial f(A)}{\partial A_{m2}} & \cdots & \frac{\partial f(A)}{\partial A_{mn}} \end{bmatrix}$$

- In general $(\nabla_A f(A))_{ij} = \frac{\partial f(A)}{\partial A_{ij}}$
- If A is a vector $x \in \mathbb{R}^n$

$$\nabla_x f(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}$$

- Note the size of $\nabla_A f(A)$ is always same as the size of A
- Gradient of a function is *only* defined if the function is real-valued
  - E.g. cannot take the gradient of Ax, $A \in \mathbb{R}^{n \times n}$ wrt x

Matrix Calculus Exercise

Exercise

Suppose $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ and the function $f : \mathbb{R}^{2\times 2} \to \mathbb{R}$ is given by

$$f(A) = \frac{3}{2}A_{11} + 5A_{12}^2 + A_{21}A_{22}$$

Find $\nabla_A f(A)$

Solution

Use $(\nabla_A f(A))_{ij} = \frac{\partial f(A)}{\partial A_{ij}}$ to find

$$\nabla_A f(A) = \begin{bmatrix} \frac{3}{2} & 10A_{12} \\ A_{22} & A_{21} \end{bmatrix}$$

Matrix Calculus Exercise

Exercise

Suppose $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$ and the function $f : \mathbb{R}^{2\times 2} \to \mathbb{R}$ is given by

$$f(A) = \frac{3}{2}A_{11} + 5A_{12}^2 + A_{21}A_{22}$$

Find $\nabla_A f(A)$

Solution

Use $(\nabla_A f(A))_{ij} = \frac{\partial f(A)}{\partial A_{ij}}$ to find

$$\nabla_A f(A) = \begin{bmatrix} \frac{3}{2} & 10A_{12} \\ A_{22} & A_{21} \end{bmatrix}$$

Properties

- $\nabla_x(f(x)+g(x)) = \nabla_x f(x) + \nabla_x g(x)$
- $t \in \mathbb{R}, \quad \nabla_x(tf(x)) = t\nabla_x f(x)$

Working with gradients can be tricky (!)

- $A \in \mathbb{R}^{m\times n}$ matrix of fixed coefficients
- $b \in \mathbb{R}^m$ vector of fixed coefficients
- $f : \mathbb{R}^m \to \mathbb{R}$ defined by $f(z) = z^T z$ such that $\nabla_z f(z) = 2z$

How do we express $\nabla f(Ax)$?

1. Recall $\nabla_z f(z) = 2z$. Interpret $\nabla f(Ax)$ as evaluating the gradient at point Ax

$$\nabla f(Ax) = 2(Ax) = 2Ax \in \mathbb{R}^m$$

2. Interpret $f(Ax)$ as a function of input variables x. If $g(x) = f(Ax)$ then

$$\nabla f(Ax) = \nabla_x g(x) \in \mathbb{R}^n$$

- Make explicit variables which we are differentiating with respect to

1. $\nabla_z f(Ax)$ - Differentiate f wrt its arguments z then substituting Ax
2. $\nabla_x f(Ax)$ - Differentiate composite $g(x) = f(Ax)$ wrt x directly

The Hessian

- $f : \mathbb{R}^n \to \mathbb{R}$
- ***Hessian*** matrix wrt x, $\nabla_x^2 f(x)$ or H, is $n \times n$ matrix of partial derivatives

$$\nabla_x^2 f(x) \in \mathbb{R}^{n\times n} = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1^2} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial x_2^2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_n^2} \end{bmatrix}$$

- In general $\nabla_x^2 f(x) \in \mathbb{R}^{n\times n}$ with

$$(\nabla_x^2 f(x))_{ij} = \frac{\partial^2 f(x)}{\partial x_i \partial x_j}$$

- Note Hessian is symmetric since

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} = \frac{\partial^2 f(x)}{\partial x_j \partial x_i}$$

- Hessian defined only when $f(x)$ is real-valued

The Hessian

- Gradient is the analogue of the first derivative for functions of vectors
- Hessian is the analogue of the second derivative

Caveats to keep in mind

1. For real-valued functions of one variable $f : \mathbb{R} \to \mathbb{R}$, the second derivative is the derivative of the first derivative

$$\frac{\partial^2 f(x)}{\partial x^2} = \frac{\partial}{\partial x}\frac{\partial}{\partial x} f(x)$$

2. For functions of a vector, the gradient of the function is a vector, and we cannot take the gradient of a vector

$$\nabla_x \nabla_x f(x) = \nabla_x \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix} \qquad (!)$$

The Hessian

- Hessian is not the gradient of the gradient
- *Almost true* in the following sense
  - Look at i^{th} entry of the gradient $(\nabla_x f(x))_i = \partial f(x)/\partial x_i$
  - Take the gradient wrt x

$$\nabla_x \frac{\partial f(x)}{\partial x_i} = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_i \partial x_1} \\ \frac{\partial^2 f(x)}{\partial x_i \partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_i \partial x_n} \end{bmatrix}$$

  - which is i^{th} column (or row) of Hessian. Hence

$$\nabla_x^2 f(x) = \begin{bmatrix} \nabla_x(\nabla_x f(x))_1 & \nabla_x(\nabla_x f(x))_2 & \dots & \nabla_x(\nabla_x f(x))_n \end{bmatrix}$$

  - $\nabla_x^2 f(x) = \nabla_x(\nabla_x f(x))^T$

Gradients of Linear Functions

- $x \in \mathbb{R}^n, \quad f(x) = b^T x$ for known $b \in \mathbb{R}^n$

$$
\begin{aligned}
f(x) &= \sum_{i=1}^{n} b_i x_i \\
\frac{\partial f(x)}{\partial x_k} &= \frac{\partial}{\partial x_k} \sum_{i=1}^{n} b_i x_i = b_k
\end{aligned}
$$

  - $\nabla_x b^T x = b$
  - $\partial / (\partial x) ax = a \quad$ (single variable calculus)

Gradients of Quadratic Functions

- Quadratic function $f(x) = x^T Ax$ for $A \in \mathbb{S}^n$

$$
\begin{aligned}
f(x) &= \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}x_ix_j \\
\frac{\partial f(x)}{\partial x_k} &= \frac{\partial}{\partial x_k}\sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}x_ix_j \\
&= \frac{\partial}{\partial x_k}\left[\sum_{i\neq k}\sum_{j\neq k} A_{ij}x_ix_j + \sum_{i\neq k} A_{ik}x_ix_k + \sum_{j\neq k} A_{kj}x_kx_j + A_{kk}x_k^2\right] \\
&= \sum_{i\neq k} A_{ij}x_i + \sum_{j\neq k} A_{kj}x_j + 2A_{kk}x_k \\
&= \sum_{i=1}^{n} A_{ik}x_i + \sum_{j=1}^{n} A_{kj}x_j \\
&= 2\sum_{i=1}^{n} A_{ki}x_i
\end{aligned}
$$

- k^{th} entry of $\nabla_x f(x)$ is inner product of k^{th} row of A and x
- $\nabla_x x^T Ax = 2Ax$
- $\partial/(\partial x)ax^2 = 2ax$ (single variable calculus)

Hessians of Quadratic Functions

- Quadratic function $f(x) = x^T A x$ for $A \in \mathbb{S}^n$

$$
\begin{aligned}
\frac{\partial^2 f(x)}{\partial x_k \partial x_l} &= \frac{\partial}{\partial x_k}\left[\frac{\partial f(x)}{\partial x_l}\right] \\
&= \frac{\partial}{\partial x_k}\left[2\sum_{i=1}^n A_{li} x_i\right] \\
&= 2A_{lk} \\
&= 2A_{kl}
\end{aligned}
$$

  - $\nabla_x^2 x^T A x = 2A$
  - $\partial^2/(\partial x^2) a x^2 = 2a \quad$ (single variable calculus)

Recap

- $\nabla_x b^T x = b$
- $\nabla_x x^T A x = 2Ax \quad$ (if A symmetric)
- $\nabla_x^2 x^T A x = 2A \quad$ (if A symmetric)

Least Squares

- $A \in \mathbb{R}^{m \times n}$ of full rank
- $b \in \mathbb{R}^m$ such that $b \notin \mathcal{R}(A)$
  $\hookrightarrow$ Not able to find a vector $x \in \mathbb{R}^n$ such that $Ax = b$
  $\hookrightarrow$ Find a vector x such that Ax is as close as possible to b, measured by Euclidean norm $\|Ax - b\|_2^2$

$$\begin{aligned} \|Ax - b\|_2^2 &= (Ax - b)^T(Ax - b) \\ &= x^T A^T A x - 2b^T A x + b^T b \end{aligned}$$

- Take gradient wrt x

$$\begin{aligned} \nabla_x (x^T A^T A x - 2b^T A x + b^T b) &= \nabla_x x^T A^T A x - \nabla_x 2b^T A x + \nabla_x b^T b \\ &= 2A^T A x - 2A^T b \end{aligned}$$

- Set to zero and solve for x

$$x = (A^T A)^{-1} A^T b$$

Eigenvalues as Optimization

- Equality constrained optimization problem

$$\max_{x \in \mathbb{R}^n} x^T A x \quad \text{subject to } \|x\|_2^2 = 1$$

- ***Lagrangian***

$$\mathcal{L}(x, \lambda) = x^T A x - \lambda x^T x$$

  - λ - Lagrange multiplier associated with equality constraint

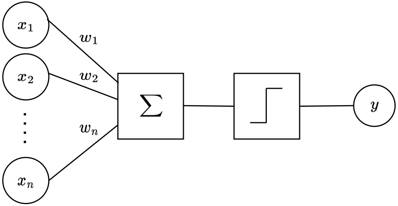
- For x^* to be an optimal point, the gradient of the Lagrangian has to be zero at x^*

$$\begin{aligned} \nabla_x \mathcal{L}(x, \lambda) &= \nabla_x (x^T A x - \lambda x^T x) \\ &= 2A^T x - 2\lambda x \\ &= 0 \end{aligned}$$

  - Linear equation $Ax = \lambda x$
  - The only points that can possible maximize (or minimize) $x^T A x$ assuming $x^T x = 1$ are eigenvectors of A

The Perceptron

Structure:



- Weighted sum of input features

$$\begin{aligned} z &= \sum_{i=1}^{n} w_i x_i + b \\ &= \mathbf{w}^T \mathbf{x} + b \end{aligned}$$

- Followed by the ***sign*** function

$$y = \text{sign}(z)$$

Learning task: Given input data

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(m)} \in \mathbb{R}^n$$

of corresponding labels $t^{(1)}, t^{(2)}, \ldots, t^{(m)} \in \{-1, 1\}$

- Goal is to learn a collection of parameters $(\mathbf{w}, b)$ such that

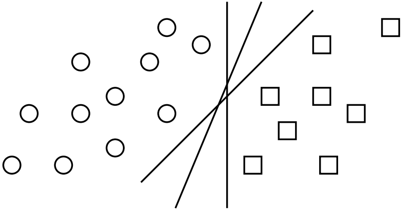
$$\min_{\mathbf{w}, b} \sum_{j=1}^{m} \mathcal{L}(t^j, \mathbf{w}^T \mathbf{x}^j + b)$$

- $\mathcal{L}(\mathbf{w}, b)$ denotes the error function

The Perceptron

- Predictions of the perceptron for each datapoint

$$
\begin{aligned}
z^{(j)} &= \mathbf{w}^T \mathbf{x}^{(j)} + b \\
y^{(j)} &= \text{sign}(z^{(j)})
\end{aligned}
$$



Question:

Can all the points be correctly classified

$$\exists(\mathbf{w}, b) : y^{(j)} = t^{(j)}, \forall_{j=1}^{m}?$$

The Perceptron Algorithm

Perceptron Algorithm

- Initialize $\mathbf{w} = \mathbf{0}$ and $b = 0$
- Repeat for $j = 1, ..., m$
  - If $\mathbf{x}^{(j)}$ is correctly classified ($y^{(j)} = t^{(j)}$), continue
  - If $\mathbf{x}^{(j)}$ is wrongly classified ($y^{(j)} \neq t^{(j)}$), update

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} + \eta \cdot \mathbf{x}^{(j)} t^{(j)} \\ b &\leftarrow b + \eta \cdot t^{(j)} \end{aligned}$$

  for some learning rate η
- Until all examples are classified correctly

Optimization View of Perceptron

Proposition

The perceptron is equivalent to the gradient descent of the so-called ***Hinge Loss***

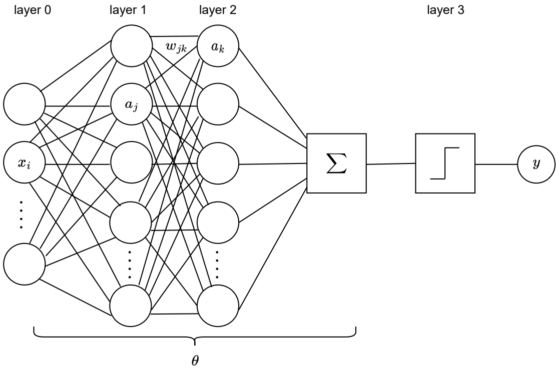
$$\mathcal{L}(\mathbf{w}, b) = \frac{1}{m} \sum_{j=1}^{m} \underbrace{\max(0, -z^{(j)} t^{(j)})}_{\mathcal{L}_j(\mathbf{w}, b)}$$

Proof.

$$\begin{aligned}
\mathbf{w} - \eta \frac{\partial \mathcal{L}_j}{\partial \mathbf{w}} &= \mathbf{w} - \eta \cdot 1_{-z^{(j)} t^{(j)} > 0} \cdot \left(-\frac{\partial z^{(j)}}{\partial \mathbf{w}} t^{(j)} \right) \\
&= \mathbf{w} - \eta \cdot 1_{y^{(j)} \neq t^{(j)}} \cdot \left(-\frac{\partial z^{(j)}}{\partial \mathbf{w}} t^{(j)} \right) \\
&= \mathbf{w} + \eta \cdot 1_{y^{(j)} \neq t^{(j)}} \cdot \mathbf{x}^{(j)} t^{(j)}
\end{aligned}$$

- Proceed similarly for the parameter b

From Perceptron to Deep Neural Networks



Idea:

Stack multiple perceptrons together to generalize the formulation where z is the output of a multilayer neural network with parameters θ

$\hookrightarrow$ Updated error function $\mathcal{L}(\theta)$

Numerical Differentiation

Question:

How hard is it to compute the gradient of the error function w.r.t. the model parameters

$$\theta = \theta - \eta \frac{\partial \mathcal{L}}{\partial \theta} \ ?$$

Idea:

Use the definition of the derivative

$$\forall_t : \frac{\partial \mathcal{L}}{\partial \theta_t} = \lim_{\varepsilon \to 0} \frac{\mathcal{L}(\theta + \varepsilon \cdot \delta_t) - \mathcal{L}(\theta)}{\varepsilon}$$

- δ_t denotes an indicator vector for the parameter t

Properties:

- Applicable to any error function $\mathcal{L}$
- Re-evaluate the function as many times as there are parameters ($\hookrightarrow$ slow for a large number of parameters)
- Neural networks typically have between 10^3 and 10^9 parameters ($\hookrightarrow$ numerical differentiation unfeasible)
- Need to use high-precision due to small ε and numerator

Non-convex error function

Problems:

- $\mathcal{L}(\theta)$ is non-convex and non-linear
- For complex functions, the computation of $\nabla_\theta \mathcal{L}$ is tricky to be done by hand

Question:

Can we do this automatically?

- A general rule to find the weights θ was not discovered until 1974 (Paul Werbos) / 1985 (LeCun) / 1986 (Rumelhart et al.)

Idea:

Need to compute the gradient $\partial \mathcal{L} / \partial w_{jk}$
$\hookrightarrow$ Compute the error at the output, and propagate that back to the neurons in the earlier layers
$\hookrightarrow$ Compute the gradient

Recall the Chain Rule

- Assume some parameter of interest θ_q and the output of the network z are linked through a sequence of functions

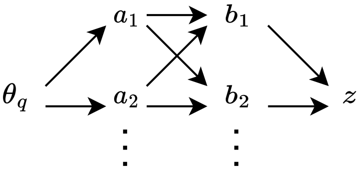
$$\boldsymbol{\theta_q} \longrightarrow \boldsymbol{a} \longrightarrow \boldsymbol{b} \longrightarrow \boldsymbol{z}$$

- Applying the chain rule for derivatives, the derivative w.r.t. the parameter of interest is the product of local derivatives along the path connecting θ_q to z

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

The Multivariate Chain Rule

- The parameter of interest may be linked to the output of the network via multiple paths, formed by all neurons in layers between θ_q and z



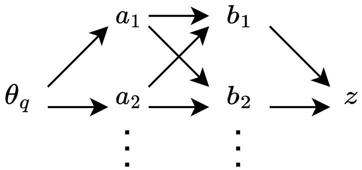
- Multivariate scenario $\Rightarrow$ the chain rule enumerates all the paths between θ_q and z

$$\frac{\partial z}{\partial \theta_q} = \sum_i \sum_j \frac{\partial a_i}{\partial \theta_q} \frac{\partial b_j}{\partial a_j} \frac{\partial z}{\partial b_j}$$

  where $\sum_i$ and $\sum_j$ run over all indices of the nodes in the corresponding layers

- Nested sum - complexity grows exponentially with the number of layers

Factor Structure in the Multivariate Chain Rule

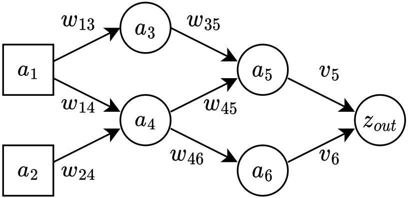


- Re-write the computation - perform the summing operation incrementally
- Re-use intermediate computation for different paths and parameters for which we would like to compute the gradient

$$\frac{\partial z}{\partial \theta_q} = \sum_i \frac{\partial a_i}{\partial \theta_q} \underbrace{\sum_j \frac{\partial b_j}{\partial a_j} \underbrace{\frac{\partial z}{\partial b_j}}_{\delta_j}}_{\delta_i}$$

- The resulting gradient computation w.r.t. all parameters in the network is linear with the size of the network ($\Rightarrow$ fast!)

Backpropagation through Example

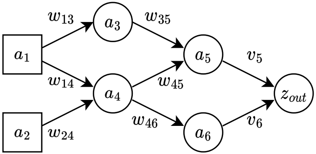


Forward pass:

$$\begin{aligned} a_1 &= x_1 \\ a_2 &= x_2 \\ a_3 &= \tanh(z_3) \\ a_4 &= \tanh(z_4) \\ a_5 &= \tanh(z_5) \\ a_6 &= \tanh(z_6) \end{aligned}$$

$$\begin{aligned} z_3 &= a_1 w_{13} \\ z_4 &= a_1 w_{14} + a_2 w_{24} \\ z_5 &= a_3 w_{35} + a_4 w_{45} \\ z_6 &= a_4 w_{46} \\ z_{out} &= a_5 v_5 + a_6 v_6 \\ \mathcal{L} &= \max(0, -z_{out} \cdot t) \end{aligned}$$

Backpropagation through Example



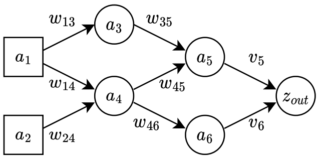
$$
\begin{aligned}
z_3 &= a_1 w_{13} \\
z_4 &= a_1 w_{14} + a_2 w_{24} \\
z_5 &= a_3 w_{35} + a_4 w_{45} \\
z_6 &= a_4 w_{46} \\
z_{out} &= a_5 v_5 + a_6 v_6 \\
\mathcal{L} &= \max(0, -z_{out} \cdot t)
\end{aligned}
$$

$$
\begin{aligned}
a_1 &= x_1 \\
a_2 &= x_2 \\
a_3 &= \tanh(z_3) \\
a_4 &= \tanh(z_4) \\
a_5 &= \tanh(z_5) \\
a_6 &= \tanh(z_6)
\end{aligned}
$$

Backward pass:

$$
\begin{aligned}
\delta_{out} &= \frac{\partial \mathcal{L}}{\partial z_{out}} = 1_{\{-z_{out} \cdot t > 0\}} \cdot (-t) \\
& \frac{\partial \mathcal{L}}{\partial v_6} = \frac{\partial z_{out}}{\partial v_6} \frac{\partial \mathcal{L}}{\partial z_{out}} = a_6 \cdot \delta_{out} \\
& \frac{\partial \mathcal{L}}{\partial v_5} = \frac{\partial z_{out}}{\partial v_5} \frac{\partial \mathcal{L}}{\partial z_{out}} = a_5 \cdot \delta_{out}
\end{aligned}
$$

Backpropagation through Example



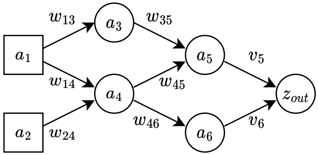
$$
\begin{aligned}
z_3 &= a_1 w_{13} \\
z_4 &= a_1 w_{14} + a_2 w_{24} \\
z_5 &= a_3 w_{35} + a_4 w_{45} \\
z_6 &= a_4 w_{46} \\
z_{out} &= a_5 v_5 + a_6 v_6 \\
\mathcal{L} &= \max(0, -z_{out} \cdot t)
\end{aligned}
$$

$$
\begin{aligned}
a_1 &= x_1 \\
a_2 &= x_2 \\
a_3 &= \tanh(z_3) \\
a_4 &= \tanh(z_4) \\
a_5 &= \tanh(z_5) \\
a_6 &= \tanh(z_6)
\end{aligned}
$$

Backward pass:

$$
\begin{aligned}
\delta_{out} &= \frac{\partial \mathcal{L}}{\partial z_{out}} = 1_{\{-z_{out} \cdot t > 0\}} \cdot (-t) \\
\delta_6 &= \frac{\partial \mathcal{L}}{\partial a_6} = \frac{\partial z_{out}}{\partial a_6} \frac{\partial \mathcal{L}}{\partial z_{out}} = v_6 \cdot \delta_{out} \\
\delta_5 &= \frac{\partial \mathcal{L}}{\partial a_5} = \frac{\partial z_{out}}{\partial a_5} \frac{\partial \mathcal{L}}{\partial z_{out}} = v_5 \cdot \delta_{out}
\end{aligned}
$$

Backpropagation through Example



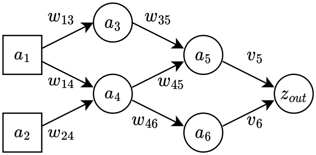
$$
\begin{aligned}
z_3 &= a_1 w_{13} \\
z_4 &= a_1 w_{14} + a_2 w_{24} \\
z_5 &= a_3 w_{35} + a_4 w_{45} \\
z_6 &= a_4 w_{46} \\
z_{out} &= a_5 v_5 + a_6 v_6 \\
\mathcal{L} &= \max(0, -z_{out} \cdot t)
\end{aligned}
$$

$$
\begin{aligned}
a_1 &= x_1 \\
a_2 &= x_2 \\
a_3 &= \tanh(z_3) \\
a_4 &= \tanh(z_4) \\
a_5 &= \tanh(z_5) \\
a_6 &= \tanh(z_6)
\end{aligned}
$$

Backward pass:

$$
\begin{aligned}
\delta_6 &= \frac{\partial \mathcal{L}}{\partial a_6} = \frac{\partial z_{out}}{\partial a_6}\frac{\partial \mathcal{L}}{\partial z_{out}} = v_6 \cdot \delta_{out} \\
\delta_5 &= \frac{\partial \mathcal{L}}{\partial a_5} = \frac{\partial z_{out}}{\partial a_5}\frac{\partial \mathcal{L}}{\partial z_{out}} = v_5 \cdot \delta_{out} \\
&\frac{\partial \mathcal{L}}{\partial w_{46}} = \frac{\partial z_6}{\partial w_{46}}\frac{\partial a_6}{\partial z_6}\frac{\partial \mathcal{L}}{\partial a_6} = a_4 \cdot \tanh'(z_6) \cdot \delta_6 \\
&\frac{\partial \mathcal{L}}{\partial w_{45}} = \frac{\partial z_5}{\partial w_{45}}\frac{\partial a_5}{\partial z_5}\frac{\partial \mathcal{L}}{\partial a_5} = a_4 \cdot \tanh'(z_5) \cdot \delta_5 \\
&\frac{\partial \mathcal{L}}{\partial w_{35}} = \frac{\partial z_5}{\partial w_{35}}\frac{\partial a_5}{\partial z_5}\frac{\partial \mathcal{L}}{\partial a_3} = a_5 \cdot \tanh'(z_5) \cdot \delta_5
\end{aligned}
$$

Backpropagation through Example



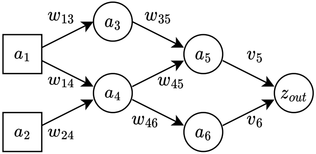
$$
\begin{aligned}
z_3 &= a_1 w_{13} \\
z_4 &= a_1 w_{14} + a_2 w_{24} \\
z_5 &= a_3 w_{35} + a_4 w_{45} \\
z_6 &= a_4 w_{46} \\
z_{out} &= a_5 v_5 + a_6 v_6 \\
\mathcal{L} &= \max(0, -z_{out} \cdot t)
\end{aligned}
$$

$$
\begin{aligned}
a_1 &= x_1 \\
a_2 &= x_2 \\
a_3 &= \tanh(z_3) \\
a_4 &= \tanh(z_4) \\
a_5 &= \tanh(z_5) \\
a_6 &= \tanh(z_6)
\end{aligned}
$$

Backward pass:

$$
\begin{aligned}
\delta_6 &= \frac{\partial \mathcal{L}}{\partial a_6} = \frac{\partial z_{out}}{\partial a_6} \frac{\partial \mathcal{L}}{\partial z_{out}} = v_6 \cdot \delta_{out} \\
\delta_5 &= \frac{\partial \mathcal{L}}{\partial a_5} = \frac{\partial z_{out}}{\partial a_5} \frac{\partial \mathcal{L}}{\partial z_{out}} = v_5 \cdot \delta_{out} \\
\delta_4 &= \frac{\partial \mathcal{L}}{\partial a_4} = \frac{\partial z_6}{\partial a_4} \frac{\partial a_6}{\partial z_6} \frac{\partial \mathcal{L}}{\partial a_6} + \frac{\partial z_5}{\partial a_4} \frac{\partial a_5}{\partial z_5} \frac{\partial \mathcal{L}}{\partial a_5} = w_{46} \cdot \tanh'(z_6) \cdot \delta_6 + w_{45} \cdot \tanh'(z_5) \cdot \delta_5 \\
\delta_3 &= \frac{\partial \mathcal{L}}{\partial a_3} = \frac{\partial z_5}{\partial a_3} \frac{\partial a_5}{\partial z_5} \frac{\partial \mathcal{L}}{\partial a_5} = w_{35} \cdot \tanh'(z_5) \cdot \delta_5
\end{aligned}
$$

Backpropagation through Example



$$\begin{aligned} z_3 &= a_1 w_{13} \\ z_4 &= a_1 w_{14} + a_2 w_{24} \\ z_5 &= a_3 w_{35} + a_4 w_{45} \\ z_6 &= a_4 w_{46} \\ z_{out} &= a_5 v_5 + a_6 v_6 \\ \mathcal{L} &= \max(0, -z_{out} \cdot t) \end{aligned}$$

$$\begin{aligned} a_1 &= x_1 \\ a_2 &= x_2 \\ a_3 &= \tanh(z_3) \\ a_4 &= \tanh(z_4) \\ a_5 &= \tanh(z_5) \\ a_6 &= \tanh(z_6) \end{aligned}$$

Backward pass:

$$\begin{aligned} \delta_4 &= \frac{\partial \mathcal{L}}{\partial a_4} = \frac{\partial z_6}{\partial a_4}\frac{\partial a_6}{\partial z_6}\frac{\partial \mathcal{L}}{\partial a_6} + \frac{\partial z_5}{\partial a_4}\frac{\partial a_5}{\partial z_5}\frac{\partial \mathcal{L}}{\partial a_5} = w_{46} \cdot \tanh'(z_6) \cdot \delta_6 + w_{45} \cdot \tanh'(z_5) \cdot \delta_5 \\ \delta_3 &= \frac{\partial \mathcal{L}}{\partial a_3} = \frac{\partial z_5}{\partial a_3}\frac{\partial a_5}{\partial z_5}\frac{\partial \mathcal{L}}{\partial a_5} = w_{35} \cdot \tanh'(z_5) \cdot \delta_5 \\ & \frac{\partial \mathcal{L}}{\partial w_{24}} = \frac{\partial z_4}{\partial w_{24}}\frac{\partial a_4}{\partial z_4}\frac{\partial \mathcal{L}}{\partial a_4} = a_2 \cdot \tanh'(z_4) \cdot \delta_4 \\ & \frac{\partial \mathcal{L}}{\partial w_{14}} = \frac{\partial z_4}{\partial w_{14}}\frac{\partial a_4}{\partial z_4}\frac{\partial \mathcal{L}}{\partial a_4} = a_1 \cdot \tanh'(z_4) \cdot \delta_4 \\ & \frac{\partial \mathcal{L}}{\partial w_{13}} = \frac{\partial z_3}{\partial w_{13}}\frac{\partial a_3}{\partial z_3}\frac{\partial \mathcal{L}}{\partial a_3} = a_1 \cdot \tanh'(z_3) \cdot \delta_3 \end{aligned}$$

Formalization for a Standard Neural Network

- Propagate the gradient of the error from layer to layer using the chain rule

$$\underbrace{\frac{\partial \mathcal{L}}{\partial a_j}}_{\delta_j} = \sum_k \underbrace{\frac{\partial a_k}{\partial a_j}}_{w_{jk} g'(z_k)} \cdot \underbrace{\frac{\partial \mathcal{L}}{\partial a_k}}_{\delta_k}$$

- Extract gradients w.r.t. parameters at each layer as

$$\frac{\partial \mathcal{L}}{\partial w_{jk}} = \sum_k \underbrace{\frac{\partial a_k}{\partial w_{jk}}}_{a_j g'(z_k)} \cdot \underbrace{\frac{\partial \mathcal{L}}{\partial a_k}}_{\delta_k}$$

- Re-write equations as matrix-vector products

$$\begin{aligned} \delta^{(l-1)} &= W^{(l-1,l)} \cdot (g'(\mathbf{z}^{(l)}) \odot \delta^{(l)}) \\ \frac{\partial \mathcal{L}}{\partial W^{(l-1,l)}} &= \mathbf{a} \cdot (g'(\mathbf{z}^{(l)}) \odot \delta^{(l)})^T \end{aligned}$$

Vanishing gradient

- In general

$$\partial\mathcal{L}/\partial W^{(l-1,l)} \gg \partial\mathcal{L}/\partial W^{(l-2,l-1)}$$

$\Rightarrow$ the more left you get in the network, the more the gradient vanishes

- tanh has gradients in the range $(0, 1]$
  $\Rightarrow$ in an $n-$layer network the gradient decreases exponentially with n

Ways to circumvent vanishing gradients

- Use many labeled data (e.g., well possible for images)
- Train ”longer” (possible with GPUs)
- Better weight initialization (e.g., Xavier/Glorot)
- Regularize with “dropout“
- Other activation functions: ReLU

Choice of Nonlinear Activation Function

Choose the nonlinear function such that

- Its gradient is defined (almost) everywhere
- A significant portion of the input domain has a non-zero gradient
- Its gradient is informative, i.e., indicate decrease/increase of the activation function

Commonly used activation functions:

- ***Sigmoid*** $g(z) = \exp(z)/(1 + \exp(z))$
- ***tanh*** $g(z) = \tanh(z)$
- ***ReLU*** $g(z) = \max(0, z)$

Problematic activation functions:

- $g(z) = \max(0, z - 100)$
- $g(z) = 1_{z>0}$
- $g(z) = \sin(100 \cdot z)$

Automatic Differentiation

- Automatically generate backpropagation equations from the forward equations
- Automatic differentiation widely available in deep learning libraries (PyTorch, Tensorflow, JAX, etc.)

Consequences:

- No need to do backpropagation, just program the forward pass
  $\hookrightarrow$ backward pass comes for free
- Motivated the development of neural networks that are way more complex, and with much more heterogeneous structures (e.g. ResNet, Yolo, transformers, etc.)
- In few cases, it is still useful to express the gradient analytically (e.g. to analyze theoretically the stability of a gradient descent procedure)

Training Neural Networks

Basic gradient descent algorithm

- Initialize θ at random
- Repeat for T steps
  - Compute the forward pass
  - Use backpropagation to extract $\partial \mathcal{L} / \partial \theta$
  - Perform a gradient step

$$\theta = \theta - \gamma \frac{\partial \mathcal{L}}{\partial \theta}$$

  for some learning rate γ

Summary

- Gradient descent to minimize the error of a classifier (e.g. Perceptron, neural network + backpropagation)
- Error backpropagation provides a computationally efficient way of computing the gradient compared to the formula for numerical differentiation
- Error backpropagation is a direct application of the multivariate chain rule, where the different terms can be factored due to the structure of the neural network graph
- Use certain techniques to circumvent vanishing gradients
- No need to program error backpropagation manually, use automatic differentiation techniques instead

THANK YOU!

Slides available at:

`www.shpresimsadiku.com`

Check related information on Twitter at:

`@shpresimsadiku`