Calculus for Machine Learning and Data Science

Week1: Derivatives and Optimization:

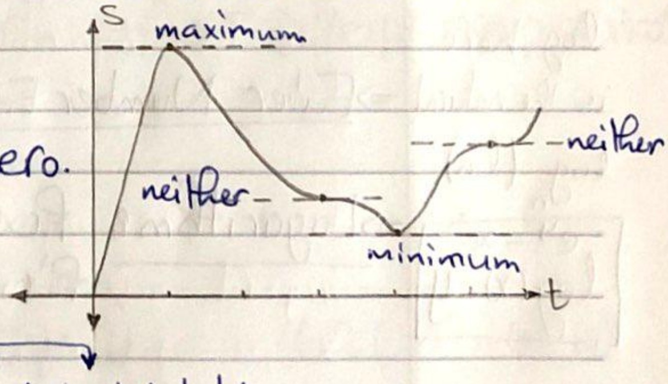
importance — Calculus in machine learning depends largely on derivatives for optimizations (minimizing or maximizing functions).

Definition — Derivative is the instantaneous change of function. This is expressed as the value of the slope of the tangent at the meant point.

The maximum or the minimum of a specific function are located where the function has tangent of slope equals zero (derivative equals zero).

The slope of the tangent can be zero, but the value doesn't have to be a local maximum or minimum.

horizontal line has slope zero.



Notation

Lagrange's Notation

$f'(x)$

Leibniz's Notation

$\frac{dy}{dx} = \frac{d}{dx} f(x)$

⇒ Line: $f(x) = ax + b$ $\quad f'(x) = a$

$f(x) = \text{constant}$ $\quad f'(x) = \text{zero}$

⇒ quadratic: $f(x) = ax^n + b$ $\quad f'(x) = an\,x^{n-1}$

$f(x) = \frac{1}{x} = x^{-1}$ $\quad f'(x) = (-1)(x^{-2}) = -\frac{1}{x^2}$

definition # The inverse function undoes what the original function does:

→ if $g(x)$ is the inverse of $f(x)$, it is denoted as:

$$g(x) = f^{-1}(x) \neq \frac{1}{f(x)}$$

$$g(f(x)) = x$$

example: $f(x) = x^2$ for $x \geq 0$

$g(x) = \sqrt{x}$ for $x \geq 0$

⇒ Inverse: if $g(x) = f^{-1}(x)$, then $g'(x) = \frac{1}{f'(x)}$

knowing that the coordinates are either → $(x, f(x))$ or → $(y, g(y))$ or → $(g(f(x)), f(x))$ or → $(f(x), g(f(x)))$

, so the points have to be set according to the plotting used.

⇒ Trignometric: $f(x) = \sin(x)$ $\quad f'(x) = \cos(x)$

$f(x) = \cos(x)$ $\quad f'(x) = -\sin(x)$

definition # The exponential $(e) = \left(1 + \frac{1}{\infty}\right)^{\infty} = 2.71828182...$ (euler number), and $f(x) = e^x$, then $f^{-1}(y) = \log(y)$

log here is the natural $\log_n$ (ln)

$\boxed{\begin{array}{c} e^y = x \\ \log(x) = y \end{array}}$

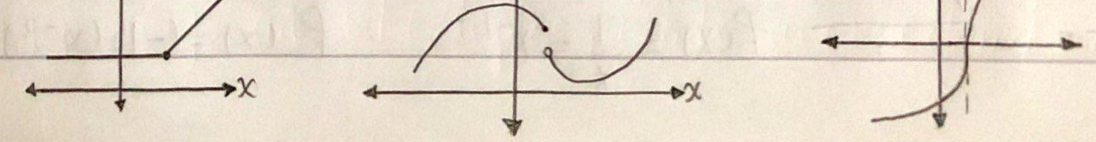
⇒ Euler Number Exponential: $f(x) = e^x$ $\quad f'(x) = e^x$

⇒ Logarithm: $f(x) = e^x$ $\quad f^{-1}(y) = \log(y)$

$f'(x) = e^x$ $\quad \frac{d}{dy} f^{-1}(y) = \frac{1}{f'(f^{-1}(y))}$

$$\frac{d}{dy} \log(y) = \frac{1}{e^{\log(y)}} = \frac{1}{y}$$

The function can be non-differentiable at a corner or a cusp, jump discontinuity or has a vertical horizontal.



Multiplication by a scalar $\Rightarrow$ if $f(x) = \underline{c}\, g(x)$, then $f'(x) = 4\, g'(x)$

sum rule $\Rightarrow$ if $f(x) = g(x) + h(x)$, then $f'(x) = g'(x) + h'(x)$

product rule $\Rightarrow$ if $f(x) = g(x) \cdot h(x)$, then $f'(x) = g'(x) \cdot h(x) + g(x) \cdot h'(x)$

chain rule $\Rightarrow$ if $f(x) = g(h(x))$, then $f'(x) = g'(h(x)) \cdot h'(x)$

$$\frac{d}{dx} f(x) = \frac{dg(x)}{dh} \cdot \frac{dh(x)}{dx} = \frac{dg(x)}{dx}$$

The rule is called a chain, because it can be chained to another function, and the process continues.

<u>examples</u> $f(x) = j(i(h(g(x))))$, then $\frac{df}{dx} = \frac{dj}{di} \cdot \frac{di}{dh} \cdot \frac{dh}{dg} \cdot \frac{dg}{dx}$

$f'(x) = j'(i(h(g(x)))) \cdot i'(h(g(x))) \cdot h'(g(x)) \cdot g'(x)$

SymPy is a python library for symbolic computations. An expression can be defined as follows:

```
from sympy import *
x, y = symbols ('x y')
expr = 2* x**2 - x * y                    // expr = 2x^2 - xy
expr_manip = x * (expr + x*y + x**3)      // expr_manip = x(x^3+2x^2)
expand (expr_manip)
  → x^4 + 2x^3
factor (expr_manip)
  → x^3 (x+2)
```

evaluate function

```
expr.evalf(subs = {x:-1, y:2})            // 2(3)^2 - (-1)(2)
  → 4.0
```

#

```
import numpy as np
x_array = np.array ([1,2,3])
f_symb = x**2
try:
    f_symb(x_array)
```

sympy functions can't operate

on numpy arrays[objects]

```
except TypeError as err:
        print(err)
  → 'Pow' object is not callable
```

make the function numpy-friendly

```
from sympy.utilities.lambdify import lambdify
f_symb_numpy = lambdify(x, f_symb, 'numpy')
f_symb_numpy(x_array)
  → [1 4 9]
```

differentiate f_symb with respect to x

```
# diff (f_symb, x)
  → 2x
```

⇒ Autograd and JAX are the most commonly used frameworks to build neural networks. Their functionality depends on automatic differentiation which is breaking down the function into common basic fuctions (sin, cos, log, ...etc.) and construct computational graph from them, then chain rule is used to differentiate any node on the graph.

⇒ jax.numpy replaces numpy when jax is used, and can be imported as jnp or np, but jnp sometimes uses different approach:

```
from jax import grad, vmap
import jax.numpy as jnp
x_array = np.array([1,2,3])
x_array_jnp = jnp.array([1,2,3])
x_array_jnp[0]
  → 1
x_array[2] = 4                      // [1, 2, 4]
y_array_jnp = x_array_jnp.at[2].set(4)
```

jnp arrays are like tuples; can be accessed not modified

Most of jax functions work on jnp and np arrays.

```
print (jnp.log (x_array))
→[0.        0.6931472 , 1.0986123]
print (jnp.log (x_array_jnp))
→[0.        0.6931472   1.0986123]
```

To find the derivative of a function at value of x:

f= x**2

```
grad (f)(3.0)
→ 6.0
```

always float only

, and we can use vmap to solve the problem of using integers and float:

```
vmap (grad (f))(x_array_jnp)
→ [2.  4.  6.]
```

Optimization

Derivatives are used to get the maxima and minima:

→ local maxima have +ve slope before & -ve slope after.

→ local minima have -ve slope before & +ve slope after.

, Then the local minima and maxima are then compared to get the absolute (global) maximum or minimum.

⇒ The Square Loss:

Minimize $(x-a_1)^2+(x-a_2)^2+\cdots+(x-a_n)^2$

solution $x=\dfrac{a_1+a_2+\cdots+a_n}{n}$

⇒ The Log loss:

If we have a large equation of probabilities, then we take the log of the equation, and use the logarithm properties:

→ $\log(ab) = \log a + \log b$
→ $\log(x^a) = a \cdot \log(x)$
→ $\log\left(\frac{a}{b}\right) = \log(a) - \log(b)$
→ $\frac{d}{dx} \log(x) = \frac{1}{x}$
→ The logarithm of very small numbers is very large negative number.

Logarithm is used to simplify equations; instead of :→ product rule, use sum rule.
→ dealing with very tiny probabilities, we deal with large negative numbers.

If $\log(g(p)) = G(p)$, then we use $-G(p)$, because $0 \leq$ probability $(p) \leq 1$, so its log always ≤ 0.