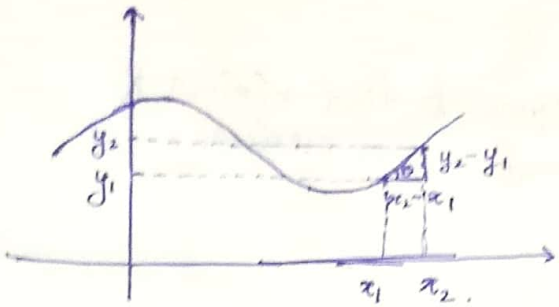
Solving Optimization Problems

Differentiation



$\frac{dy}{dx}$ — diff^n of y wrt x.

$\frac{dy}{dx}$ — intuitively means rate of change of y wrt x.

$$\frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta y}{\Delta x} = \text{Tan}\,\theta.$$

$$\frac{dy}{dx} = \lim_{\Delta x \to 0} \frac{\Delta y}{\Delta x} = \text{Tan}\,\theta.$$

Geometrically, $\left.\frac{dy}{dx}\right|_{x=x_1}$ = slope of tangent to $f(x)$ at $x = x_1$.

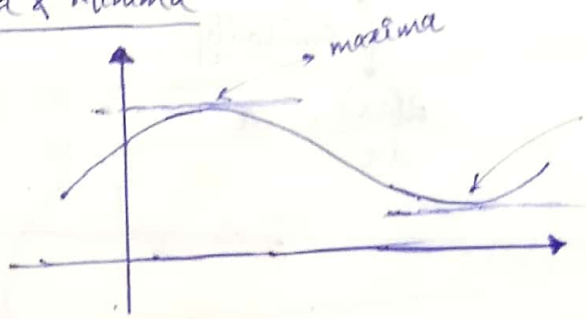
** $y = f(x)$

"All of deep-learning is just chain rule" — Jeff Dean.

Online derivative calculators

↳ derivative-calculator.net.
symbolab.com — step by step breakdown.
wolframalpha.com - plots.

Maxima & Minima



At maxima & minima, slope = 0.

A function may/may'nt have maxima & minima. or even may have more than one maxima & minima.

Ex $f(x) = x^2 - 3x + 2$.

$\frac{df}{dx}$ = slope = 0 $\Rightarrow$ $2x - 3 = 0$

$\Rightarrow$ $x = 3/2$ — whether maxima or minima?

Random check $\rightarrow$ $f(3/2) < f(0)$. So, minima.

$f(x) = \log(1 + \exp(ax))$

$\frac{df}{dx} = \frac{a \cdot \exp(ax)}{1 + \exp(ax)} = 0$ } Solving this not so trivial

We'll learn some computational techniques to find minima & maxima.
└ Gradient Descent

Vector differentiation : Grad

x: vector $\quad x = \langle x_1, x_2, \cdots x_d \rangle$

$y = a^T x \quad a = \langle a_1, a_2, \cdots a_d \rangle$.

$f(x) = y = \sum_{i=1}^{d} a_i x_i$

$\frac{df}{dx} = \nabla_x f$ → grad or del.

$\nabla_x f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{bmatrix}$ → vector $\in \mathbb{R}^d$.

$\frac{\partial f}{\partial x_1}$ = partial diff^n.

$y = a^T x = \sum_{i=1}^{d} a_i x_i$

$= a_1 x_1 + a_2 x_2 + \cdots a_d x_d$.

$\nabla_x f = \begin{bmatrix} \frac{\partial f}{\partial x_1} \\ \frac{\partial f}{\partial x_2} \\ \frac{\partial f}{\partial x_3} \\ \vdots \\ \frac{\partial f}{\partial x_d} \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_d \end{bmatrix} = a$

So, $\nabla_x (a^T x) = a$.

↕ similarity

$\frac{d(ax)}{dx} = a$

Gradient Descent Algorithm

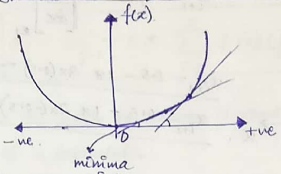
↳ Iterative algorithm

Say the optimisation eqn is $x^* = \underset{x}{\operatorname{argmin}} f(x)$

Initially, start off with some initial guess. (x_0).

Iteratively update x value & after say k iterations, it'll converge towards ~~same~~ the optimal value.

Core geometric idea of Gradient Descent



Left side of Minima :→ slope (-)ve.

Right side of minima :→ slope (+)ve.

At minima :→ slope is 0.

Observations

* Slope changes its sign from (-)ve to (+)ve at minima.
* slope gradually increases as we move from left to right.
* slope gradually decreases as we move from right to left.

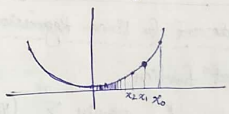
Gradient descent

① pick an initial pt = x_0 at random.

② $x_{i+1} = x_i - \gamma * \left[\frac{df}{dx}\right]_{x_i}$ → causes x to move towards minima irrespective of which side of minima it is currently.

step-size - (+)ve.

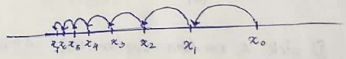
if $|x_{k+1} - x_k|$ is very small, then we terminate the iterations & declare $x^* = x_k$.



* As the pt moves more & more closer towards minima, the rate of convergence decreases.

$x_{i+1} = x_i - \gamma \left[\frac{df}{dx}\right]_{x_i}$ → update function

$\left[\frac{df}{dx}\right]_{x_0} > \left[\frac{df}{dx}\right]_{x_1} > \left[\frac{df}{dx}\right]_{x_2}$

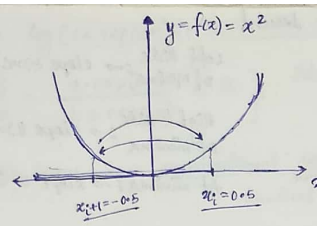


Learning Rate (γ)

There is a problem of oscillation if we keep the learning rate constant.

Let $y = f(x) = x^2$. $x_0 = 0.5$ & $\gamma = 1$.

So, $\frac{df}{dx} = 2x$



$$x_{i+1} = x_i - r * \left[\frac{df}{dx}\right]_{x_i}$$

$x_{i+1} = 0.5 - 1 * 2x(0.5) = -0.5$

$x_{i+2} = -0.5 - 1 * 2x(-0.5) = 0.5$

So, we basically keep on oscillating between 0.5 & -0.5. & will never converge to minima.

Remedy for oscillation → change r with each iteration.

one method → reduce r with each iteration.

$r = h(i)$ $\quad i$ = #iterations $\quad i \uparrow, r \downarrow$

In deep learning will cover more efficient ways of handling r.

Gradient descent for linear Regression

optimization function

$$w^* = \underset{w}{\operatorname{argmin}} \sum_{i=1}^{n} (y_i - w^T x_i)^2$$

{for simplicity, leaving out w_0 & regularization term}

$$L(w) = \sum_{i=1}^{n} (y_i - w^T x_i)^2$$

$$\nabla_w L = \sum_{i=1}^{n} \{2(y_i - w^T x_i)(-x_i)\}$$

① pick a random vector $w_0 = \langle w_1, w_2 \cdots w_d \rangle$.

② $w_1 = w_0 - r * \sum_{i=1}^{n} (-2x_i)(y_i - w_0^T x_i)$

③ $w_2 = w_1 - r * \sum_{i=1}^{n} (-2x_i)(y_i - w_1^T x_i)$.

$w_0, w_1, w_2 \cdots w_k, w_{k+1} \cdots$ stop when $w_{k+1} - w_k$ is a vector of very small values.

But there is a small issue with this gradient descent algo.

$$w_j = w_{j-1} - r * \sum_{i=1}^{n} (-2x_i)(y_i - w^T x_i)$$ → update function.

for the update to occurs, it requires to go through the whole dataset of n datapoints.

If n is large ($n \approx 1$ million), each converging step will take a significant amount of time.

In order to handle this problem we have stochiastic gradient descent.

Stochiastic Gradient Descent (SGD)

↳ the most important optimization algorithms.

GD:- $w_{j+1} = w_j - r * \sum_{i=1}^{n} (-2x_i)(y_i - w_j^T x_i)$

SGD:- $w_{j+1} = w_j - r * \sum_{i=1}^{k} (-2x_i)(y_i - w_j^T x_i)$, $1 \le k \le n$

So, SGD basically picks a random set of k-pts to update the weight vector.

k = # of random pts that we pick at each iteration for updating.
(also called batch size)

$w^*_{GD} = w^*_{SGD}$ ← proven.

* In each iteration we select random set of k-pts.

GD:- 100 iterations to converge. ($n = 1M$)

SGD:- $k = 1000$, takes > 100 iterations to converge. (500).

SGD:- $k = 100$, (1000) iterations to converge.

k: batch size in SGD.

often times $k = 1$ → 'SGD'.

$n << k > 1$ → batch SGD.

Constrained Optimization

General form of a constrained optimizat^n is as follows —

$$\max_x f(x) \quad \text{s.t.} \quad g(x) = c \text{ — equality constraint}$$
$$\& \; h(x) \geq d \text{ — inequality constraint.}$$

$f(x)$ → Objective function.

Lagrangian Multipliers

$$L(x, \lambda, \mu) = f(x) - \lambda\{g(x) - c\} - \mu\{d - h(x)\}$$

λ & μ are Lagrangian Multipliers.

$\lambda, \mu \geq 0$

The value of x found by solving → $\boxed{\frac{\partial L}{\partial x} = 0, \frac{\partial L}{\partial \lambda} = 0, \frac{\partial L}{\partial \mu} = 0}$

is same as solving the original constrained optimization.

Revisiting the PCA problem, we have

$$\max_u \frac{1}{n} \sum_{i=1}^{n} (u^T x_i)^2 \longrightarrow \frac{1}{n} \sum_{i=1}^{n} (u^T x_i)(u^T x_i).$$

s.t. $u^T u = 1$

$$= \frac{1}{n} \sum_{i=1}^{n} (u^T x_i)(x_i^T u).$$

$$= \frac{1}{n} u^T \left[\sum_{i=1}^{n} (x_i x_i^T)\right] \cdot u.$$

$$= u^T S u, \quad \boxed{S = \frac{1}{n} \sum_{i=1}^{n} x_i x_i^T}$$

S → covariance matrix

So, $\max_u u^T S u$, s.t. $u^T u = 1$.

$$L(u, \lambda) = u^T S u - \lambda(u^T u - 1)$$

$$\frac{\partial L}{\partial u} = 0 \Rightarrow \frac{\partial}{\partial u}(u^T S u - \lambda u^T u + \lambda) = 0$$

$$\Rightarrow 2(Su - \lambda u) = 0$$

$$\Rightarrow \boxed{Su = \lambda u}$$ — def^n of eigen value & eigen vector.

u: eigen vector, λ: eigen value } of S.

Logistic Regression Revisited

$$w^* = \underset{w}{\text{argmin}} \; (\text{logistic loss}) + \lambda w^T w$$

$\lambda w^T w$ ↑ regularizat^n.

Regularizat^n is nothing but Lagrangian multiplier.

We can write logistic regression as constrained optimization problem as follows →

$$w^* = \underset{w}{\text{argmin}} \; (\text{logistic loss}) \quad \text{s.t.} \quad w^T w = 1$$

(logistic loss) → objective function.

$$L = (\text{logistic loss}) - \lambda(1 - w^T w).$$

$$L = (\text{logistic loss}) + \lambda w^T w - \lambda$$

↓ same as L^{2}R. with regularization.

* So, regularization can be thought as imposing an equality constraint.

Why L1-Regularization creates sparsity

$$\min_w \; \underbrace{\text{loss} + \lambda \|w\|_2^2}_{L_2}.$$

Only difference in both is $\|w\|_2^2$ & $\|w\|_1$

So will focus on that.

$$\min_{w_1, w_2 \ldots w_d} (w_1^2 + w_2^2 + \cdots + w_d^2)$$

$$\min_{w_i} (w_i^2) \text{ - considering only 1 component for simplicity.}$$

$w_i^2 \rightarrow L_2(w_i)$

$$\frac{\partial L_2}{\partial w_1} = 2w_1$$

$$\text{So, } (w_1)_{j+1} = (w_1)_j - r(2 * w_j)$$

$r = 0.01$, $w_1 j = 0.05$

$2 \times r \times w_j = 0.001$ v. small.

Becomes further smaller on convergence. So L2 reg doesn't change value after few iterations.

$$\underset{w}{\min} \; \underbrace{\text{loss} + \lambda \|w\|_1}_{L1}$$

$$\min_{w_1, w_2 \ldots} (|w_1| + |w_2| + |w_3| - \cdots - |w_d|)$$

$$\min_{w_1} (|w_1|) \rightarrow L_1(w_1)$$

$$\frac{\partial L_1}{\partial w_1} = \begin{cases} 1, & w_1 > 0 \\ 0 - 1, & w_1 < 0 \end{cases}$$

say w_1 is (+)ve.

Then, $(w_1)_{j+1} = (w_1)_j - (r) * 1$

$r \times 1 = 0.01$ small.

So converges faster than L2, hence more likely to converge towards 0.

* Gradient is constant — major advantage.