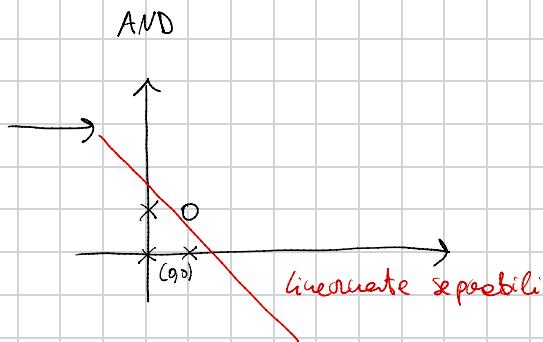


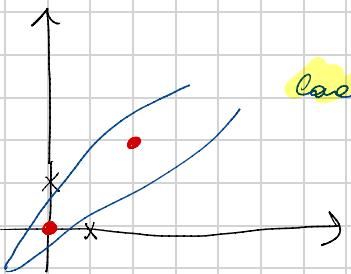
Problema dello XOR

XOR	x_1	x_2	AND	OR	XOR
	0	0	0	0	0
	1	0	0	1	1
	0	1	0	1	1
	1	1	1	1	0



lineariamente separabili

XOR



Cose separano?

Singolo perceptron non è in grado di risolvere il problema dello XOR

↳ Linee!

↳ Perceptron è un classificatore lineare!

↳ Geometricamente

Consideriamo che
pole lato della retta
d'appoggio i punti?

$$v_0 + v_1 x_1 + v_2 x_2 > 0$$

$$x_2 v_2 + x_2 v_2 - v_0 \Rightarrow x_2 v_1 + x_2 v_2 > t \quad \text{qui}$$

$$\Rightarrow 0 v_1 + 0 v_2 < t \Rightarrow t > 0$$

Class 1 se $v_1 + v_2 > t$

Class 0 se $v_1 + v_2 \leq t$

↳ Perceptron in uno spazio bidimensionale
→ iperplane

↳ Solo lineerette per separare classi!

↳ Punti dati dello XOR NON possono essere separati
in due classi distinte con una singola linea retta!

$$\begin{cases} 1 v_1 + 1 v_2 > t \\ 1 v_1 + 1 v_2 \leq t \end{cases} \quad \begin{cases} v_1 > 0 \\ v_2 > 0 \end{cases} \quad \begin{cases} v_1 > 0 \\ v_2 > 0 \end{cases}$$

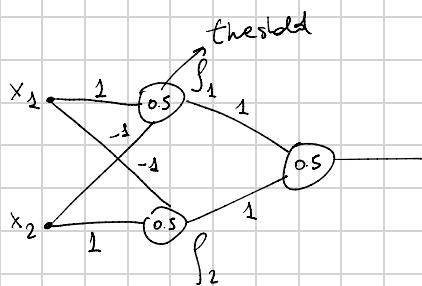
Nessun singolo perceptron può risolvere lo XOR!
Perceptron può solo compiere confronti decisionali lineari

$t > v_1 + v_2 > 2t \Rightarrow \text{CONTRODIREZIONE!}$

↳ Sappiamo che $v_1 > t, v_2 > t$
 $\Rightarrow v_1 + v_2 > 2t$ da entrambe le
 $v_1 + v_2 < t$

Risoluzione problema XOR \rightarrow Appigge un hidden layer
 Problema A \rightarrow come deciderlo?
 Come determinare gli hidden neurons?

Rete a due strati di Perceptron \rightarrow Componete lo XOR!



$$f_1(x_1, x_2) = [x_1 - x_2 > 0.5] \begin{cases} 0 \\ 1 \end{cases}$$

$$f_2(x_1, x_2) = [x_2 - x_1 > 0.5] \begin{cases} 0 \\ 1 \end{cases}$$

$$f_3(f_1(x_1, x_2), f_2(x_1, x_2)) = 1[f_1(x_1, x_2) + 1f_2(x_1, x_2) > 0.5] \begin{cases} 0 \\ 1 \end{cases}$$

\Downarrow Compo f3

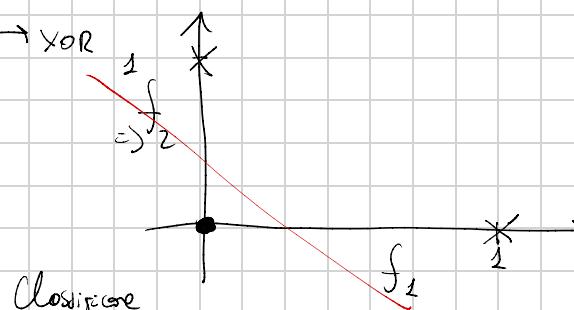
x_1	x_2	f_1	f_2	f_3
0	0	0	0	0
1	0	1	0	1
0	1	0	1	1
1	1	0	0	0

Possiamo plottare il campione decisionale

$$f_2 = -f_1 + 0.5$$

Problema XOR risolto! \rightarrow non sapevo come decidere!

\hookrightarrow Prima delle scuole della backpropagation non esisteva un algoritmo per apprendere i pesi dei layer nascosti



Adesso possiamo separare linearmente!

Ripetizione → Concetto → trovare una funzione f che mappa da uno spazio di input \mathbb{R}^m a uno spazio di output \mathbb{R}^n

$f_{\text{reg}} : \mathbb{R}^m \rightarrow \mathbb{R}^n$

↓
parametrizzata

Esempio Regressione → Stima pos. oggetto in un'immagine

Previsione prezzo di una casa → $f_{\text{reg}} : \mathbb{R}^m \rightarrow \mathbb{R}^n$

↓
PREDICTION valore continuo (insieme di valori continui) basandosi su un input

OBJETTIVO → Trovare una funzione che possa modellare le relazioni fra le variabili di input e le variabili (o variabili) di output.

In generale

$x \in \mathbb{R}^m$	value input
$y \in \mathbb{R}^n$	goal that
$\hat{y} \in \mathbb{R}^n$	value predicted

quindi $\hat{y} = f(x)$

→ trovare una f che mappa da uno spazio \mathbb{R}^m a \mathbb{R}^n

→ Cosa succede?

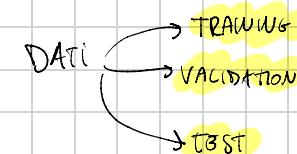
TIROVARE LA FUNZIONE di REGRESSIONE f

Consideriamo di avere un Dataset D di coppie di dati etichettati

$$D = \{(x^{(i)}, y^{(i)})\}$$

↓

→ Ogni coppia di dati $(x^{(i)}, y^{(i)})$ è composta da un input $x^{(i)} \in \mathbb{R}^m$ e da un target $y^{(i)} \in \mathbb{R}^n$.
Assumiamo di dividere dataset in



Misure di Performance

Sicendo il preditore di ML, dobbiamo definire una misura per valutare le performance del voto o output di regressione.

Consideriamo:

$$\begin{matrix} \text{pred campionato diretto} \\ \sum_{i=1}^n g_i^{(c)} \end{matrix} \rightarrow \begin{matrix} \text{voto diretto} \\ g_{\text{true}} \end{matrix}$$

$$g = \begin{bmatrix} g_1^{(c)} \\ \vdots \\ g_n^{(c)} \end{bmatrix}$$

$$\begin{aligned} \hat{g} &= \left\{ f_{\theta}(x^{(c)}) \right\}_{i=1}^m \\ \hat{g} &= \begin{bmatrix} \hat{g}_1^{(c)} \\ \vdots \\ \hat{g}_m^{(c)} \end{bmatrix} \end{aligned}$$

$$\text{dedicate } g_{\text{true}} = \hat{g}_{\text{true}}$$

OBJETTIVO del predice è: approssimato (\hat{g}_{true})

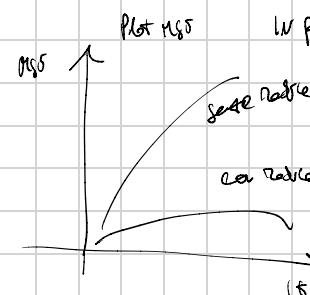
TRAVERSANDO i valori ottenuti dei predetti \hat{g} che minimizzano queste forme di costo. Minimizzare le forme di costo significa fare più errori di previsione del modello sui dati di training

MSB → formule di costo per la regressione lineare

Consideriamo $g \in \mathbb{R}^m$ grand truth
 $\hat{g} \in \mathbb{R}^m$ valore predetto] vettore un-dimensionale

\hat{g} è una buona approssimazione di g ?
 Calcolare distanza euclidea

$$\|g - \hat{g}\|_2 = \sqrt{\sum_{i=1}^m (\hat{g}_i - g_i)^2}$$



In pratica, mai spesso utilizziamo la distanza euclidea senza radice

→ Vogliamo far pesare di più gli errori più grandi

$$\text{error}(g - \hat{g}) = \|g - \hat{g}\|_2^2 = \sum_{i=1}^m (\hat{g}_i - g_i)^2$$

Vai a prendere l'errore medio dell'intero di tutto il test set per avere uno strutturale di prestazioni → MSB

$$\text{MSB}(g_{\text{true}}, \hat{g}_{\text{true}}) = \frac{1}{m} \sum_{j=1}^m \|g_j^{(c)} - \hat{g}_j^{(c)}\|_2^2$$

Queste misure di performance sono razionali ma non siamo sicuri.

Un buon regolare avrà un errore basso.

Casi Speciali di Regressione

$$f_{\text{reg}} : \mathbb{R}^m \rightarrow \mathbb{R}^m$$

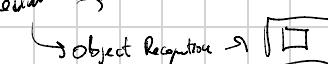
$m = n = 1 \rightarrow$ Simple Regression, cioè $f_{\text{reg}}(x) = g$

$n > 1$ AND $m = 1$ Multiple regression, cioè $f_{\text{reg}}(x) = g$

$n \geq 1$ AND $m \geq 1$ Multivariata Regression

dei dati i gesti più pronostici sono

Modelli Linea Regressione



Modello di Regressione Lineare

Regressione Lineare modello che puo' essere applicato a tre casi specifici di regressione:

Regressione semplice $\rightarrow m = 1 \Rightarrow f(x) = V_0 + V_1 x_1$

$f(x) = V_0 + V_1 x_1$

parametri del modello

intervale
caso
modello lineare

Per imparare un regressore lineare significa trovare i valori per i parametri V_0 e V_1 tali che piu' buon predittore per y

Regressione multiple $\rightarrow m > 1 \wedge n = 1 \rightarrow f(x) = V_0 + V_1 x_1 + \dots + V_m x_m = V^T x \rightarrow$ in mezzo metrica

Regressione multivariata $\rightarrow m > 1 \wedge n > 1 \rightarrow f_v(x) = \begin{bmatrix} f_1(x) \\ f_2(x) \\ \vdots \\ f_m(x) \end{bmatrix}$

Piu' problemi lineari da risolvere

Loss Function

Ci concentriam sulle forme generali delle regressione lineare multiple:

$$f_{\text{reg}}(x) = V_0 + V_1 x_1 + \dots + V_m x_m$$

dati di training

Dove f_{reg} indice che f dipende dai parametri: $V = (V_0, \dots, V_m)$.

Per imparare il regressore f significa trovare i valori per i parametri V che minimizzano l'errore nel TRAINING SET

Noi possiamo quantificare l'errore complessivo del regressore nel TEST SET definendo le separate FUNZIONI DI COSTO

$$J(V) = \frac{1}{2} \sum_{i=1}^m (f_{\text{reg}}(x^{(i)}) - y^{(i)})^2$$

Funzioni di costo devono essere differenziabili!

Disequazione del gradiente

Differenziabile!

implicazione

simile ad MSE!

idealmente noi vogliamo avere il costo simile a zero \Rightarrow MSE training set $= 0$

MINIMIZZARE $J(V)$

→ Trovare i parametri che rendono le predizioni del modello il più vicino possibile ai valori reali nei dati di training prediciti così esatti!

Quindi una buona scelta per V sarebbe quella che minimizza le funzione costo $J(V)$

$$V = \underset{V}{\operatorname{arg\,min}} J(V) \rightarrow \text{obiettivo} \rightarrow \text{ridurre } V \text{ per ottenere } V \text{ buoni!}$$

Discesa del Gradiente

April 17, 2025

1 Discesa del Gradiente

Nel contesto dell'addestramento di modelli di regressione per trovare i parametri ottimali abbiamo visto come algoritmo di ottimizzazione l'algoritmo di discesa del gradiente. Nel contesto della regressione, l'obiettivo è trovare una funzione f parametrizzata da θ che possa mappare un input x in R^n a un output y in $R^{n'}$. Per fare ciò, è necessario aggiustare i parametri θ del modello utilizzando i dati di training. Abbiamo visto una funzione di costo $J(\theta)$ che misura l'errore del modello sui dati di training in funzione dei suoi parametri θ . Per la regressione lineare, una tipica funzione di costo è l'Errore Quadratico Medio (MSE), o una forma ad esso proporzionale come:

$$J(\theta) = \frac{1}{2} \sum_{i=1}^m (f_\theta(x^{(i)}) - y^{(i)})^2$$

Abbiamo introdotto $\frac{1}{2}$ poichè ci semplifica i calcoli durante la derivazione (ad

esempio se applichiamo il gradiente)

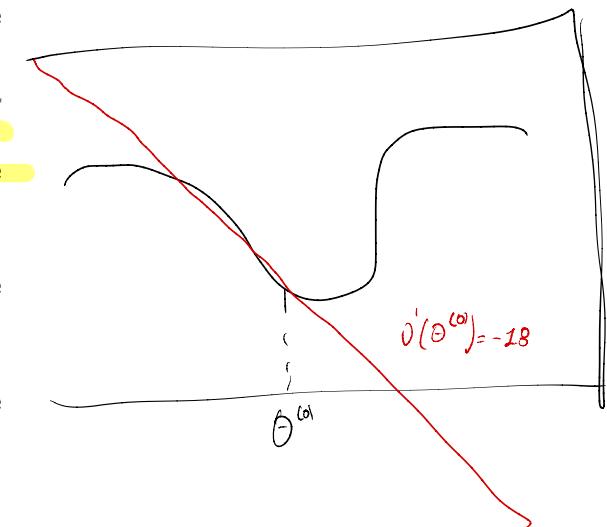
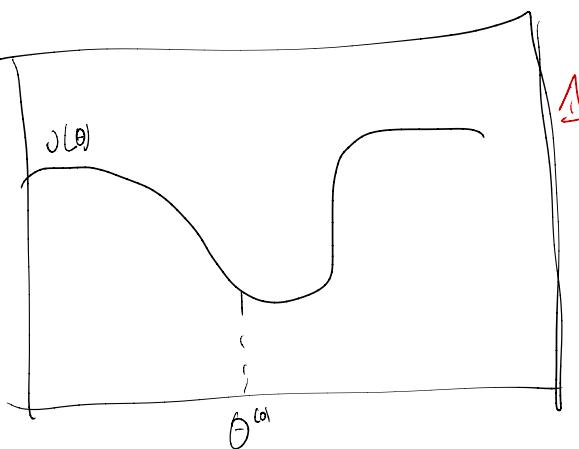
Questa funzione di costo esplicita i parametri θ tramite la funzione $f_\theta(x^{(i)})$ che rappresenta le **previsioni del modello dipendenti da θ** . Questa formulazione è specifica per modelli parametrici, come la regressione lineare, dove le previsioni sono calcolate come una funzione esplicita dei parametri θ .

L'obiettivo dell'addestramento è **trovare i valori di θ che minimizzano $J(\theta)$** .

Una possibilità per risolvere questo problema sarebbe quello di calcolare $J(\theta)$ per tutti i valori possibili di θ e scegliere i valori di θ che minimizzano il costo. Tuttavia, questa opzione non è praticabile poiché θ può assumere un numero infinito di valori. Quindi abbiamo bisogno di un modo per trovare i valori di θ che minimizzano $J(\theta)$ senza dover calcolare $J(\theta)$ per tutti i possibili valori di θ . Introduciamo l'algoritmo della discesa del gradiente considerando inizialmente il problema di minimizzare una funzione di una singola variabile $J(\theta)$. Successivamente estenderemo al caso di più variabili.

L'algoritmo della discesa del gradiente si basa sull'osservazione che, **se una funzione $J(\theta)$ è definita e differenziabile in un intorno di un punto θ^0 , allora $J(\theta)$ diminuisce più rapidamente se ci si muove da θ^0 nella direzione della derivata negativa di J calcolata in θ^0** . Consideriamo la funzione $J(\theta)$ mostrata nel grafico presente nelle slide: Supponiamo di essere al punto iniziale θ^0 . Dal grafico, possiamo vedere che dovremmo muoverci verso la parte destra dell'asse x per raggiungere il minimo della funzione.

La derivata prima della funzione in quel punto $J'(\theta^0)$ sarà uguale al coefficiente angolare della tangente alla curva nel punto $(\theta^0, J(\theta^0))$. Poichè



la curva sta diminuendo in un intorno di θ^0 , anche la retta tangente sarà decrescente. Pertanto, il suo coefficiente angolare di $J'(\theta^0)$ sarà negativo. Se vogliamo spostarci a destra, dovremmo seguire la direzione inversa della derivata della curva in quel punto. La discesa del gradiente è un algoritmo iterativo; quindi, non stiamo cercando di raggiungere il minimo della funzione in un solo passo. Invece, vorremmo spostarci in un altro punto θ^1 tale che $J(\theta^1) < J(\theta^0)$. Se possiamo farlo per ogni punto, possiamo raggiungere il minimo in un certo numero di passi.

Ad ogni passo, ci muoveremo proporzionalmente al valore della derivata. Questo si basa sull'osservazione che valori assoluti più grandi della derivata indicano curve più ripide. Se scegliamo un fattore moltiplicativo γ , ci sposteremo al punto:

$$\theta^1 = \theta^0 - \gamma J'(\theta^0)$$

Ad esempio, se scegliamo $\gamma = 0.02$, ci sposteremo al punto $\theta^1 = 0.4 + 0.02 * 1.8 = 0.436$. La procedura lavora iterativamente fino a che la derivata è così piccola che non è possibile alcun movimento.

Quando siamo in quella situazione, ci troveremo in un minimo locale. L'ottimizzazione termina qui. Abbiamo trovato il valore $\theta^3 = \arg\min J(\theta)$.

In pratica, l'algoritmo viene terminato seguendo un dato criterio di terminazione.

1.1 Una variabile

L'algoritmo del gradiente discendente può essere scritto nella seguente forma nel caso di una variabile:

1. Scegliere un punto iniziale casuale θ ;
2. Calcolare la derivata prima della funzione J' nel punto corrente θ : $J'(\theta)$;
3. Aggiornare la posizione del punto corrente utilizzando la formula $\theta = \theta - \gamma J'(\theta)$;
4. Ripetere i passaggi 2-3 fino a che non vengono soddisfatti alcuni criteri di terminazione.

1.2 Variabili multiple

L'algoritmo del gradiente discendente si generalizza al caso in cui la funzione J da ottimizzare dipende da più variabili $J(\theta_1, \theta_2, \dots, \theta_n)$.

Ad esempio, consideriamo una funzione di due variabili $J(\theta_1, \theta_2)$. Possiamo tracciare tale funzione come un grafico 3D (a sinistra) o come un grafico a contorno (a destra). In entrambi i casi, il nostro obiettivo è raggiungere il punto con il valore minimo (il "centro" dei due grafici).

Dato un punto $\theta = (\theta_1, \theta_2)$, la direzione di discesa più ripida è il gradiente della funzione nel punto.

Il gradiente è una generalizzazione a più variabili della derivata. Il gradiente di una funzione di n variabili calcolato in un punto θ è un vettore la cui i -esima componente è data dalla derivata parziale della funzione rispetto alla i -esima variabile:

$$\nabla J(\boldsymbol{\theta}) = \begin{pmatrix} J_{\theta_1}(\boldsymbol{\theta}) \\ J_{\theta_2}(\boldsymbol{\theta}) \\ \vdots \\ J_{\theta_n}(\boldsymbol{\theta}) \end{pmatrix}$$

Figure 1: Didascalia dell'immagine

Nel caso di due variabili, il gradiente sarà un vettore bidimensionale (il gradiente) che indica la direzione da seguire. Poiché in generale vogliamo ottimizzare funzioni a più variabili, l'algoritmo è chiamato "discesa del gradiente". Lo pseudocodice della procedura, nel caso di più variabili, è il seguente:

1. Inizializzare $\theta = (\theta_1, \theta_2, \dots, \theta_n)$ casualmente.
2. Per ogni variabile x_i :

- Calcolare la derivata parziale nel punto:

$$\frac{\partial}{\partial \theta_i} J(\theta)$$

- Aggiornare la variabile corrente utilizzando la formula:

$$\theta_i = \theta_i - \gamma \frac{\partial}{\partial \theta_i} J(\theta)$$

3. Ripetere il passo 2 finché non sono soddisfatte le condizioni di terminazione.

1.3 Discesa del gradiente e regressione lineare

Useremo l'algoritmo della discesa del gradiente per ottimizzare il nostro problema di regressione lineare, ovvero per trovare $\hat{\theta} = \arg_{\theta} \min J(\theta)$. Ciò viene fatto inizializzando θ casualmente (questo fornirà il punto di partenza per la discesa del gradiente) e applicando ripetutamente la seguente regola di aggiornamento per ciascuna variabile:

$$\theta_j = \theta_j - \gamma \cdot \frac{\partial}{\partial \theta_j} J(\theta)$$

Per implementare questo, dobbiamo calcolare le derivate parziali della funzione di costo rispetto a ciascun parametro. Scriviamo prima esplicita-

mente la funzione di costo in termini dei parametri θ :

$$J(\theta) = \frac{1}{2} \sum_{i=1}^N (f_\theta(\mathbf{x}^{(i)}) - y^{(i)})^2 = \frac{1}{2} \sum_{i=1}^N (\theta_0 x_0^{(i)} + \theta_1 x_1^{(i)} + \dots + \theta_n x_n^{(i)} - y^{(i)})^2$$

Dove abbiamo introdotto i termini $x_0^{(i)} = 1$, come precedentemente discusso. Possiamo calcolare la derivata parziale della funzione di costo rispetto alla j -esima variabile come segue:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \frac{1}{2} \sum_{i=1}^N 2(f_\theta(\mathbf{x}^{(i)}) - y^{(i)}) \frac{\partial}{\partial \theta_j} (\theta_0 x_0^{(i)} + \dots + \theta_n x_n^{(i)} - y^{(i)})$$

Notiamo che:

$$\frac{\partial}{\partial \theta_j} (\theta_0 x_0^{(i)} + \dots + \theta_n x_n^{(i)} - y^{(i)}) = x_j^{(i)}$$

Quindi, otteniamo:

$$\frac{\partial}{\partial \theta_j} J(\theta) = \sum_{i=1}^N (f_\theta(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}$$

La regola di aggiornamento può essere scritta come segue:

$$\theta_j = \theta_j - \gamma \sum_{i=1}^N (f_\theta(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)}$$

Notiamo che la regola di aggiornamento sopra è essenzialmente l'algoritmo ADALINE!

$$\begin{aligned} J(\theta) &= \frac{1}{2} \sum_{i=1}^N (f_\theta(\mathbf{x}^{(i)}) - y^{(i)})^2 \\ &= \frac{1}{2} \sum_{i=1}^N 2(f_\theta(\mathbf{x}^{(i)}) - y^{(i)}) \frac{\partial}{\partial \theta_j} \\ &\quad \text{with } f_\theta(\mathbf{x}^{(i)}) = \theta_0 x_0^{(i)} + \dots + \theta_n x_n^{(i)} - g^{(i)} \\ &\quad \text{and } x_j^{(i)} = 1 \\ \theta_j &= \theta_j - \gamma \sum_{i=1}^N (f_\theta(\mathbf{x}^{(i)}) - y^{(i)}) x_j^{(i)} \end{aligned}$$

1.4 Lezione in classe

Per utilizzare un algoritmo di ottimizzazione come la **discesa del gradiente**, la **funzione di costo deve essere differenziabile**. La differenzialità permette di calcolare il **gradiente della funzione di costo rispetto ai parametri**, il quale indica la direzione di massima pendenza.

1.4.1 Cos'è la Discesa del Gradiente?

L'**algoritmo di discesa del gradiente** è un **algoritmo iterativo** che parte da un'inizializzazione casuale dei parametri θ .

Ad ogni iterazione, i parametri vengono **aggiornati nella direzione opposta al gradiente della funzione di costo**, con l'obiettivo di muoversi verso il minimo della funzione.

La formula generale per l'aggiornamento di un parametro θ_j è:

$$\theta_j^t = \theta_j^{t-1} - \alpha \frac{\partial J(\theta^t)}{\partial \theta_j}$$
. Dove α è il learning rate, un iperparametro che determina la dimensione del passo compiuto ad ogni iterazione.

Viene utilizzata nella formula di aggiornamento della discesa del gradiente la **derivata parziale della funzione di costo MSE** (specificata per la regressione lineare) rispetto ad un singolo parametro θ_j .

E' cruciale monitorare il valore della funzione di costo $J\theta$ ad ogni iterazione per verificare se l'apprendimento sta procedendo correttamente. Un valore di $J\theta$ che diminuisce nel tempo suggerisce che i parametri si stanno

muovendo verso un minimo, riducendo l'errore sul **training set**. Se il valore di $J(\theta)$ dovesse aumentare, potrebbe indicare un **learning rate** troppo elevato.

La scelta del **learning rate** (α) è importante. Un learning rate troppo piccolo può rendere la convergenza molto lenta, mentre un learning rate troppo grande può far "saltare" l'algoritmo oltre il minimo, impedendo la convergenza. La determinazione di un buon learning rate può avvenire tramite l'osservazione del grafico della funzione di costo nel tempo.

Inoltre, la discesa del gradiente è in grado di trovare un minimo locale della funzione di costo, che potrebbe non essere il minimo globale.

Anche la **feature scaling** deve essere trattata. Se le feature hanno scale molto diverse, la funzione di costo può avere una forma che rende difficile la convergenza per l'algoritmo di discesa del gradiente. Normalizzare le feature, ad esempio tra 0 e 1, può rendere più facile e veloce raggiungere il minimo.