Chain Rule e Ottimizzazione dei Calcoli

Lezione per Liceo Scientifico (4°-5° anno)

1 Prerequisiti

- Derivate di funzioni elementari (potenze, esponenziali, logaritmi, trigonometriche)
- Composizione di funzioni: $(f \circ g)(x) = f(g(x))$
- Regola della catena (chain rule): se $h(x) = f(g(x))$, allora $h'(x) = f'(g(x)) \cdot g'(x)$

---

2 Il Problema: Funzioni Composte Multiple

2.1 Situazione

Immaginiamo di avere una funzione composta da **molti strati**:

$$x \to g_1(x) \to g_2(\cdot) \to g_3(\cdot) \to ... \to g_n(\cdot) \to y$$

Esempio concreto con 3 funzioni:

$$y = f(x) = e^{\sin(x^2+1)} \tag{1}$$

Possiamo scomporla come composizione:

- $g_1(x) = x^2 + 1$ (funzione quadratica)
- $g_2(u) = \sin(u)$ (funzione seno)
- $g_3(v) = e^v$ (funzione esponenziale)
- $y = g_3(g_2(g_1(x)))$

---

3 Obiettivo Didattico

Vogliamo calcolare **in un punto x_0**:
1. Il **valore** della funzione: $y = f(x_0)$
2. La **derivata** della funzione: $y' = f'(x_0)$

Domanda chiave: Come possiamo essere **efficienti** ed evitare calcoli ridondanti?

---

4 Approccio Naive (Inefficiente)

4.1 Calcolo del valore $y = f(x_0)$

Calcoliamo dall'interno verso l'esterno con $x_0 = 0.5$:

Step	Calcolo	Risultato
1	$u_1 = g_1(0.5) = (0.5)^2 + 1$	1.25
2	$u_2 = g_2(1.25) = \sin(1.25)$	0.9490
3	$y = g_3(0.9490) = e^{0.9490}$	2.5832

4.2 Calcolo della derivata $f'(x_0)$ - Approccio Naive

4.2.1 Passo 1: Derivare la formula analitica

Prima troviamo la formula generale di $f'(x)$ usando la chain rule.

Abbiamo: $f(x) = e^{\sin(x^2+1)}$

Scomponiamo:
- $g_1(x) = x^2 + 1$
- $g_2(u) = \sin(u)$
- $g_3(v) = e^v$

Derivate delle singole funzioni:
- $g_{1'}(x) = 2x$
- $g_{2'}(u) = \cos(u)$
- $g_{3'}(v) = e^v$

Applicazione della chain rule:

$$f'(x) = g_{3'}(g_2(g_1(x))) \cdot g_{2'}(g_1(x)) \cdot g_{1'}(x) \tag{2}$$

Sostituiamo le derivate:

$$f'(x) = e^{\sin(x^2+1)} \cdot \cos(x^2 + 1) \cdot 2x \tag{3}$$

Forma finale (NON si semplifica ulteriormente!):

$$f'(x) = 2x \cdot \cos(x^2 + 1) \cdot e^{\sin(x^2+1)} \tag{4}$$

Nota: La derivata contiene **tutte e tre le funzioni** (x^2, sin, e) - non c'è semplificazione!

4.2.2 Passo 2: Valutare numericamente in $x_0 = 0.5$

Sostituiamo $x = 0.5$:

$$f'(0.5) = 2 \cdot 0.5 \cdot \cos(0.5^2 + 1) \cdot e^{\sin(0.5^2+1)} \tag{5}$$

$$f'(0.5) = 1.0 \cdot \cos(1.25) \cdot e^{\sin(1.25)} \tag{6}$$

Calcoliamo ogni componente:

- $x^2 + 1 = 0.25 + 1 = 1.25$ ← **RICALCOLATO** (già fatto prima!)
- $\sin(1.25) \approx 0.9490$ ← **RICALCOLATO** (già fatto prima!)
- $\cos(1.25) \approx 0.3153$
- $e^{\sin(1.25)} = e^{0.9490} \approx 2.5832$ ← **RICALCOLATO** (era già y!)

Quindi:

$$f'(0.5) = 1.0 \cdot 0.3153 \cdot 2.5832 \approx 0.8145 \tag{7}$$

Problema: Abbiamo **RICALCOLATO** i valori intermedi che avevamo già ottenuto:

- $x^2 + 1 = 1.25$ (già calcolato per y)
- $\sin(1.25) = 0.9490$ (già calcolato per y)
- $e^{\sin(1.25)} = 2.5832$ (questo **È** y che avevamo già calcolato!)

---

5 Approccio Ottimizzato: Memorizzare i Valori Intermedi

5.1 Idea Chiave

Quando calcoliamo y, salviamo tutti i valori intermedi in memoria.

Poi, quando calcoliamo $f'(x)$, **riusiamo questi valori** invece di ricalcolarli!

5.2 Passo 1: Calcolo di y (salvando i valori intermedi)

Calcoliamo la funzione normalmente e **salviamo** tutti i valori intermedi:

Layer	**Calcolo**	**Risultato**	**Azione**
Input	x	0.5	SALVA x
1	$u_1 = g_1(x) = x^2 + 1$	1.25	SALVA u_1
2	$u_2 = g_2(u_1) = \sin(1.25)$	0.9490	SALVA u_2
3	$y = g_3(u_2) = e^{0.9490}$	2.5832	SALVA y

Valori salvati in memoria: $x = 0.5$, $u_1 = 1.25$, $u_2 = 0.9490$, $y = 2.5832$

5.3 Passo 2: Calcolo di $f'(x)$ (riusando i valori salvati)

Ora calcoliamo la derivata usando la **chain rule**:

$$f'(x) = g_{1'}(x) \cdot g_{2'}(u_1) \cdot g_{3'}(u_2) \tag{8}$$

L'ottimizzazione: Invece di ricalcolare u_1 e u_2, li prendiamo dalla memoria!

Layer	**Derivata**	**Usa valore salvato**	**Risultato**
1	$g_{1'}(x) = 2x$	$x = 0.5$	1.0
2	$g_{2'}(u_1) = \cos(u_1)$	$u_1 = 1.25$	0.3153
3	$g_{3'}(u_2) = e^{u_2}$	$u_2 = 0.9490$	2.5832

Chain rule: $f'(0.5) = 1.0 \times 0.3153 \times 2.5832 = 0.8145$

---

6 Confronto: Naive vs Ottimizzato

6.1 Approccio Naive

Operazioni totali:

- Calcolo di y: 3 funzioni
- Calcolo di y': 3 derivate + **RICALCOLO** di g_1, g_2
- **Totale: 8 valutazioni**

6.2 Approccio Ottimizzato

Operazioni totali:

- Calcolo y (salvando valori): 3 funzioni
- Calcolo y' (riusando valori): 3 derivate
- **Totale: 6 operazioni**

Risparmio: 25% in questo esempio semplice

Con funzioni più complesse (10-100 layer): il risparmio diventa **50%+**!

---

7 Calcolo Unificato: Funzione e Derivata in Parallelo

7.1 Schema del Flusso di Calcolo

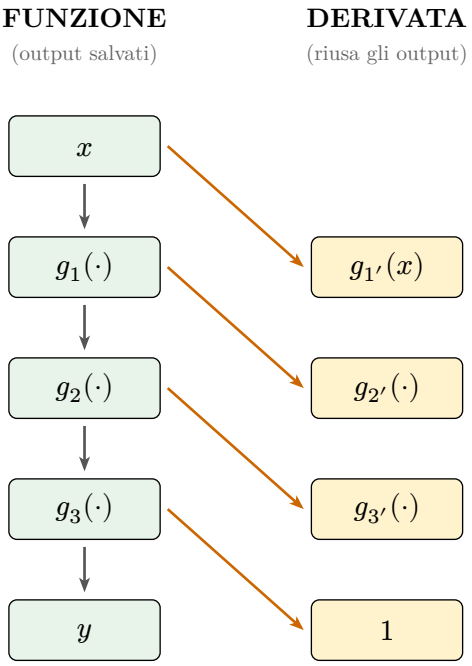

Figura 1: Schema del flusso di calcolo (esempio con 3 funzioni)

7.1.1 Schema Generale (n funzioni)

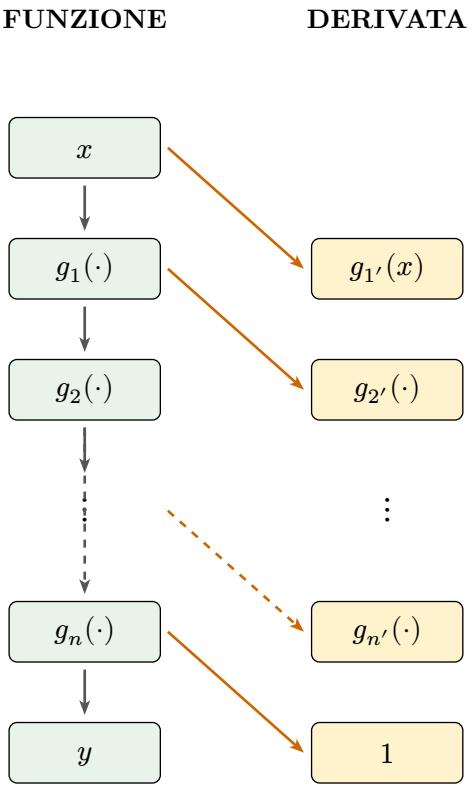

Figura 2: Schema generale per composizione di n funzioni: $y = g_n(g_{n-1}(...g_2(g_1(x))...))$

7.2 Tabella Riassuntiva

Input	Funzione	Output	→	Derivata (usa output a sinistra)	Valore
x_0	x	**0.5**	→	$g_{1'}(x) = 2x$	1.0
0.5	$g_1(x) = x^2 + 1$	**1.25**	→	$g_{2'}(u_1) = \cos(u_1)$	0.3153
1.25	$g_2(u_1) = \sin(u_1)$	**0.9490**	→	$g_{3'}(u_2) = e^{u_2}$	2.5832
0.9490	$g_3(u_2) = e^{u_2}$	**2.5832**		(nessuna)	

Risultati:

- $f(0.5) = y = 2.5832$
- $f'(0.5) = 1.0 \times 0.3153 \times 2.5832 = 0.8145$

7.3 Cosa mostra questo schema?

1. **Frecce verticali (↓)**: L'output di ogni funzione diventa l'input della funzione successiva
2. **Frecce orizzontali (→)**: L'output della funzione (colonna sinistra) è usato come input per calcolare la **derivata successiva** (colonna destra)
3. **Sfalsamento**: La derivata $g_{(i+1)'}(\cdot)$ usa l'output di $g_i(\cdot) \rightarrow$ sono sulla stessa riga!
4. **Nessun ricalcolo**: Le derivate «pescano» i valori già calcolati dalla colonna di sinistra
5. **Ultima riga**: L'ultima funzione (g_3) non ha derivata corrispondente a destra

8 Esempio con Funzione Più Complessa (6 Layer)

8.1 Funzione

$$y = f(x) = \cos\left(\sqrt{e^{\sin(\ln(x^2))}}\right) \tag{9}$$

Composizione di 6 funzioni:

- $g_1(x) = x^2$
- $g_2(u) = \ln(u)$
- $g_3(v) = \sin(v)$
- $g_4(w) = e^w$
- $g_5(z) = \sqrt{z}$
- $g_6(t) = \cos(t)$

8.2 Tabella di Calcolo Dettagliata

Input	Funzione	Output	→	Derivata (usa output a sinistra)	Valore
x_0	x	**1.5**	→	$g_{1'}(x) = 2x$	3.0000
1.5	$g_1(x) = x^2$	**2.2500**	→	$g_{2'}(u_1) = \frac{1}{u_1}$	0.4444
2.2500	$g_2(u_1) = \ln(u_1)$	**0.8109**	→	$g_{3'}(u_2) = \cos(u_2)$	0.6890
0.8109	$g_3(u_2) = \sin(u_2)$	**0.7247**	→	$g_{4'}(u_3) = e^{u_3}$	2.0640
0.7247	$g_4(u_3) = e^{u_3}$	**2.0640**	→	$g_{5'}(u_4) = \frac{1}{2\sqrt{u_4}}$	0.3480
2.0640	$g_5(u_4) = \sqrt{u_4}$	**1.4367**	→	$g_{6'}(u_5) = -\sin(u_5)$	−0.9911
1.4367	$g_6(u_5) = \cos(u_5)$	**0.1330**		(nessuna)	

8.3 Risultati

Valore della funzione:

$$f(1.5) = 0.1330 \tag{10}$$

Derivata (prodotto di tutte le derivate):

$$f'(1.5) = 3.0 \times 0.4444 \times 0.6890 \times 2.0640 \times 0.3480 \times (-0.9911) \tag{11}$$

$$f'(1.5) = -0.6156 \tag{12}$$

8.4 Osservazioni

1. **6 funzioni, 6 valori salvati**: $u_1, u_2, u_3, u_4, u_5, y$
2. **6 derivate calcolate**: ognuna usa il valore della riga corrispondente
3. **Nessun ricalcolo**: ogni valore nella colonna FUNZIONE è usato esattamente una volta per la derivata
4. **Segno negativo**: la derivata di cos è $-\sin$, quindi $f'(1.5) < 0$

8.5 Confronto con Approccio Naive

Approccio Naive:

- Calcolo y: 6 funzioni
- Calcolo y': ricalcolo 5 valori + 6 derivate
- **Totale: 17 operazioni**

Approccio Ottimizzato:

- Calcolo y (con memorizzazione): 6 funzioni
- Calcolo y' (riusando valori): 6 derivate
- **Totale: 12 operazioni**

Risparmio: 30% (e aumenta con più layer!)

---

9 Collegamento con le Reti Neurali

9.1 Perché è importante per il Deep Learning?

Nelle reti neurali:

- Ogni **layer** è una funzione: $z = W \cdot x + b$ seguito da attivazione $\sigma(z)$
- Una rete con 100 layer ha **100 composizioni** di funzioni
- Durante il **training**, dobbiamo calcolare:
  1. **Predizione**: calcolare l'output y (passando attraverso tutti i layer)
  2. **Gradienti**: calcolare le derivate per aggiornare i pesi

Senza memorizzazione: dovremmo ricalcolare l'output di ogni layer → impossibile!

Con memorizzazione: calcoliamo ogni layer una volta, salviamo, riusiamo → efficiente!

9.2 Analogia Diretta

Matematica (Chain Rule)	**Deep Learning (Backpropagation)**
Funzioni g_1, g_2, g_3	Layer della rete (Dense, Conv, etc.)
Valori intermedi u_1, u_2	Attivazioni di ogni layer
Calcolo di y	Forward propagation
Calcolo di $f'(x)$ con chain rule	Backpropagation (calcolo gradienti)
Memorizzazione valori intermedi	Cache delle attivazioni

---

10 Implementazione Python

10.1 Codice Didattico

```python
import numpy as np

# Definizione delle funzioni e delle loro derivate
def g1(x):
    return x**2 + 1

def g1_prime(x):
    return 2 * x

def g2(u):
    return np.sin(u)

def g2_prime(u):
    return np.cos(u)

def g3(v):
    return np.exp(v)

def g3_prime(v):
    return np.exp(v)

# FORWARD PASS - Calcolo di y e salvataggio valori intermedi
def forward_pass(x):
    cache = {}  # Dizionario per salvare i valori
    cache['x'] = x
    cache['u1'] = g1(x)
    cache['u2'] = g2(cache['u1'])
    cache['y'] = g3(cache['u2'])
    return cache['y'], cache

# Calcolo della derivata usando i valori salvati
def compute_derivative(cache):
    g1_p = g1_prime(cache['x'])    # Usa x salvato!
    g2_p = g2_prime(cache['u1'])   # Usa u1 salvato!
    g3_p = g3_prime(cache['u2'])   # Usa u2 salvato!
    return g1_p * g2_p * g3_p

# ESEMPIO
x0 = 0.5
y, cache = forward_pass(x0)
dy_dx = compute_derivative(cache)

print(f"f({x0}) = {y:.4f}")
print(f"f'({x0}) = {dy_dx:.4f}")
```

---

11 Esercizi Proposti

11.1 Esercizio 1: Calcolo Manuale

Data la funzione: $y = \cos(e^x)$ con $x = 0$

1. Scomponi in $g_1(x) = e^x$ e $g_2(u) = \cos(u)$
2. Calcola la funzione (trova y)
3. Calcola la derivata (trova y')
4. Verifica che non stai ricalcolando valori

11.2 Esercizio 2: Implementazione Python

Implementa il calcolo per:

$$y = \sqrt{x^3 + 2x} \tag{13}$$

Valuta in $x = 2$.

11.3 Esercizio 3: Funzione a 4 Layer

Data la funzione:

$$y = \ln\Big(\sin\Big(\sqrt{x^2 + 1}\Big)\Big) \tag{14}$$

1. Identifica i 4 layer (g_1, g_2, g_3, g_4)
2. Costruisci la tabella del calcolo per $x = 1$
3. Calcola $y'(1)$

---

12 Domande di Verifica

1. **Perché salvare i valori intermedi durante il calcolo della funzione?**
   - Risposta: Per evitare di ricalcolarli durante il calcolo della derivata
2. **Cosa succederebbe se non salvassimo nulla?**
   - Risposta: Dovremmo ricalcolare $g_1(x)$, $g_2(g_1(x))$, ecc. ogni volta $\rightarrow$ molto inefficiente
3. **Quante volte valutiamo ogni funzione con l'approccio ottimizzato?**
   - Risposta: Esattamente 1 volta
4. **Perché questo è cruciale per le reti neurali?**
   - Risposta: Le reti hanno centinaia di layer $\rightarrow$ senza ottimizzazione, training impossibile

---

13 Conclusione

Concetto chiave: La chain rule + memorizzazione dei valori intermedi = algoritmo efficiente!

Strategia vincente:
1. **Calcola la funzione**: Passa attraverso tutti i layer, **memorizza ogni risultato intermedio**
2. **Calcola la derivata**: Applica la chain rule, **riusa i valori già calcolati**

Perché funziona:
- Le derivate $g_{1'}(x)$, $g_{2'}(u_1)$, $g_{3'}(u_2)$ hanno bisogno degli stessi valori che abbiamo già calcolato!
- $g_{3'}(u_2)$ richiede u_2 → già in memoria ✓
- $g_{2'}(u_1)$ richiede u_1 → già in memoria ✓
- $g_{1'}(x)$ richiede x → già in memoria ✓

Impatto:
- Evita ricalcoli ridondanti (50%+ risparmio con molti layer)
- Rende possibile il training di reti profonde
- Fondamento della backpropagation

Machine Learning = Matematica + Ottimizzazione Algoritmica

---

14 Risorse Extra

14.1 Video Consigliati

- 3Blue1Brown: «Backpropagation calculus» (visualizzazione eccellente)
- StatQuest: «Backpropagation Step by Step»

14.2 Letture

- «Deep Learning» (Goodfellow et al.) - Chapter 6: Backpropagation
- «Neural Networks and Deep Learning» (Nielsen) - Online free book

14.3 Tool Interattivi

- TensorFlow Playground: visualizza forward/backward in tempo reale
- Desmos: grafica le funzioni composte passo per passo