Differenziazione automatica

Opzioni: 1) derivate simboliche: uso calcolo simbolico / formule

2) Derivate numeriche: scelgo $h>0$, approssimo $f'(x)$ con $\tilde{f}(x) = \dfrac{f(x+h)-f(x)}{h}$

Due fonti di errore

a) $\tilde{g}$ non è esattamente $f'(x)$:

$$f(x+h) = f(x) + f'(x)\cdot h + \frac{1}{2}f''(\xi)h^2$$

$$\frac{f(x+h)-f(x)}{h} - f'(x) = \frac{1}{2}f''(\xi)h$$

b) aritmetica di macchina in $f(x+h)$, $f(x)$

Anche con $\tilde{g}$ perfetto $\qquad |\delta_i| \leq u$ prec. macchina

$$\frac{\left(f(x+h)(\cancel{1}+\delta_1) - f(x)(\cancel{1}+\delta_2)\right)}{h}(1+\delta_3)(1+\delta_4) - \cancel{g}$$

$$\doteq \frac{f(x+h)}{h}(\delta_1+\delta_2+\delta_3) + \frac{f(x)}{h}(\delta_2+\delta_3+\delta_4) + O(u^2)$$

$$= \left(\left|\frac{f(x+h)}{h}\right| + \left|\frac{f(x)}{h}\right|\right)u$$

Errore globale, $\qquad u \approx 10^{-16}$

$$|\tilde{g} - f'(x)| \leq \underbrace{\left|\frac{1}{2}f''(\xi)\right|}_{=O(1)} \cdot h + \underbrace{\left(|f(x+h)| + |f(x)|\right)}_{=O(1)} \cdot \frac{u}{h}$$

$h = 10^{-4}$ $\qquad 10^{-4}$ $\qquad 10^{-12}$

$h = 10^{-8}$ $\qquad 10^{-8}$ $\qquad 10^{-8}$ $\leftarrow h = O(u^{1/2})$

$h = 10^{-12}$ $\qquad 10^{-12}$ $\qquad 10^{-4}$

Più formalmente,
$$\frac{\left|\frac{1}{2}f''(\xi)\right|h + \left(|f(x+h)| + |f(x)|\right)\cdot\frac{u}{h}}{2} \geq \sqrt{\left|\frac{1}{2}f''(\xi)\right|\left(|f(x+h)| + |f(x)|\right) h\cdot\frac{u}{h}}$$

$= C \cdot u^{1/2}$, con uguaglianza se $\left|\frac{1}{2} f''(\xi)\right| h = \left(|f(x+h)| + |f(x)|\right) \frac{u}{h}$

Complex-step: suppongo f olomorfa, e di avere codice per calcolarla anche con input complessi

Uso incremento ih immag. puro

$$f(x+ih) = f(x) + f'(x) \cdot ih - \frac{1}{2} f''(x) h^2 + O(h^3)$$

$$\underbrace{\mathrm{Im}\left[\frac{f(x+ih)-f(x)}{h}\right]}_{d} - f'(x) = \cancel{\underbrace{\frac{1}{2} f''(x) h}} + O(h^2)$$

$$|\hat{d} - f'(x)| \le \underbrace{\left|\frac{1}{6} f'''(\xi)\right|}_{O(1)} h^2 + \underbrace{\left|\mathrm{Im}\, f(x+ih)\right|}_{O(1)} \frac{u}{h} + O(h^2) + O(u)$$

10^{-4}	10^{-8} +	10^{-16}
10^{-8}	10^{-16} +	10^{-16}
10^{-12}	10^{-24} +	10^{-16}

$$f(x+ih) = \mathrm{Re}\, f(x+ih)^{(1+\delta_r)} + \mathrm{Im}\, f(x+ih)(1+\delta_i)$$

$$\mathrm{Im}\, f(x+ih)(1+\delta_i)$$

tipicamente piccola, dell'ordine di h

e anche calcolata (solitamente) con quantità dell'ordine di h esplicitamente,

$$(x+ih)(x+ih) = (x^2 - h^2) + i(x \cdot h + h \cdot x)$$

Codice che funziona su tipi più generali

$\Rightarrow$ errore minore.

Se f funziona anche su matrici, cosa che richiede piccoli cambiamenti, ad es.

```
z = x * x              n = size(x, 1)
w = x + 5      -->     z = x * x
y = z * w              w = x + 5 * eye(n)
                       y = z * w
```

Funziona anche con loop, funzioni definite su matrici come l'esponenziale (expm), ...

Questo, in pratica, è propagare sviluppi di Taylor:

invece di

$z = x * x$

$w = x + 5$

$y = z * w$

$x = 5$	
$z = 25$	
$w = 10$	
$y = 250$	

calcolo su sviluppi di Taylor di una perturb.

$$x = 5 + 1\cdot\varepsilon = 5 + 1\cdot\varepsilon + 0\cdot\varepsilon^2 + O(\varepsilon^3)$$

$$z = (5 + 1\cdot\varepsilon + 0\cdot\varepsilon^2 + O(\varepsilon^3))(5 + 1\cdot\varepsilon + 0\cdot\varepsilon^2 + O(\varepsilon^3)) = $$
$$= 25 + 10\cdot\varepsilon + 1\cdot\varepsilon^2 + O(\varepsilon^3)$$

$$w = 5 + 1\cdot\varepsilon + 0\cdot\varepsilon^2 + O(\varepsilon^3) + 5 = 10 + 1\cdot\varepsilon + 0\cdot\varepsilon^2 + O(\varepsilon^3)$$

$$y = (25 + 10\varepsilon + 1\cdot\varepsilon^2 + O(\varepsilon^3)) * (10 + 1\cdot\varepsilon + 0\,\varepsilon^2 + O(\varepsilon^3))$$

$$= \underset{f(5)}{250} + \underset{f'(5)}{125}\,\varepsilon + \underset{f''(5)}{20}\,\varepsilon^2 + O(\varepsilon^3)$$

$$\begin{bmatrix} 25 & 10 & 1 \\ 0 & 25 & 0 \\ 0 & 0 & 25 \end{bmatrix} \cdot \begin{bmatrix} 10 & 1 & 0 \\ 0 & 10 & 1 \\ 0 & 0 & 10 \end{bmatrix}$$

$x = \text{Taylor}[5\ 1\ 0]$

$z = x * x = \text{Taylor}[5\ 1\ 0] * \text{Taylor}[5\ 1\ 0] = \text{Taylor}[25\ 10\ 1]$

$w = x + 5 = \text{Taylor}[5\ 1\ 0] + 5 = \text{Taylor}[10\ 0\ 0]$

$y = \text{Taylor}[\ ..\ ] * \text{Taylor}[\ ..\ ]$

<u>Regola</u>: $\text{Taylor}[a_1\ a_2\ a_3] * \text{Taylor}[b_1\ b_2\ b_3]$
$= \text{Taylor}[a_1 b_1,\ a_1 b_2 + a_2 b_1,\ a_1 b_3 + a_2 b_2 + a_3 b_1]$

<u>Regola</u>: $\text{Taylor}[a_1, a_2, a_3] + \text{Taylor}[b_1\ b_2\ b_3] = \text{Taylor}[a_1 + a_2, b_1 + b_2, a_3 + b_3]$

<u>Regola</u>: a si può convertire in $\text{Taylor}[a, 0, 0]$

Nello stesso modo, uno può definire $\text{Taylor}[a] / \text{Taylor}[b]$

$\exp(\text{Taylor}[a]), \dots$

$$\exp(a_0 + \varepsilon a_1 + \varepsilon^2 a_2) = 1 + (a_0 + \varepsilon a_1 + \varepsilon_2 a_2) + O(\varepsilon^3) + \frac{1}{2}(a + \varepsilon_1 a_1 + \varepsilon_2 a_2)^2 + O(\varepsilon^3) + O(\varepsilon^3) + \dots$$

In generale, per una certa op. elementare $z = \varphi(a, b, \dots)$

$$z' = \boxed{\frac{\partial \varphi}{\partial a}(a,b)\, a'} + \frac{\partial \varphi}{\partial b}(a,b) \cdot b'$$

$$z'' = \left(\frac{\partial^2 \varphi}{\partial a^2}(a,b) a' + \frac{\partial^2 \varphi}{\partial a \partial b}(a,b) b' \right) a' + \frac{\partial \varphi}{\partial a}(a,b) \cdot a'' + \text{(der. secondo termine..)}$$

Caso più frequente: 1 derivata sola, $[a_1, a_2] \rightsquigarrow a_1 + \varepsilon a_2$

"dual numbers": potete pensare a queste operazioni come definite su un anello $\mathbb{R}[\varepsilon] / (\varepsilon^2)$

$$(a_1 + a_2 \varepsilon)(b_1 + b_2 \varepsilon) = a_1 b_1 + (a_1 b_2 + a_2 b_1)\varepsilon + \cancel{a_2 b_2 \varepsilon^2}$$

(Solo un altro formalismo per le stesse operazioni)

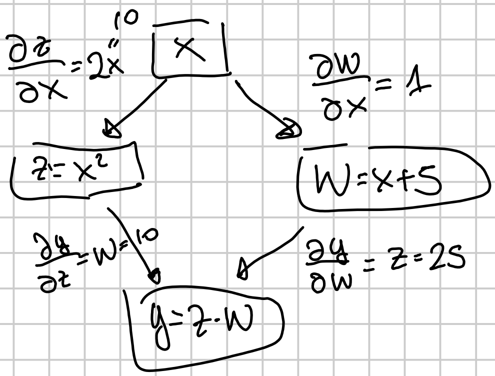
La tecnica usata finora è nota come "forward mode" della "automatic differentiation"

C'è anche un "reverse mode", "back-propagation"

Idea: se il forward mode (per una derivata sola) parte da $\frac{dx}{dx} = 1$, e calcola $\frac{da}{dx}$ per ogni quantità calcolata che compare nel codice $\frac{dz}{dx}, \frac{dw}{dx}, \frac{dy}{dx} \dots$,

il reverse mode parte dall'output y, con $\frac{\partial y}{\partial y} = 1$, e calcola $\frac{\partial y}{\partial a}$ per ogni quantità calcolata a che compare nel codice, a partire dall'ultima.

Di solito, tramite grafi: $\boxed{x=5}$



Posso calcolare (dal fondo) derivate parziali di y rispetto a qualunque cosa:

$$\frac{\partial y}{\partial z} = w = 10 \qquad \frac{\partial y}{\partial w} = z = 25$$

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial w} \cdot \frac{\partial w}{\partial x} + \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial x} = 25 \cdot 1 + 10 \cdot 10 = 125$$

Si può costruire algoritmicamente via trasformazioni del codice, ma è più complicato del forward mode per cui basta definire un tipo.

Qual è il vantaggio del reverse mode? Per $f:\mathbb{R}\to\mathbb{R}$, nessuno.

Per $f:\mathbb{R}^m\to\mathbb{R}^n$ di cui voglio tutte le derivate (Jacobiano),
il 'forward mode' richiede meno operazioni se $n >> m$
il 'reverse mode' richiede meno operazioni se $m >> n$

Infatti, se $f:\mathbb{R}\to\mathbb{R}^n$, (nel forward mode) mi basta usare "sviluppi di Taylor per vettori,"

cioè, $\begin{bmatrix}1\\2\\3\end{bmatrix} \longrightarrow \begin{bmatrix}1\\2\\3\end{bmatrix} + \varepsilon \begin{bmatrix}4\\5\\6\end{bmatrix} = \mathrm{Taylor}\left(\begin{bmatrix}1\\2\\3\end{bmatrix}, \begin{bmatrix}4\\5\\6\end{bmatrix}\right)$

Se ho tanti input, devo ripetere il calcolo tante volte considerando diverse perturbazioni: $X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$, derivo rispetto a x_1 prima,

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} + \varepsilon \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \text{poi } x_2, \quad \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} + \varepsilon \begin{bmatrix} 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \ldots \quad \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} + \varepsilon \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 1 \end{bmatrix}$$

Machine Learning (ad alto livello): fittare funzioni con molti input con tecniche di ottimizzazione tipo gradient descent, $X_{k+1} \leftarrow X_k - \frac{\partial f}{\partial X} \cdot \delta_k$