Lez_11_ RetiNeuraliConvoluzionali

November 16, 2023

1 Lezione 11 - Reti Neurali Convoluzionali

Quando abbiamo lavorato sul database di *Fashion MINST*, il nostro obiettivo era classificare gli oggetti nelle immagini. Ogni immagine è griglia bidimensionale di colori. Noi non ci siamo curati della griglia, perchè abbiamo usato un vettore di pixel, ovvero mettendo una riga dopo l'altra, e dato in pasto alla rete. Potevamo mescolare gli elementi, magari prendendo i pixel in modo sparso. La rete avrebbe funzionato nello stesso modo, perchè non c'è differenza tra le unità di input. Vengono sommati input livelli precedenti con un peso, ma non c'è differenza nel come prendo i pixel. Tuttavia, visto dall'occhio umano, la posizione dei pixel è fondamentale per capire cosa abbiamo davanti. Quindi, forse, potremmo sfruttare questo aspetto per migliorare i risultati.

Appiattire non è quindi una soluzione sempre buona. Supponiamo di classificare *cani* e *gatti*. Abbiamo immagini già etichettate, e supponiamo siano 1 mega pixel, cioè *1 milione di pixel*. Sono foto con poca risoluzione. Diamo queste cose ad una rete neurale, dopo il livello di input, mettiamo un livello nascosto completamente connesso. Potremmo mettere un milione di unità, che sarebbe però esagerato. Proviamo con 1000 unità. Abbiamo peso per ogni interconnessione del livello nascosto, quindi $10^6 \cdot 10^3 = 10^9$, sembra un po' costoso. (il primo termine è il milione di pixel, il secondo le unità)

1.1 Introduzione alle CNN

Il problema originale era riconoscere il CAP scritto a mano. Sono dati aventi una struttura ben definita, come serie temporali. La differenza rispetto i normali vettori di input risulta nell'ordine, infatti nella serie *temporale* il tempo ha una certa rilevanza. Nel nostro esempio, la posizione dei pixel ha un ruolo ben preciso. Sono dette *convoluzionali*, da *convoluzione*, usata nelle reti neurali. Non è l'unica operazione eseguita, basta un livello di convoluzione per parlare di questo tipo di reti.

1.1.1 Cosa è la convoluzione

La convoluzione tra due funzioni reali x(t) e w(t) è definita come:

$s(t) = (x * w)(t) = \int_{-\infty}^{\infty} x(u)w(t-u)du$

Quando argomento prima funzione cresce, la seconda funzione decresce.

Nel caso discreto, si lavora con sommatoria: $s(t) = (x * w)(t) = \sum_{i=-\infty}^{\infty} x(i)w(t-i)$

Esempio Navicella spaziale, prendiamo la posizione ad ogni istante t mediante un laser. Può avere interferenze. Per irrobustire il tutto, facciamo media ultime misure. Vorrei dare importanza alle misure più recenti, perchè la navicella si sposta. Tale peso è espresso $w(a)$, se a cresce, quindi dato vecchio, allora il peso decresce. s è la stima della posizione al tempo t, dove rapportiamo

$t - u$, ovvero il tempo attuale e quello in cui abbiamo preso la misura. Questo rapportandoci alla formula della convoluzione nel caso continuo.

1.2 Convoluzione coi tensori

Lavoriamo con array multidimensionali. Lavoriamo con dati discreti, non continui. La convoluzione è tra due tensori, il primo è l'input del livello (o della rete), il secondo è detto *kernel/feature map*. Quest'ultimo è parametro livello di convoluzione. La formula in figura unidimensionale è:

$s(t) = (x * w)(t) = \sum_{i=-\infty}^{\infty} x(i)w(t-i)$

noi la riadattiamo per lavorare con matrici:

$(X * K)(i,j) = \sum_a \sum_b X(a,b)K(i-a, j-b)$

La logica non cambia, lo facciamo sia sulle righe sia sulle colonne.

1.3 Costruzione livello nascosto

Un livello convoluzionale prende un tensore X in input e produce un output in H (tramite convoluzione). Possiamo anche applicare funzioni di attivazione come `relu`. Il kernel K è fatto da parametri calcolati durante il training, è scomparsa la matrice w, che mi pesava le connessioni tra le unità di input ad ogni unità del livello nascosto. Il passaggio da input a livello nascosto avviene tramite il kernel.

1.4 Correlazione vs Cross-Correlation

Possiamo invertire il prodotto *kernel* ed *input*. La Convoluzione è **commutativa**.

$\sum_a \sum_b X(a,b)K(i-a, j-b) = \sum_a \sum_b K(a,b)X(i-a, j-b)$

Nelle librerie Python si usa la versione la *cross correlation*, in cui lavoriamo con segni + invece che segni −.

$S(i,j) = \sum_a \sum_b K(a,b)X(i+a, j+b)$

E' un problema? No, il kernel è fatto dai parametri del livello, che impariamo durante il training, quindi al massimo cambiano i parametri. A noi tanto interessa il risultato.

1.4.1 Esempio cross-correlation

Input I

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 2 & 5 & 0 & 0 \\ 7 & 8 & 9 & 1 \end{bmatrix}$$

Kernel K

$$\begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

$$(K * I)(1,1) = a + 2b + 2c + 5d$$

$$(K * I)(1,2) = 2a + 3b + 5c$$

Esempio 1

Input X

$$\begin{bmatrix} 0 & 1 & 2 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix}$$

Kernel K

$$\begin{bmatrix} 0 & 1 \\ 2 & 3 \end{bmatrix}$$

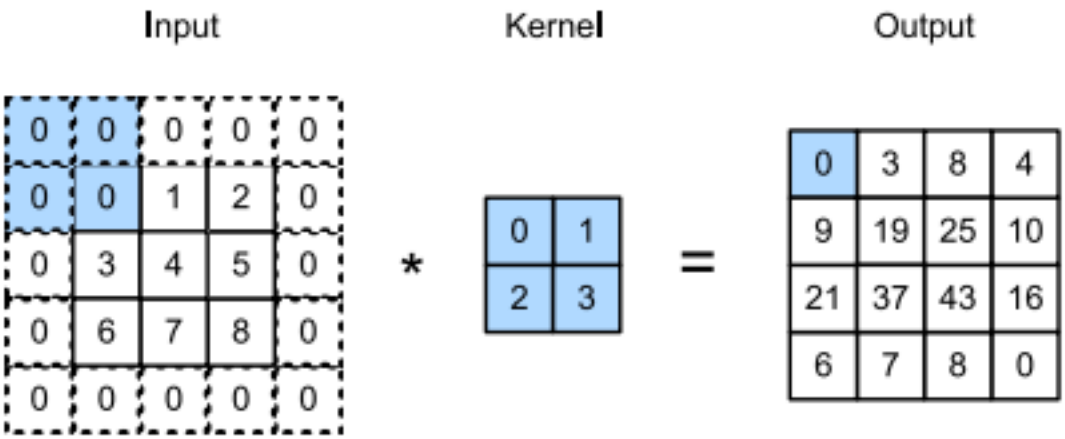
H

$$\begin{bmatrix} 19 & 25 \\ 37 & 43 \end{bmatrix}$$

Esempio 2 Ciò che notiamo è che siamo partiti da input 3x3, e siamo arrivati ad output 2x2. Il kernel non può scorrere nè verso il basso, nè verso destra, non potevo andare oltre! Con N livelli convoluzionali? L'input si andrebbe sempre a restringere, non è ideale. Soluzione?

1.5 Zero Padding

Aggiungiamo pixel, di solito nulli, per aumentarne le dimensioni.

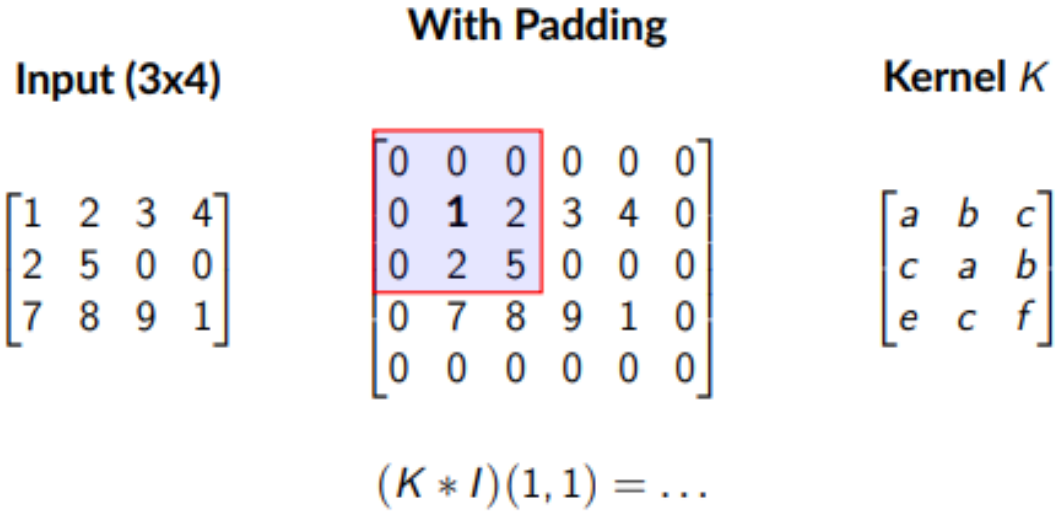


L'input esteso tiene conto di questi valori nulli. Le dimensioni dell'output sono:

- Con input di dimensione n_h x n_w e kernel di dimensione k_h x k_w, l'output ha dimensione $(n_h - k_h + 1)$ x $(n_w - k_w + 1)$
- Se aggiungiamo p_h righe e p_w colonne di padding, si avrà $(n_h + p_h - k_h + 1)$ x $(n_w + p_w - k_w + 1)$
- Per avere input di dimensione come l'output, settiamo $p_h = k_h - 1$ e $p_w = k_w - 1$, per mantenere le dimensioni originali.

Se kernel ha stesse dimensioni dell'input, posso sovrapporlo solo una volta

Nota: Una best practice, richiede che i kernel abbiano dimensioni *dispari*, in tal caso aggiungo all'input un numero di righe e colonne *pari*, cioè metà sopra e metà sotto, metà a destra e metà a sinistra… è simmetrica. Come beneficio di conseguenza, l'output $H_{i,j}$ è calcolato prendendo in modo centrato sull'input originale $X_{i,j}$



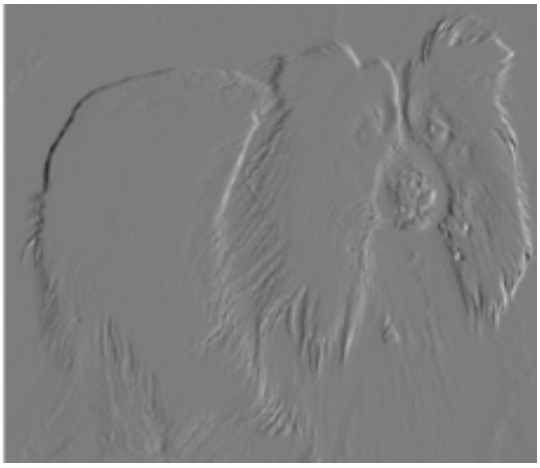
2 Perchè la convoluzione

Concetti interessanti sono:

- **Interazioni sparse**: ogni unità di output può interagire con ogni unità di input (a meno di pesi nulli). Nelle reti convoluzionali si sceglie *kernel* più piccolo dell'input, così come lo sarà il numero di parametri.
- Condivisione dei parametri
- Rappresentazioni equivarianti

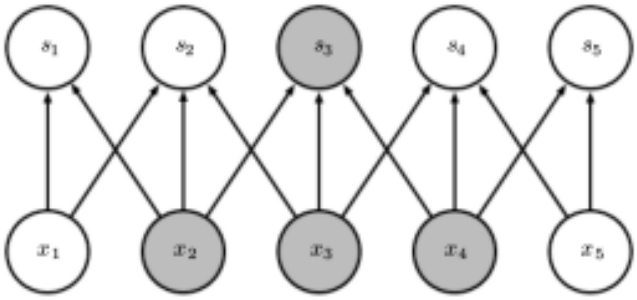
2.1 Esempio detecting vertical edges

Anche con $K = [1, -1]$, viene fatta sottrazione di due pixel adiacenti. Nelle regioni uguali, la differenza è 0, quindi convoluzione non produce alcuna uscita. Con differenza di colori, abbiamo livello convoluzionale che ci dice quali sono i bordi dell'immagine.

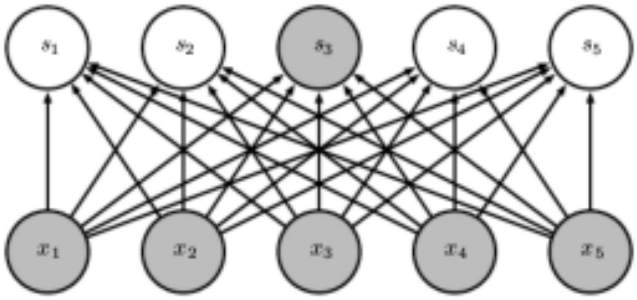


2.2 Receptive Field

Nella convoluzione, per un valore di uscita di un'unità di output (colore grigio), tocchiamo 3 pixel del kernel. Nel caso fully connected, tocchiamo tutto.



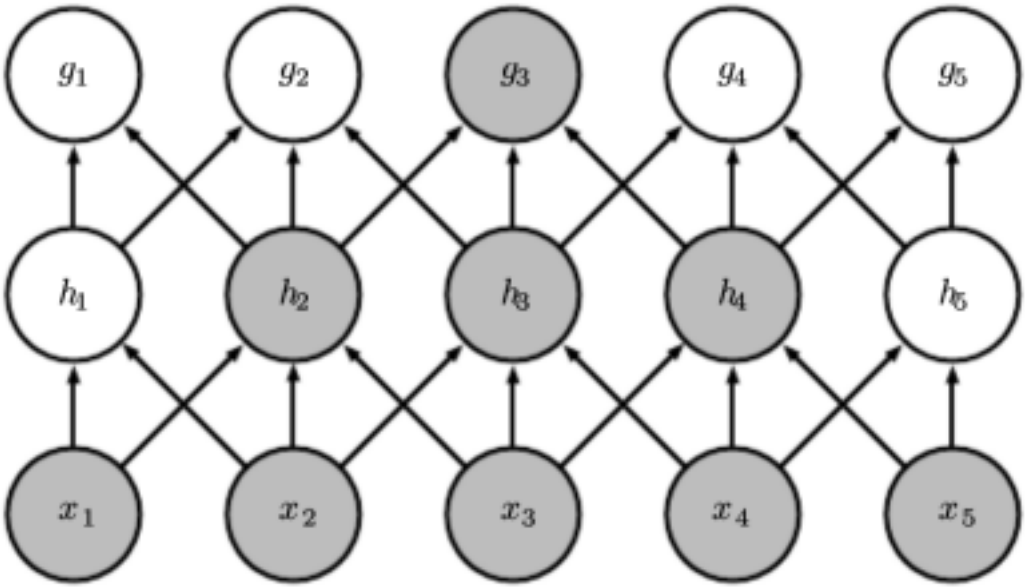
Convolutional
(3-pixel kernel)



Fully Connected

2.3 Con più livelli

Sembra che nella rete convoluzionale non si riesca più a catturare aspetti interessanti per noi. Con più livelli, risolviamo tale problema, toccando più parti dell'immagine. Meno interconnessioni (-parametri), ma non perdiamo capacità di andare a tenere conto di tutti gli aspetti dell'input. A ciò si collega altro concetto di **Parameter Sharing**.



2.4 Parameter Sharing

In una rete neurale tradizionale, ogni elemento della matrice dei pesi viene utilizzato esattamente una volta nel calcolo dell'output di uno strato.

Nelle CNN (quindi reti convoluzionali), gli stessi elementi del kernel vengono utilizzati in ogni posizione dell'input. Quindi gli stessi parametri sono *condivisi* tra le unità di output. Cioè, il Kernel è sempre lo stesso per tutte le posizioni delle immagini, cioè sono questi gli unici parametri che devono essere memorizzati. L'input può essere grande quanto vuole, facendo così ottimizziamo memoria e parametri.

La complessità della retropropagazione non cambia, mentre otteniamo importanti benefici, tra cui requisiti di memoria ridotti, oltre al numero ridotto di parametri da apprendere.

L'algoritmo di back propagation è sempre lo stesso, solo che i parametri entrano più volte in gioco.