Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE

Corso di Laurea in Matematica

TOPOLOGIE NON CONVENZIONALI PER RETI DI NEURONI ARTIFICIALI

Relatore:
Prof.
MASSIMO FERRI

Presentata da:
FEDERICO CILIEGI

V Sessione
Anno Accademico 2018/2019

Introduzione

Le reti neurali sono uno strumento informatico che si è progressivamente affermato dalla sua nascita, e si è distinto per le sue grandi potenzialità. Pur non essendo ancora presente una teoria matematica formale che le descriva, esse sono state approfonditamente studiate da ingegneri e informatici, rendendo il loro studio una branca tanto ampia quanto preziosa.

In questa tesi presento alcuni modelli di rete neurale e ne illustro le caratteristiche a livello di topologia, elaborazione e addestramento.
Nella prima sezione illustro le motivazioni dello studio delle reti neurali e il tipo di problemi che queste possono affrontare, distinguendosi da altri modelli computazionali.
Nella seconda sezione descrivo il neurone artificiale, mattone fondamentale di tutta la teoria. Di questo semplice modello computazionale fornisco due possibili definizioni, una più formale e una più pratica per le applicazioni, e illustro le più semplici funzioni di attivazione.
Nella terza sezione mostro quali sono le classi principali di reti neurali, distinguendone le topologie e le regole di propagazione, caratteristiche fondamentali dei diversi tipi di architettura.
Nella quarta e ultima sezione descrivo come diversi tipi di reti e neuroni artificiali possono essere addestrati, fornendo gli algoritmi per cambiare i pesi dei collegamenti in modo efficace e rendere le reti capaci di affrontare una ampia classe di problemi.

Indice

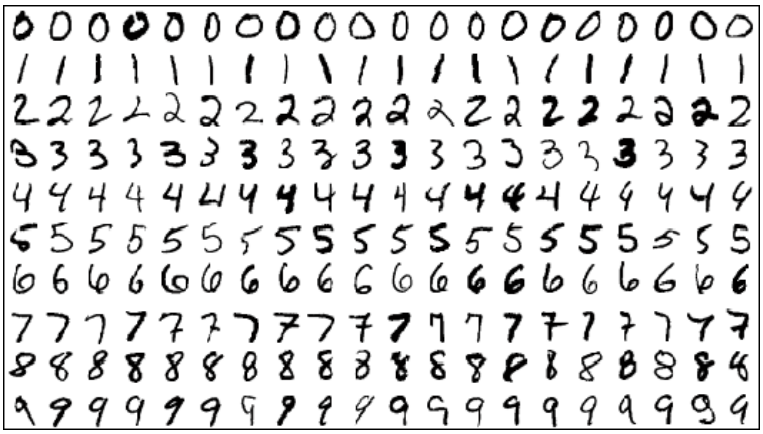
Capitolo 1

Le reti neurali

Le reti neurali sono strumenti informatici nati nel secolo scorso di riconosciuta validità, che hanno mostrato di poter svolgere compiti tradizionalmente ritenuti risolubili solo da esseri umani, e non da macchine, costituendo un ambito di studi fondamentale per l'intelligenza artificiale.

1.1 Le motivazioni

I computer si sono dimostrati, nel corso della storia e sin dalla loro nascita, strumenti potenti per la risoluzione di ogni tipo di problema: anche i primi calcolatori, oggi ritenuti lentissimi, erano in grado di effettuare calcoli, pur semplici, a velocità inarrivabili per un essere umano, e la programmazione si è evoluta tanto da fornire algoritmi per la risoluzione di una ampia classe di problemi tramite le sole operazioni che un computer è in grado di svolgere. Ciò nondimeno, più complicato è il problema, più difficile è scrivere un programma che lo risolva. Ci sono compiti molto ardui, se non impossibili, da svolgere tramite la programmazione tradizionale, ma che per gli esseri umani sono del tutto banali: si prenda come esempio il riconoscimento di cifre e testi, la successiva lettura di questi testi, la traduzione da una lingua ad un'altra, o il riconoscimento di un viso. La difficoltà nello scrivere un programma che, per esempio, riconosca le cifre o le lettere, sta nel fatto che ci sono molti modi diversi di scrivere lo stesso simbolo.



Dove la programmazione tradizionale non arriva, dunque, occorrono altre strategie per risolvere questi problemi. Oggi ce ne sono diverse, e alcune di esse sfruttano lo strumento delle reti neurali, o reti di neuroni artificiali.

L'idea alla base di questi strumenti è di simulare l'attività di un cervello umano partendo dalle sue più semplici parti, ovvero i neuroni. Questi sono unità che svolgono operazioni molto semplici, individualmente, ma si organizzano in complessi collegamenti che scambiano informazioni sino a portare a forme di elaborazione molto più articolate, e dunque con migliori potenzialità.

1.2 Il neurone artificiale

Il concetto di neurone artificiale nasce quindi come oggetto di calcolo semplice. Non essendo ancora lo studio delle reti neurali artificiali una branca matematica formalizzata, esistono diverse definizioni di neurone artificiale. Una possibilità molto generale e astratta è la seguente.

Definizione 1 (Neurone artificiale a tempo discreto)**.** Dati un insieme di segnali X, un insieme di stati S e un insieme di output Y, delle sequenze temporali di input $x_1(t), \ldots, x_n(t)$, $t \in \mathbb{N}$, una funzione detta funzione di prossimo stato

$$f : X^n \longrightarrow S, s(t) = f(x_1(t-1), \ldots, x_n(t-1), s(t-1))$$

e una funzione detta di output

$$g : S \longrightarrow Y, y(t) = g(s(t))$$

un neurone artificiale è la funzione che alle sequenze temporali $x_1, \ldots, x_n$ associa la sequenza y.

Questa è solo una possibile definizione formale di neurone artificiale, ma è molto astratta. Al momento della sua creazione il neurone artificiale era un modello computazionale molto più semplificato, e tutt'oggi la definizione più usata in ambito informatico è assai più specifica.

Definizione 2 (Neurone artificiale)**.** Dati un vettore detto di pesi $w \in \mathbb{R}^n$, un numero reale $\theta \in \mathbb{R}$ detto bias (o soglia) e una funzione $\mathcal{F} : \mathbb{R} \longrightarrow \mathbb{R}$ detta funzione di attivazione, il neurone artificiale con pesi w, bias (o soglia) θ e funzione di attivazione $\mathcal{F}$ è la funzione

$$x \in \mathbb{R}^n, x \longmapsto \mathcal{F}(< x, w > -\theta).$$

Dunque la definizione di neurone artificiale a tempo discreto viene applicata con $X = S = Y = \mathbb{R}$, come funzione di prossimo stato si sceglie una funzione affine di $x_1, \ldots, x_n$, che possono ora essere viste come successioni reali costanti; viene lasciata una certa libertà circa la funzione di output, chiamata comunemente funzione di attivazione in analogia col caso biologico, in cui i neuroni non si attivano, ovvero non emettono ulteriori segnali, se la carica dei messaggi ricevuti non supera un certo valore (detto appunto soglia).
La prima funzione di attivazione tradizionalmente utilizzata fu la funzione di Heaviside, con output binario:

$$H(x) = \chi_{\mathbb{R}^+ \cup \{0\}}(x) = \begin{cases} 1 & \text{se } x \geq 0 \\ 0 & \text{se } x < 0 \end{cases}$$

Questa funzione è particolarmente utile per compiti di classificazione. In alternativa a questa è stata usata un'altra funzione a gradino, la funzione

segno, con output bipolare:

$$sgn(x) = \begin{cases} 1 & \text{se } x \geq 0 \\ -1 & \text{se } x < 0 \end{cases}$$

Un'altra scelta classica come funzione di attivazione è la sigmoide generica. Dati dei parametri reali $a, b, c, d \in \mathbb{R}$, la funzione sigmoide con questi parametri è

$$\begin{aligned} \sigma : & \mathbb{R} \longrightarrow \mathbb{R} \\ & x \longmapsto \frac{a}{1 + e^{b \cdot x + c}} + d \end{aligned}$$

Con la scelta di $a = 1, b = 1, c = 0, d = 0$ si ottiene la funzione sigmoide semplice, con la scelta di $a = 2, b = 2, c = 0, d = -1$ si ottiene la tangente iperbolica, e queste sono due scelte comuni perché seguono l'andamento rispettivamente della funzione di Heaviside e della funzione segno.
Ultimamente sono molto utilizzate anche la funzione ReLu (per *Rectified Linear unit*), che mappa $x \in \mathbb{R}$ in $max\{0, x\}$, e alcune sue semplici variazioni.

La scelta della funzione di attivazione comporta diversi vantaggi e svantaggi in sede di elaborazione, e poi di addestramento. Infatti lo scopo del neurone artificiale è l'addestramento.
Il neurone artificiale dipende dalla scelta di $n + 1$ parametri reali, il vettore di pesi w e la soglia θ. L'idea dietro l'utilizzo di questo modello nell'apprendimento supervisionato è la ricerca, tramite un'opportuno algoritmo, di pesi e soglia ottimali per approssimare una funzione di cui si conoscono alcuni esempi, ovvero si conoscono solo alcune associazioni input-output e da questi si vuole dedurre i valori assunti dalla funzione su punti la cui immagine non è nota a priori.

1.3 Le reti neurali

Il modello del neurone artificiale riscosse un notevole successo al momento della sua creazione, mostrandosi efficace in diversi compiti di classificazione. Manifestò presto, però, anche degli importanti limiti: un singolo neurone

con funzione di attivazione H, per esempio, può simulare le funzioni logiche AND, OR e NOT, ma non può in alcun modo simulare la funzione XOR. Può, in effetti, risolvere solo problemi di classificazione di insiemi linearmente separabili. Per ovviare a questo problema si possono adoperare più neuroni artificiali collegati gli uni agli altri, cosicché i valori di input di alcuni neuroni siano i valori di output di alcuni altri: questa è una rete neurale.

La struttura di una rete neurale è ovviamente più complicata di quella di un singolo neurone. Si può immediatamente osservare che il numero di parametri aumenta, in quanto ogni singolo neurone nella rete ha un proprio vettore di pesi e un proprio bias. In particolare per ogni coppia di neuroni i, j c'è un peso (eventualmente nullo) che definisce la dipendenza del neurone j dal neurone i, il numero $w_{i,j} \in \mathbb{R}$. Questo fa sì che per una rete neurale si possa definire non più un vettore, ma una matrice di pesi W.

In questa rete alcuni neuroni saranno di input, ovvero ricevono dati dall'esterno, alcune saranno di output, cioè i valori di output di questi neuroni sono restituiti come output della rete, e altri potranno essere nascosti (indicati con h, da *hidden*), cioè ricevono informazioni solo da altri neuroni e i loro valori di output sono restituiti solo come input ad altri neuroni. I neuroni nascosti sono il corrispettivo artificiale di quelli che, nei sistemi nervosi biologici, vengono detti interneuroni.

In una rete neurale ci sono due caratteristiche fondamentali: una è la topologia, ovvero il grafo orientato associato alla rete in cui i nodi sono i neuroni artificiali e l'arco dal neurone i al neurone j è presente se il valore di output del neurone i viene usato come input per il neurone j; l'altra è la regola di propagazione, ovvero il modo in cui dall'input della rete e dai pesi si calcola un vettore di output.

Dal momento che lo scopo è trovare i pesi ottimali per approssimare una funzione di cui sono dati esempi, un'altra parte importante di una rete neurale è la cosiddetta regola di apprendimento, ovvero l'algoritmo con cui vengono scelte le modifiche progressive dei pesi in base agli esempi dati.

Le diverse topologie possono influenzare le regole di propagazione o di addestramento, e si suddividono in due macroclassi: feedforward e ricorrenti.

1.3.1 Le topologie feedforward

Una importante classe di topologie per le reti neurali è l'insieme delle topologie in cui non sono presenti cicli: queste sono dette topologie feedworward, perché le informazioni, in questo tipo di reti, scorrono in una sola direzione: dai neuroni di input ai neuroni di output. È uso comune dividere le reti di questo tipo in strati, e in questa suddivisione il primo strato è composto dai neuroni di input, il secondo strato è composto dai neuroni che ricevono dati solo dai neuroni del primo strato, il terzo strato riceve informazioni solo dal secondo, e così via fino allo strato di output.

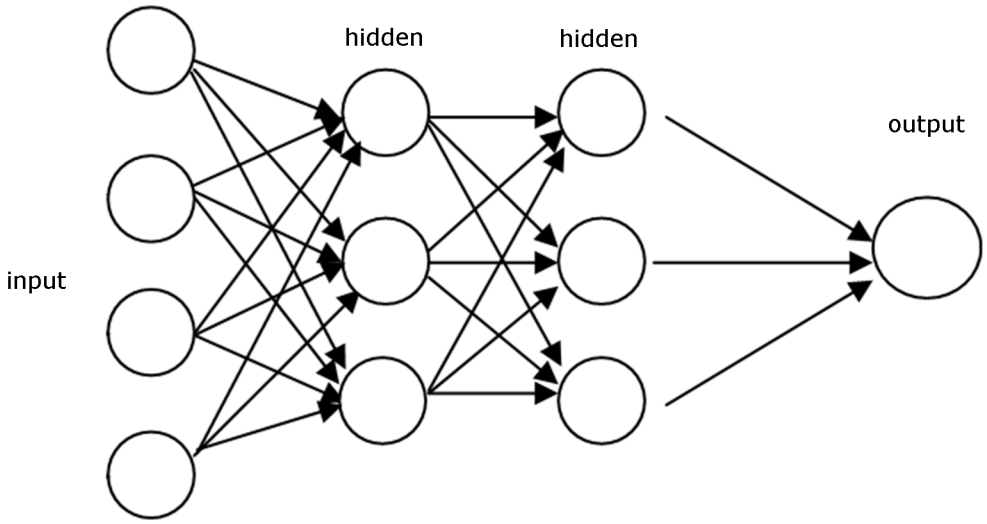


Figura 1.1: Esempio di rete feedforward

Il vantaggio di queste topologie è che si prestano a una scelta particolarmente intuitiva per la regola di propagazione: vengono innanzitutto calcolati i valori dei neuroni del primo strato nascosto, ovvero del secondo strato, e poi degli strati successivi in sequenza fino allo strato di output.
Le reti neurali feedforward non sono molto più complicate dei singoli neuroni artificiali dal punto di vista del calcolo, ma sono comunque uno strumento molto potente: si può infatti dimostrare che una rete neurale con un singolo strato nascosto, detta rete neurale "vanilla", le cui funzioni di attivazione

soddisfino alcune ipotesi, può approssimare una qualsiasi funzione continua $A \longrightarrow \mathbb{R}$ con $A \subset \mathbb{R}^n$ compatto. Questo risultato è noto come teorema di approssimazione universale, e dimostra un grande potere di rappresentazione di reti neurali molto semplici. Lo svantaggio di queste reti può essere nella pesante elaborazione, quando sia necessaria una grande quantità di neuroni nello strato nascosto, o nella lentezza dell'addestramento e del raggiungimento di un errore abbastanza piccolo rispetto alla funzione che si intende rappresentare.

1.3.2 Un caso particolare di topologia ricorrente: la rete di Hopfield

Le topologie che ammettono cicli vengono dette ricorrenti, e chiaramente ne esistono molteplici tipi. Un caso del tutto particolare è quello introdotto da Hopfield nel 1982 di una rete in cui tutti i collegamenti siano simmetrici, ovvero se il neurone i è collegato al neurone j allora il neurone j è collegato al neurone i, e $w_{i,j} = w_{j,i}$. In sostanza, si richiede che la matrice dei pesi sia simmetrica. Per le reti neurali ricorrenti, come questa, la scelta di una regola di propagazione non è scontata come nel caso feedforward. Hopfield ne propose una nel caso di reti con questa topologia, in cui ogni neurone sia sia di input che di output e in cui le funzioni di attivazione siano tutte di Heaviside (oppure bipolari, cioè con output +1/-1), associando alla rete una funzione energia

$$E = -\frac{1}{2}\sum_{i,j} s_i s_j w_{i,j} + \sum_i s_i \theta_i$$

in cui s_i rappresenta l'output del neurone i.
Ora, consideriamo una rete del genere in cui sia immesso un vettore di input. Si scelga un neurone in modo aleatorio, e se ne aggiorni l'output calcolandolo secondo la regola ordinaria, $s_i(t+1) = H(\sum_j s_j(t) w_{j,i} - \theta_i)$. Se $s_i(t+1) = s_i(t)$, la funzione energia resta invariata. Se invece $s_i(t) = 0$ e $s_i(t+1) = 1$,

allora la funzione energia cambia valore con

$$\begin{aligned}\Delta E &= -\frac{1}{2}\sum_j (w_{i,j}s_i + w_{j,i}s_j) + \theta_i \\ &= -\sum_j w_{i,j}s_i s_j + \theta_i \\ &\leq 0.\end{aligned}$$

Dunque l'aggiornamento di un singolo neurone scelto casualmente può diminuire o lasciare invariato il valore dell'energia della rete. Si parla ora di aggiornamento asincrono, poiché viene aggiornato solo un neurone per volta. Un neurone di questa rete si dice stabile in un dato momento se ricalcolandone l'output questo non viene modificato. Uno stato della rete viene detto stabile se tutti i suoi neuroni sono stabili. Si può osservare che gli stati stabili costituiscono dei minimi locali per l'energia della rete, e che la regola di aggiornamento sopra descritta garantisce la convergenza della rete ad uno stato stabile in un numero finito di passi. La regola di propagazione per una rete di Hopfield è proprio questa: dato un vettore di input per la rete, quest'ultima viene fatta evolvere sino al raggiungimento di uno stato stabile, e questo stato costituisce l'output della rete.

Una rete analoga costruita con funzione di attivazione polare (sgn) anziché di Heaviside fa sì che se una configurazione è simmetrica lo sia anche quella inversa, mentre non è sempre così per valori di attivazione $0/1$.

Le reti Hopfield sono un caso particolare di una classe di reti neurali ricorrenti la cui regola di propagazione si ottiene considerando la rete come un sistema dinamico. Queste reti vengono dette reti ad attrattori (*attractor networks*), e i loro output si ottengono dai punti di stabilità del sistema dinamico, detti appunto attrattori.

1.3.3 Reti neurali ricorrenti: un modello per la memoria

Uno dei compiti considerati standard per un umano, ma di grande difficoltà per una macchina, è la comprensione di un testo. Dato un insieme ordinato di parole, come si può insegnare ad una macchina a comprenderne (o

classificarne) il significato? Risulta evidente come in questo compito ci sia una relazione fra i dati di input più sottile rispetto ad altri casi: quando si volesse, per esempio, classificare il contenuto di un'immagine, tutta l'immagine verrebbe elaborata assieme dalla macchina, contemporaneamente; ciò non ha senso nell'elaborazione di un testo, poiché il significato delle parole non dipende solo dalle parole stesse, ma anche dal loro contesto. Per comprendere un testo quindi non basta conoscere l'insieme di parole che in esso occorrono, ma è necessario metterle in relazione rispettando l'ordine in cui esse vengono lette: è necessario considerare, e successivamente ricordare, un contesto temporale.

Le reti neurali ricorrenti permettono sostanzialmente di ricordare dati che potrebbero risultare significativi nel corso del processo che si vuole studiare. Questo dipende dalla regola di propagazione che si utilizza. Per capire il funzionamento di questo tipo di propagazione, e di memoria, si può considerare un caso particolare di rete neurale ricorrente: la rete di Elman.
Una rete di Elman è molto simile ad una rete neurale feedforward a singolo strato nascosto, ma nello strato nascosto viene aggiunto un insieme di neuroni detti unità di contesto. Per ogni neurone presente nello strato nascosto si aggiunge un'unità di contesto che riceve come input l'output del neurone nascosto corrispondente, e restituisce il proprio output allo stesso neurone nascosto.

Con questa topologia la regola di propagazione per un input $x(t), t = 1, \ldots, n$ è: in primo luogo si introduce $x(1)$ nella rete, si calcola lo stato dei neuroni nascosti senza considerare le unità di contesto, e successivamente si calcolano gli output della rete tramite il terzo strato di neuroni, aggiornando contemporaneamente le unità di contesto; ad ogni passo successivo si introduce $x(t)$ nella rete, poi si aggiornano i valori dei neuroni nascosti, $y_i(t) = \mathcal{F}(\sum_{j=1}^{k} x_j(t) w_{x_j,y_i} + u_i(t) w_{u_i,y_i})$, con $\mathcal{F}$ funzione di attivazione dei neuroni dello strato nascosto, dopodiché si aggiornano lo strato di output e le unità di contesto per ottenere i vettori $z(t)$ e $u(t)$. Si può osservare che, di fatti, $u(t) = y(t-1)$, quindi contemplare queste unità di stato con alcuni pesi fissati equivale a considerare un collegamento dai neuroni dello strato nascosto a sé stessi, con $w_{y_i,y_i} = w_{u_i,y_i}$.

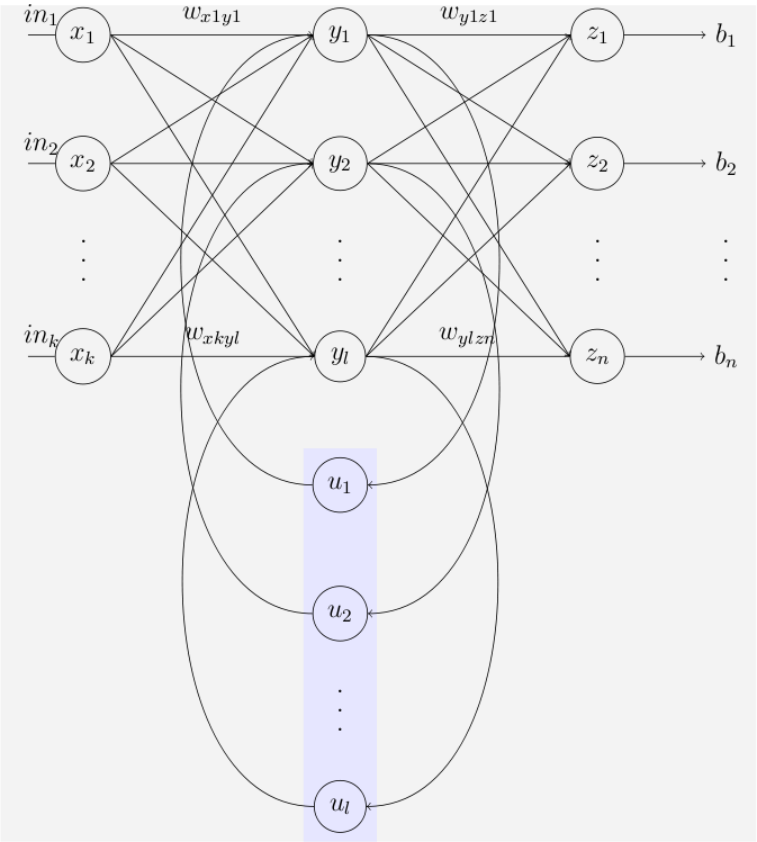


Figura 1.2: Rete di Elman

Un tipo di rete molto simile a quella di Elman è quella di Jordan, in cui le unità di contesto salvano gli stati dei neuroni di output anziché quelli dei neuroni nascosti. L'idea alla base, però, è la stessa del caso di Elman, ed è la stessa di molte altre reti neurali ricorrenti che si basano sul principio: ricevendo in input una sequenza di dati si elabora una nuova sequenza di dati in output ottenuta ricalcolando successivamente i dati degli stessi neuroni.
Le reti ricorrenti che si basano su questo principio sono molteplici, e le singole topologie si scelgono per affrontare problemi differenti. Per esempio, nel caso in cui non sia sufficiente ricordare lo stato precedente della rete, ma possano essere necessarie informazioni elaborate molti passi prima, possono essere utilizzate le reti neurali Long Short-Term Memory.

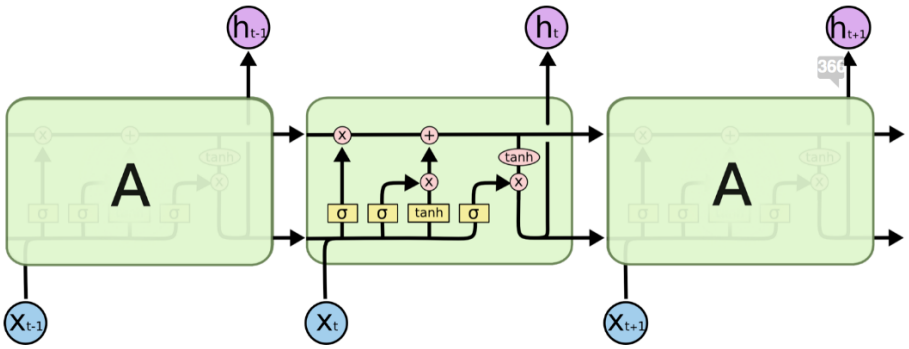


Figura 1.3: L'architettura di una rete LSTM

1.3.4 Come affrontare le dipendenze a lungo termine: le reti LSTM

Come visto, le reti neurali ricorrenti possono essere utilizzate per affrontare problemi che riguardino sequenze in cui il contesto temporale sia importante. Le reti di Elman, però, hanno una memoria piuttosto breve: salvano solo i valori di attivazione dei neuroni nascosti al passo precedente. Talvolta può essere utile, o persino necessario, conservare alcune informazioni più a lungo, per avere un contesto più ampio o più preciso per effettuare una previsione più accurata.
Un esempio è dato dalla creazione di una tastiera predittiva, ovvero uno strumento che, dato un testo, cerchi di prevedere quale sarà la parola successiva. Potremmo voler adoperare questo strumento per un testo di questo tipo:

"Ho trascorso diversi anni in Francia per motivi di lavoro. [...]
Parlo fluentemente il ..."

Dalle parole precedenti possiamo facilmente dedurre che la prossima parola sarà il nome di una lingua, ma per sapere che questa sarà il francese è necessario ricordare un'informazione riportata più indietro.
Per affrontare questo tipo di problemi di dipendenza a lungo termine esiste un'architettura specifica di rete ricorrente: la rete Long Short Term Memory ($LSTM$).

L'elemento chiave in questa struttura è il vettore C_t rappresentato dalla linea orizzontale in cima: esso conserva le informazioni anche a lungo termine, svolgendo la funzione di una sorta di "nastro trasportatore". Viene modificato a seconda delle informazioni ricevute man mano dalla rete, ma non tutte le informazioni modificano questa unità di stato allo stesso modo.

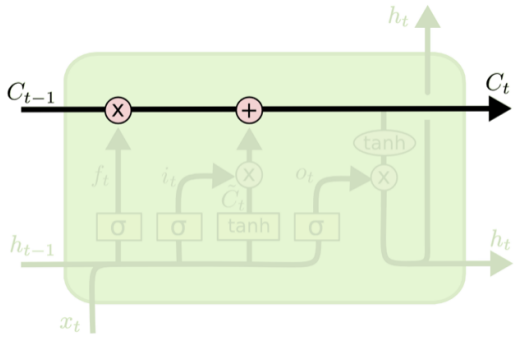
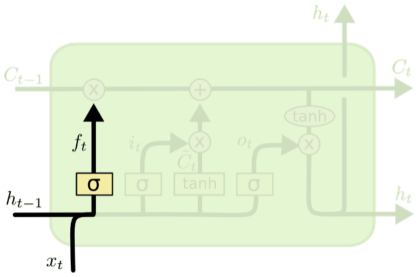


Figura 1.4: L'unità di stato

Il primo passo in un'iterazione è la decisione di quali informazioni vadano dimenticate dall'unità di stato. Per fare ciò si utilizza uno strato feedforward con funzione di attivazione sigmoide, che osserva quali informazioni siano contenute nel vettore h_{t-1} e nel vettore di input attuale x_t per capire come il contesto cambi, e restituisce un vettore i cui elementi siano compresi fra 0 e 1. Questo vettore sarà moltiplicato elemento per elemento col vettore C_{t-1} per decidere quali informazioni debbano essere dimenticate: se il vettore restituito contiene dei numeri molto vicini a 1 significa che le informazioni corrispondenti devono essere ricordate, se restituisce elementi molto vicini a 0 significa che le informazioni corrispondenti devono essere dimenticate.
Nell'esempio della tastiera predittiva, l'unità di stato potrebbe conservare informazioni circa il genere e il numero del soggetto di una frase, e queste andrebbero dimenticate in presenza di un nuovo soggetto.
Per la sua funzione, questo strato della rete è detto "cancello dimenticante" (*forget gate layer*).

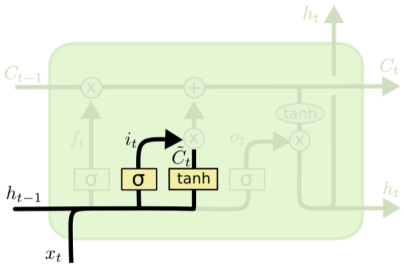
A questo punto le informazioni dell'unità di stato devono essere aggiornate. Per questo scopo viene adoperato il cosiddetto "cancello di input" (*input*



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Figura 1.5: Il cancello dimenticante

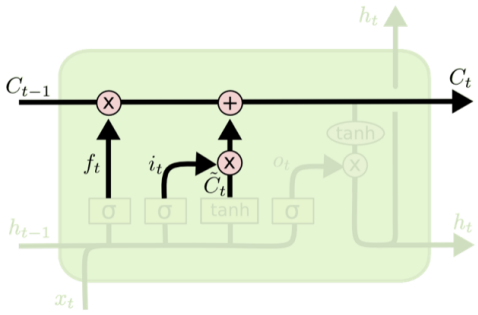
gate layer), strato con attivazione sigmoide che dovrà modulare l'aggiornamento, mentre un altro strato con attivazione *tanh* crea un candidato vettore di aggiornamento. Questi verranno moltiplicati elemento per elemento, e poi il loro prodotto verrà sommato all'unità di stato precedente per generare quella attuale.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

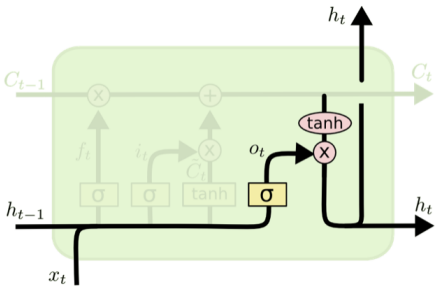
Figura 1.6: Il cancello di input

Ora che è stato determinato il contesto attuale, è il momento per generare l'output della rete. Questo sarà basato su una versione filtrata dell'unità di stato e, naturalmente, sull'input appena ricevuto.
In termini espliciti, mentre al vettore di input e al vettore di output precedente viene assegnato un vettore o_t candidato output attraverso un ulteriore strato con attivazione sigmoide, all'unità di stato viene applicata la funzione *tanh*, e poi questi due vettori vengono moltiplicati elemento per elemento per determinare l'output h_t.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figura 1.7: Aggiornamento dell'unità di stato



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figura 1.8: Calcolo dell'output

Il vettore h_t può essere l'output della rete, ma può anche essere sottoposto a ulteriori elaborazioni nel caso in cui si voglia creare una rete più complessa, aggiungendo profondità.

1.4 L'addestramento

La descrizione di ogni tipo di topologia e regola di propagazione per i diversi tipi di rete neurale potrebbe riempire libri interi, ma sarebbe priva di significato senza la descrizione di una regola di apprendimento. Le reti di neuroni artificiali, infatti, si differenziano dai metodi di programmazione classica proprio per la capacità di risolvere problemi per cui non sia noto a priori un algoritmo, tramite un opportuno addestramento.
I problemi di cui ci occuperemo saranno solo di apprendimento supervisiona-

to, ovvero si cerca di simulare una funzione di cui siano noti alcuni esempi, ovvero di cui si conoscano alcune coppie input-output. Un esempio classico sono i problemi di classificazione: dato un insieme, vogliamo determinare un criterio per assegnare ad alcuni suoi elementi il valore 1, e ad altri il valore 0 (o -1). Vogliamo cioè isolare un sottoinsieme del nostro dominio, di cui conosciamo a priori solo alcuni elementi. Nel pratico si può considerare un'immagine e richiedere ad una funzione di distinguere se contenga gatti o meno, oppure si può considerare un film e richiedere di identificarne il genere. Esistono diversi algoritmi di addestramento per ogni tipo di rete, ciascuno con pregi e difetti caratteristici, ma si basano tutti sull'idea di definire a priori una funzione di costo C dipendente da $y^p - f(x^p)$, ovvero la differenza fra l'output desiderato rispetto all'esempio (x^p, y^p), e l'output effettivamente calcolato dalla rete, e poi minimizzare questa quantità modificando i pesi delle diverse connessioni.

1.4.1 Addestrare un classificatore: la regola del percettrone

Il primo neurone artificiale concepito aveva un funzionamento molto più semplice di quelli moderni, perché come funzione di attivazione veniva contemplata, in origine, solo la funzione di Heaviside. Come risultato, il primo neurone artificiale era un classificatore lineare: divideva lo spazio degli input in due semispazi, uno dei quali veniva mappato in 1, mentre all'altro era assegnato il valore 0. Se i due sottoinsiemi di $\mathbb{R}^n$ da distinguere sono linearmente separabili, dunque, esistono un vettore di pesi w e un bias θ per cui il neurone, detto in questo caso percettrone, classifica correttamente gli insiemi. Il problema è: dato un insieme di esempi, come si può trovare un vettore che li classifichi correttamente? Per risolvere questo problema nasce la regola di apprendimento del percettrone, una delle prime regole di apprendimento supervisionato per un neurone artificiale.
Consideriamo il neurone j da addestrare, che riceva n input reali e restituisca output binario (anche se può essere usata una regola analoga nel caso di output bipolare). Scegliamo il vettore di pesi iniziale w_j^0 e il bias iniziale θ_j^0

in modo casuale, e adatteremo queste quantità secondo le equazioni

$$w_{i,j}^{t+1} = w_{i,j}^{t} + \Delta w_{i,j}$$

$$\theta^{t+1} = \theta^{t} + \Delta\theta$$

sin quando il neurone $f : \mathbb{R}^n \longrightarrow \{0,1\}, x \longmapsto H(< x, w_j > -\theta)$ classificherà correttamente un insieme di esempi.
Per aggiornare i pesi, ovvero decidere le quantità $\Delta w_{i,j}$, dato un esempio (x^p, y^p):

- se $f(x^p) = y^p$, i pesi restano invariati;
- se l'output effettivo è diverso da quello desiderato, si aggiornano i pesi con $\Delta w_{i,j} = \eta \cdot (y^p - f(x^p)) \cdot x^p$, con $\eta \in \mathbb{R}^+$ parametro detto tasso di apprendimento.

Si può dimostrare che, qualora esistano un vettore di pesi e un bias che classificano correttamente un insieme di esempi, l'algoritmo del percettrone converge a una soluzione.
Questa regola è dunque molto utile nell'addestramento di separatori lineari, e ha per questo suscitato grande entusiasmo nella comunità scientifica al momento della sua introduzione.

1.4.2 Addestrare un singolo neurone: la regola delta

Un singolo neurone j, come già visto, può essere visto come una funzione $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ che segue la legge

$$f : (x_1, \ldots, x_n) \longmapsto \mathcal{F}(\sum_{i=1}^{n} x_i w_{i,j} - \theta_j)$$

con $\mathcal{F}$ funzione di attivazione, w_j vettore di pesi e θ_j soglia del neurone.
L'idea dietro l'addestramento di un singolo neurone sta nella minimizzazione della funzione di errore $E = \sum_{p=1}^{k} E^p$ con $E^p = \frac{1}{2}(f(x^p) - y^p)^2$ errore quadratico per gli esempi $(x^p, y^p), p = 1, \ldots, k$. Fissato un insieme di esempi, la funzione E dipende dai pesi e dalla soglia, quindi il problema che ci poniamo

è di minimizzazione di una funzione in $\mathbb{R}^{n+1}$. Per svolgere questo compito di minimizzazione un possibile metodo, nel caso in cui la funzione di attivazione del neurone sia C^1, è quello dei gradienti, detto anche di discesa ripida. L'idea dietro questo metodo è semplice: per minimizzare una funzione si sceglie un punto di partenza w_0 nel dominio, poi ci si muove iterativamente nella direzione dell'antigradiente per scegliere $w_{k+1} = w_k + \eta p_k$, con direzione di discesa $p_k = -\nabla f(w_k)$ e $\eta > 0$ tasso di apprendimento.
Dunque, per modificare il peso $w_{i,j}$ dell'input i-esimo, si può scrivere $\Delta w_{i,j} = -\eta \frac{\partial E}{\partial w_{i,j}}$, e per calcolare questa quantità si può adoperare la regola della catena. Se scriviamo $s_j^p = \sum_{i=1}^n x_i^p w_{i,j} - \theta_j$,

$$\begin{aligned} -\frac{\partial E^p}{\partial w_{i,j}} &= -\frac{\partial E^p}{\partial f(x^p)} \cdot \frac{f(x^p)}{\partial w_{i,j}} \\ &= -\frac{\partial(\frac{1}{2}(f(x^p) - y^p)^2)}{\partial f(x^p)} \cdot \frac{\partial \mathcal{F}(s_j^p)}{\partial s_j^p} \cdot \frac{\partial s_j^p}{\partial w_{i,j}} \\ &= (y^p - f(x^p)) \cdot \mathcal{F}'(s_j^p) \cdot x_i^p \end{aligned}$$

Si può modificare anche il bias in questo modo considerandolo come il peso di un collegamento con un input costante -1.

Questo metodo di addestramento per il neurone viene detto regola delta in quanto, nel caso in cui la funzione di attivazione $\mathcal{F}$ sia l'identità, si può definire la quantità $\delta^p = y^p - f(x^p)$ differenza fra l'output desiderato e quello effettivo, e scrivere $\Delta w_{i,j} = \eta \cdot \delta^p \cdot x_i^p$.

Ora, è possibile addestrare una rete a minimizzare tutti gli esempi contemporaneamente, ovvero cercando di minimizzare tutti gli addendi che compongono E, nel qual caso si parla di addestramento offline, o batch updating, oppure si possono produrre le modifiche per i pesi mirando a minimizzare le singole E^p in iterazioni distinte, cioè si considerano gli esempi in modo isolato durante il calcolo dei $\Delta w_{i,j}$ e si calcolano gli output effettivi uno per volta, nel qual caso si parla di addestramento online, o exemplar updating. Ci sono indicazioni empiriche secondo le quali ci sarebbe convergenza accelerata con il metodo online.
Questo metodo di addestramento è efficace, ma ci sono alcune difficoltà che

possono essere incontrate nella pratica, e per le cui risoluzioni esistono molte varianti del metodo.

Un primo problema è comune ad ogni applicazione del metodo dei gradienti: questo metodo cerca iterativamente un minimo di una funzione "scorrendo" il grafico, ma si può facilmente incastrare in minimi locali. Questo è un probema comune ad ogni applicazione del metodo del gradiente, e una possibile tecnica per evitare i minimi locali è modificare il metodo aggiungendo una componente di "rumore artificiale" con distribuzione gaussiana, la cui media sia nulla e la cui varianza diminuisca con l'avanzare delle iterazioni (*annealing*). Questo rumore permette di evitare soluzioni subottimali, permettendo di evitare le valli del grafico che non siano dei minimi globali, e la diminuzione graduale della componente stocastica fa sì che questa non comprometta la convergenza.

Un secondo problema è la scelta del tasso di apprendimento η: un tasso troppo grande può risultare nella mancata convergenza del metodo, mentre un tasso troppo piccolo rende la convergenza più probabile, ma anche molto più lenta. Per affrontare questo problema sono state proposte varianti del metodo in cui ad ogni modifica del peso si aggiunge una componente che rappresenta una sorta di "quantità di moto" (*momentum*) secondo la formula $\Delta w_{i,j}(t+1) = (y^p - f(x^p)) \cdot \mathcal{F}'(s_j^p) \cdot x_i^p + \alpha \cdot \Delta w_{i,j}(t)$, con α costante che determina l'effetto di ogni passo del cambiamento di peso sul passo successivo. Questo cambiamento permette di evitare grandi oscillazioni qualora si scelga un tasso di apprendimento alto, consentendo di accelerare il processo di minimizzazione evitando rischi.

Un altro problema dell'addestramento di neuroni artificiali, che si presenta quando si utilizzano funzioni di attivazione di tipo sigmoide, è il fenomeno della paralisi. Si può osservare come il cambiamento di peso sia proporzionale alla derivata della funzione di attivazione calcolata nello stato interno del neurone, cioè in s_j. Questo significa che nel momento in cui i pesi diventano molto grandi in valore assoluto anche s_j diventa molto grande in valore assoluto, e questo fa sì che la derivata della funzione di attivazione tenda a zero, portando i cambiamenti nei pesi a diventare sempre più piccoli. Questo fa sì che il neurone diventi lento ad imparare dopo una certa quantità di esem-

pi. Se si optasse per un addestramento online del neurone, quindi, sarebbe necessario prestare attenzione all'ordine in cui gli esempi vengono presentati.

1.4.3 Una generalizzazione della regola delta: la retropropagazione dell'errore

L'idea della regola delta può essere adattata per l'addestramento di reti neurali feedforward dotate di neuroni con funzioni di attivazione C^1. Queste, in effetti, si prestano bene a un metodo iterativo che permette di calcolare in modo diretto i cambiamenti nei pesi dei collegamenti dall'ultimo strato nascosto allo strato di output, e poi di calcolare i cambiamenti nei pesi dei collegamenti fra lo strato $l-1$ e lo strato l in funzione dei cambiamenti nei collegamenti fra lo strato l e lo strato $l+1$. L'errore di approssimazione viene quindi in un certo senso propagato in senso contrario a quello del flusso dei dati nella rete, da qui il nome della tecnica: retropropagazioe dell'errore.
Per derivare l'algoritmo di retropropagazione per le reti feedforward, consideriamone una completamente connessa, ovvero in cui ogni neurone di ogni strato riceva dati da tutti e soli i neuroni dello strato precedente, e i cui risultati siano forniti a tutti e soli i neuroni dello strato successivo.

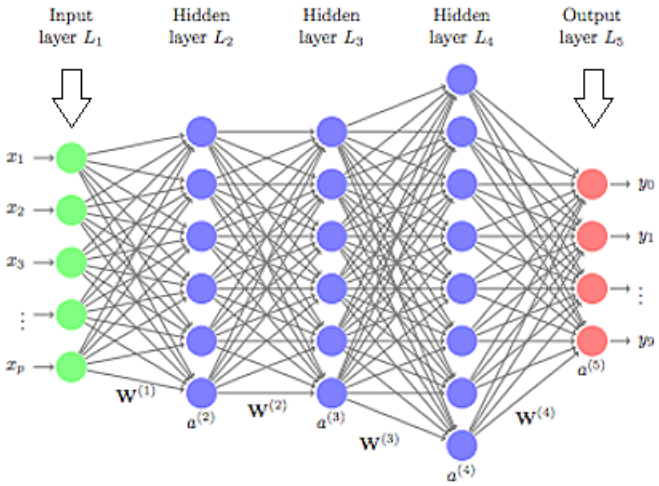


Figura 1.9: Rete feedforward completamente connessa

Consideriamo una rete con L strati (dunque $L-2$ strati nascosti), in cui lo strato l-esimo abbia $n^{(l)}$ neuroni, e sia $\mathcal{F}^{(l)}$ la funzione di attivazione associata ai neuroni dello strato l-esimo. Con abuso di notazione useremo $\mathcal{F}^{(l)}$ come funzione con dominio e codominio in $\mathbb{R}^{n^{(l)}}$, facendo agire la $\mathcal{F}^{(l)}$ per componenti. Consideriamo la funzione di attivazione del primo strato, ovvero lo strato di input, come la funzione identità.
Ora, definiamo stati interni e valori di attivazione dei neuroni:

- per i neuroni del primo strato lo stato sarà $s_j^{(1)} = x_j$, dunque $s^{(1)} = x$;
- per ogni strato di cui sia noto lo stato interno $s^{(l)}$, il valore di attivazione del neurone j-esimo al suo interno sarà $a_j^{(l)} = \mathcal{F}^{(l)}(s_j^{(l)})$, e scriveremo $a^{(l)} = \mathcal{F}^{(l)}(s^{(l)})$;
- per lo strato l-esimo, $l = 2, \dots, L$ lo stato interno del neurone j-esimo sarà la somma pesata dei valori di attivazione dei neuroni nello strato precedente, $s_j^{(l)} = \sum_{i=1}^{n^{(l-1)}} w_{i,j}^{(l)} \cdot a_i^{(l-1)} - \theta_j^{(l)}$.

Specifichiamo la notazione per i pesi dei collegamenti.
Al neurone j-esimo dello strato l associamo il bias $\theta_j^{(l)}$ e il vettore di pesi $w_j^{(l)} \in \mathbb{R}^{n^{(l-1)}}$ il cui elemento i-esimo sia il peso $w_{i,j}^{(l)}$ che collega il neurone i-esimo dello strato $l-1$ al neurone j-esimo dello strato l. Definiamo anche la matrice di pesi $W^{(l)}$ come la matrice dei pesi fra gli strati $l-1$ e l, che contenga i vettori $w_j^{(l)}$ per righe. In questo modo $W^{(l)} \in \mathbb{R}^{n^{(l)} \times n^{(l-1)}}$, e il numero $w_{i,j}^{(l)}$ occupa nella matrice il posto (j, i). La notazione è controintuitiva, ma si presta bene ai calcoli: se consideriamo $\theta^{(l)} \in \mathbb{R}^{n^{(l)}}$ come il vettore dei bias dei neuroni nello strato l-esimo, per $l = 2, \dots, L$ valgono le equazioni in forma matriciale:

$$s^{(l)} = W^{(l)} \cdot a^{(l-1)} - \theta^{(l)};$$
$$a^{(l)} = \mathcal{F}^{(l)}(s^{(l)}).$$

Ora, detti $n = n^{(1)}$ e $m = n^{(L)}$, la funzione rappresentata dalla rete sarà

$$\begin{aligned} f : \mathbb{R}^n &\longrightarrow \mathbb{R}^m \\ x &\longmapsto \mathcal{F}^{(L)}(W^{(L)} \cdot \mathcal{F}^{(L-1)}(\dots) - \theta_j^{(L)}). \end{aligned}$$

Come nel caso del singolo neurone, per l'esempio (x^p, y^p) consideriamo la funzione di errore quadratico $E^p = \frac{1}{2}\|y^p - f(x^p)\|^2$. Quando si voglia effettuare un addestramento offline si può definire per P esempi la funzione di errore $E = \frac{1}{P}\sum_{p=1}^{P} E^p$.
Ora il problema può essere formulato allo stesso modo del caso con un singolo neurone: vogliamo minimizzare la funzione di errore per alcuni esempi dati, per addestrare la rete, e per farlo adoperiamo il metodo di discesa ripida, usando la regola della catena per determinare le variazioni di ogni peso $\Delta w_{i,j}^{(l)} = -\eta \frac{\partial E^p}{\partial w_{i,j}^{(l)}}$.
La determinazione delle modifiche dei pesi $\Delta w_{i,j}^{(L)}$, dunque la modifica all'ultima matrice dei pesi $\Delta W^{(L)}$, non è molto diversa da quella usata per i singoli neuroni. Infatti

$$\begin{aligned} -\frac{\partial E^p}{\partial w_{i,j}^{(L)}} &= -\frac{\partial E^p}{\partial a_j^{(L)}} \cdot \frac{\partial a_j^{(L)}}{\partial s_j^{(L)}} \cdot \frac{\partial s_j^{(L)}}{\partial w_{i,j}^{(L)}} \\ &= (y^p - f(x^p)) \cdot (\mathcal{F}^{(L)})'(s_j^{(L)}) \cdot a_i^{(L-1)}. \end{aligned}$$

Questa equazione ci permette quindi di determinare il cambiamento nella matrice $W^{(L)}$ secondo la regola del gradiente, e ne deduciamo

$$\Delta w_{i,j}^{(L)} = \eta \cdot (y^p - f(x^p)) \cdot (\mathcal{F}^{(L)})'(s_j^{(L)}) \cdot a_i^{(L-1)}.$$

Ora, consideriamo la quantità ausiliaria $\delta_j^{(l)} = \frac{\partial E^p}{\partial s_j^{(l)}}$. Risulta evidente che per ogni $l = 2, \ldots, L$ si ha

$$\begin{aligned} -\frac{\partial E^p}{\partial w_{i,j}^{(l)}} &= -\frac{\partial E^p}{\partial s_j^{(l)}} \cdot \frac{\partial s_j^{(l)}}{\partial w_{i,j}^{(l)}} \\ &= \delta_j^{(l)} \cdot a_i^{(l-1)}, \end{aligned}$$

con le quantità $\delta_j^{(L)}$ già note. Quello che si può fare ora è ricavare $\delta_i^{(l)}$ dalle $\delta_j^{(l+1)}$, dando le regole per il procedimento iterativo che sarà chiamato retropropagazione.
Qualora si voglia calcolare la quantità $\delta_j^{(l)}$, una volta che sia noto tutto il

vettore $\delta^{(l+1)}$, si può usare ancora la regola della catena:

$$\begin{aligned}\delta_j^{(l)} &= \frac{\partial E^p}{\partial s_j^{(l)}} = \sum_{k=1}^{n^{(l+1)}} \frac{\partial E^p}{\partial s_k^{(l+1)}} \cdot \frac{\partial s_k^{(l+1)}}{\partial s_j^{(l)}} \\ &= \sum_{k=1}^{n^{(l+1)}} \delta_k^{(l+1)} \cdot \left(\sum_{h=1}^{n^{(l)}} \frac{\partial s_k^{(l+1)}}{\partial a_h^{(l)}} \cdot \frac{\partial a_h^{(l)}}{\partial s_j^{(l)}} \right) \\ &= \sum_{k=1}^{n^{(l+1)}} \delta_k^{(l+1)} \cdot \frac{\partial s_k^{(l+1)}}{\partial a_j^{(l)}} \cdot \frac{\partial a_j^{(l)}}{\partial s_j^{(l)}} \\ &= \sum_{k=1}^{n^{(l+1)}} \delta_k^{(l+1)} \cdot w_{j,k}^{(l+1)} \cdot (\mathcal{F}^{(l)})'(s_j^{(l)}).\end{aligned}$$

Usando questa formula (che si può anche riscrivere in forma matriciale, il che semplifica l'implementazione) si possono calcolare ricorsivamente tutti i vettori $\delta^{(l)}$ dopo una propagazione in avanti, e dunque si calcola facilmente la modifica dei pesi per una iterazione del metodo del gradiente.
Chiaramente, essendo questo metodo figlio della regola delta per il singolo neurone, soffre delle stesse mancanze: difficoltà coi minimi locali, rischio di paralisi, difficoltà nella scelta del tasso di apprendimento appropriato.

1.4.4 La retropropagazione per le reti ricorrenti: la Backpropagation Through Time

L'addestramento delle reti neurali ricorrenti può essere effettuato tramite una versione della retropropagazione apposita, creata considerando l'elaborazione di una rete ricorrente come equivalente all'elaborazione di una rete feedforward ottenuta "srotolando" la rete originale. Secondo questa concezione si considera l'elaborazione della rete di una sequenza di T vettori di lunghezza n come l'elaborazione di una rete che riceve un solo vettore in input di lunghezza $T \cdot n$.

Questa rete è chiaramente feedforward, e se viene data in input una sequenza $(x_t)_{t=1}^T$ il frutto dell'elaborazione della rete srotolata sarà proprio il vettore y_T ottenuto dalla rete ricorrente come T-esimo vettore della sequenza di output.

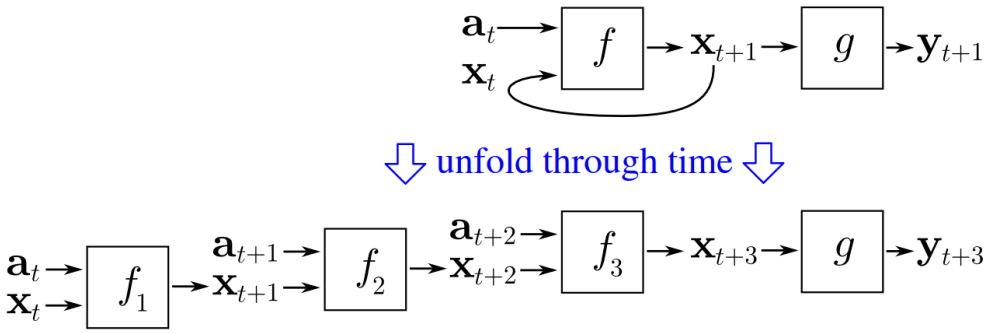


Figura 1.10: Rete ricorrente "srotolata"

Questo ci consente di adoperare l'algoritmo di retropropagazione per addestrare la rete ora srotolata, mantenendo i vincoli opportuni sui pesi: alcuni di essi dovranno essere posti uguali fra loro, per poter ripiegare la rete.
In termini più chiari, l'addestramento di una rete ricorrente può essere effettuato in questo modo: quando venga data una sequenza di input $(x_t)_{t=1}^T$ e un output desiderato y_T, si srotola la rete di modo che diventi feedforward, poi si propaga in avanti la sequenza di input per ottenere l'output effettivo; a questo punto si effettua la retropropagazione ordinaria, e si ottengono le modifiche dei pesi della rete srotolata; per ottenere la modifica dei pesi della rete ricorrente si sommano le modifiche nei pesi corrispondenti nella rete feedforward.
Questo algoritmo, per la sua evidente affinità con la retropropagazione, viene chiamato Backpropagation Through Time (BPTT), e si è dimostrato efficace nonostante le maggiori difficoltà con i minimi locali rispetto al caso feedforward. Il suo principale problema è il costo computazionale: quando sia data una sequenza di esempi come coppie input-output $(x_t, y_t)_{t=1}^T$, sarebbe necessario retropropagare ognuno degli output desiderati attraverso una rete feedforward di lunghezza sempre maggiore, cosicché il calcolo diventa velocemente impegnativo al crescere di T. Per evitare questi costi viene spesso adoperato un algoritmo molto più efficiente come variante: la BPTT troncata.
La BPTT troncata funziona in questo modo: dati una sequenza di esempi $(x_t, y_t)_{t=1}^T$ e due parametri k_1, k_2, si comincia facendo propagare in avanti gli

input, e ogni k_1 passi in avanti si applica la retropropagazione andando indietro di soli k_2 strati. Per k_2 basso, il costo dell'algoritmo diventa sempre più ridotto, rendendo l'addestramento accessibile anche nel caso di sequenze particolarmente lunghe.

Conclusioni

In questa tesi ho descritto le architetture di alcuni tipi di rete neurale, fornendo algoritmi per l'addestramento delle stesse.

Nella seconda sezione sono state analizzate alcune funzioni di attivazione per neuroni artificiali, che quindi offrono diverse possibilità per la creazione di reti diverse a seconda dello scopo.

Nella terza sezione sono stati definiti i tipi principali di rete, fra cui le reti ricorrenti, strumento ampiamente utilizzato nello studio di sequenze temporali quali testi scritti e riconoscimento vocale. Di queste sono stati descritti la regola di elaborazione, cioè come ottenere output da sequenze di input, e una architettura particolarmente utile nella trattazione di sequenze che richiedono una memoria a lungo termine per essere analizzate, la LSTM.

Nella quarta sezione sono stati forniti algoritmi di addestramento utili all'utilizzo delle reti neurali per problemi di apprendimento supervisionato. Abbiamo studiato in particolare la retropropagazione per reti feedforward e ricorrenti, e sono stati presentati accenni a come queste tecniche possano essere raffinate per risolvere problemi comuni nelle applicazioni pratiche.

Bibliografia

[1] Kröse B., van der Smagt P., *An introduction to Neural Networks*, Università di Amsterdam, 1996

[2] Beale R., Fiesler E., *Handbook of Neural Computation*, Oxford University press, 1997

[3] Sutskever I., *Training recurrent neural networks*, tesi di dottorato presso Università di Toronto, 2013

[4] Olah C., *Understanding LSTM networks*, http://colah.github.io/posts/2015-08-Understanding-LSTMs, 2015