Università degli Studi di Cassino
e del Lazio Meridionale

Dipartimento di Ingegneria Elettrica e
dell'Informazione "Maurizio Scarano"

Corso di Laurea in Ingegneria Informatica e
delle Telecomunicazioni

Tesi di Laurea
21 novembre 2023

Algoritmo forward-forward per addestramento reti neurali: studio e applicazione di esempio

Candidato
Alessio Pittiglio
alessio.pittiglio@studentmail.unicas.it

Relatore
Prof. Alessandro Bria

Anno accademico 2022/2023

Indice

Introduzione

Andando a ritroso nella storia dell'intelligenza artificiale, emergono delle pietre miliari che hanno gettato le basi di questa disciplina. Uno dei primi contributi significativi fu offerto da Walter Pitts e Warren McCulloch nel lontano 1943 quando pubblicarono un articolo intitolato "*A Logical Calculus of Ideas Immanent in Nervous Activity*" [1]. Questo articolo è unanimemente considerato come il primo a proporre un modello matematico per rappresentare una rete neurale. Il lavoro di Pitts e McCulloch si fondava su un concetto basilare: l'attività neurale può essere descritta come un fenomeno di tipo "tutto o nulla". Ciò significa che i neuroni possono essere attivi (rappresentati come 1) o inattivi (rappresentati come 0), senza sfumature intermedie. Un passo ulteriore nell'evoluzione delle reti neurali è stato compiuto da Frank Rosenblatt nel 1958 quando scrisse l'articolo intitolato "*The perceptron: a probabilistic model for information storage and organization in the brain*" [2]. In questo articolo Rosenblatt introdusse il modello di percettrone, una struttura di rete neurale che ha giocato un ruolo fondamentale nello sviluppo successivo del machine learning. Nel 1986 Geoffrey Hinton portà alla ribalta l'algoritmo di backpropagation che ha rivoluzionato il modo in cui le reti neurali apprendono dai dati. Un altro punto di svolta nell'ambito delle reti neurali è avvenuto nel 2015 quando Yann LeCun, Yoshua Bengio e Geoffrey Hinton hanno pubblicato l'articolo intitolato "*Deep Learning*" [3]. Il loro straordinario contributo a questo campo è stato ulteriormente riconosciuto nel 2019 quando hanno ricevuto il prestigioso premio Turing, sottolineando l'importanza del loro lavoro per l'intera comunità scientifica.

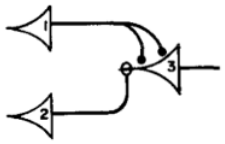


Figura 1: Rete di McCulloch e Pitts che realizza un esempio di negazione. Per generare un segnale di output, ogni neurone deve ricevere due segnali eccitatori e non ricevere segnali inibitori. Le linee che terminano con un pallino rappresentano connessioni eccitatorie. Le linee che terminano con un cerchio rappresentano connessioni inibitorie.

Notazioni

a	Scalare
$\boldsymbol{a}$	Vettore
$\boldsymbol{A}$	Matrice
$\boldsymbol{a}^\top$	Vettore trasposto
$\mathbb{R}$	Insieme dei numeri reali
V	Spazio vettoriale
∇	Gradiente
$\boldsymbol{A} \odot \boldsymbol{B}$	Prodotto componente per componente (Hadamard) di $\boldsymbol{A}$ e $\boldsymbol{B}$
$\frac{\mathrm{d}y}{\mathrm{d}x}$	Derivata di y rispetto a x
$\frac{\partial y}{\partial x}$	Derivata parziale di y rispetto a x
$\nabla_{\boldsymbol{x}} y$	Gradiente di y rispetto a $\boldsymbol{x}$
$\sigma(x)$	Funzione sigmoidea

Capitolo 1

Preliminari

1.1 Basi delle reti neurali

Una rete neurale artificiale (comunemente abbreviata come ANN, dall'inglese "*Artificial Neural Network*") è un modello computazionale che si ispira alle reti neurali biologiche. Le reti neurali biologiche si basano su complessi sistemi di neuroni interconnessi che adattano la loro configurazione in risposta a stimoli esterni. Questo adattamento è ciò che definisce il processo di apprendimento ed è un elemento chiave che le reti neurali artificiali cercano di riprodurre. Un neurone si compone di tre parti:

- i dendriti: linee di ingresso che ricevono i segnali da altri neuroni attraverso le sinapsi;
- il soma: corpo cellulare che li processa e decide quando *sparare*;
- l'assone: linea di uscita del neurone che si dirama in più rami.

Il corpo cellulare del neurone effettua una "somma pesata" dei segnali in ingresso. Se questa somma supera un valore di soglia specifico, il neurone si attiva e genera un potenziale d'azione, che viene quindi trasmesso lungo l'assone. Nel caso in cui la somma non raggiunga il valore di soglia, il neurone rimane in uno stato di riposo. Un neurone artificiale cerca di mimare questo comportamento. Quando un segnale di ingresso x_i giunge al neurone artificiale, esso viene moltiplicato per un valore di peso w_i associato a quel segnale di ingresso. In una fase successiva, i segnali di ingresso pesati vengono sommati. Inoltre, a questa somma viene aggiunto un offset chiamato bias. Infine, il risultato viene trasformato in un segnale di uscita mediante una funzione di attivazione, scelta in modo che abbia valori in $[0, 1]$. Mettendo insieme più neuroni si ottiene una rete neurale artificiale. Le reti neurali artificiali sono organizzate gerarchicamente in strati, le cui unità costitutive sono i neuroni artificiali. Si compongono di:

- uno strato di ingresso che inoltra l'ingresso al resto della rete;
- zero o più strati nascosti che elaborano i dati;
- uno strato di uscita che effettua ulteriori elaborazioni prima di mandarli in uscita.

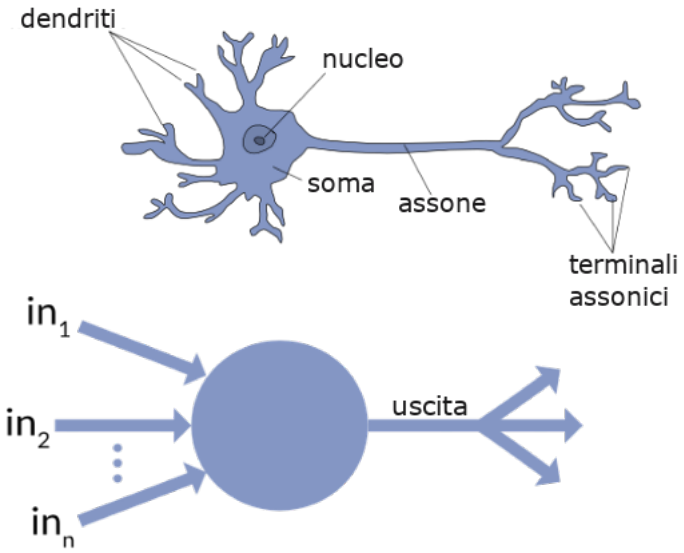


Figura 1.1: Similitudini e differenze tra un neurone biologico e un neurone artificiale.

Le reti neurali artificiali sono degli approssimatori universali di funzioni, ovverosia riescono ad apprendere qualsiasi funzione. Questo risultato è stato dimostrato da G. Cibenko, il quale ha provato che: "*una rete artificiale con un solo strato nascosto può approssimare una qualsiasi funzione g continua (a patto che ci sia un numero sufficiente di neuroni)*" [4]. Le reti neurali possono essere classificate in diversi modi. Abbiamo la distinzione tra reti neurali profonde (DNN), con più strati nascosti, e reti neurali shallow (SNN), con un solo strato nascosto. Un'altra classificazione si basa su come i neuroni sono collegati tra loro, distinguendo le reti neurali feed-forward (FNN), in cui i dati si propagano solo in avanti, dalle reti neurali ricorrenti (RNN), in cui sono ammessi cicli tra i neuroni e l'uscita di uno strato può essere messa in ingresso a uno strato precedente.

1.2 Rete feed-forward

Andiamo a vedere nel dettaglio una rete feed-forward. Brian D. Ripley ci dà una definizione formale nel glossario del suo libro "Pattern Recognition and Neural" (1996) [5]:

> La rete feed-forward è una rete in cui i neuroni possono essere numerati, in modo tale che tutte le connessioni vadano da un neurone ad un altro di numero maggiore. Nella pratica, i neuroni sono organizzati in strati, con connessioni solamente verso i neuroni di numero più alto.

In questa definizione, possiamo individuare alcuni elementi chiave che caratterizzano una rete feed-forward. Siano

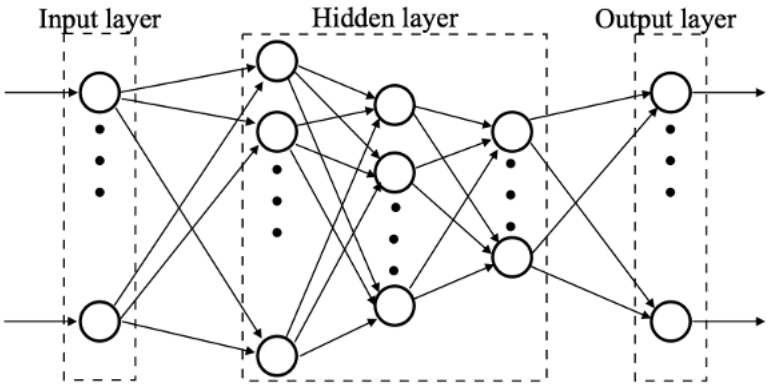


Figura 1.2: Esempio di rete neurale artificiale: in questo caso abbiamo uno strato di ingresso, 3 strati nascosti e uno strato di uscita.

- L: numero di strati presenti nella rete
- p_l: numero di neuroni nello strato l-esimo della rete
- w^l_{jk}: termine di peso associato alla connessione tra il neurone k-esimo dello strato l con il neurone j-esimo dello strato $l-1$-esimo
- b^l_j: il bias del j-esimo neurone nello strato l
- a^l_j: valore dell'attivazione del j-esimo neurone nello strato l

Con questa notazione la relazione ingresso-uscita

$$a^l_j = g^l(z^l_j) \tag{1.1}$$

con

$$z^l_j = \sum_{k=1}^{p_{l-1}} w^l_{jk} a^{l-1}_k + b^l_j \tag{1.2}$$

Si ponga ora

- $\boldsymbol{w}^l_j = [w^l_{1j} \dots w^l_{p_l j}]^\top$
- $\boldsymbol{W}^l = [\boldsymbol{w}^l_1 \dots \boldsymbol{w}^l_{p_{l-1}}]^\top$
- $\boldsymbol{b}^l = [b^l_1 \dots b^l_{p_l}]^\top$
- $\boldsymbol{a}^l = [a^l_1 \dots a^l_{p_l}]^\top$
- $\boldsymbol{z}^l = [a^l_1 \dots z^l_{p_l}]^\top$

Otteniamo la notazione vettoriale

$$\boldsymbol{z}^l = \boldsymbol{W}^l \boldsymbol{a}^{l-1} + \boldsymbol{b}^l \tag{1.3}$$

con

$$\boldsymbol{a}^l = g^l(\boldsymbol{z}^l) \tag{1.4}$$

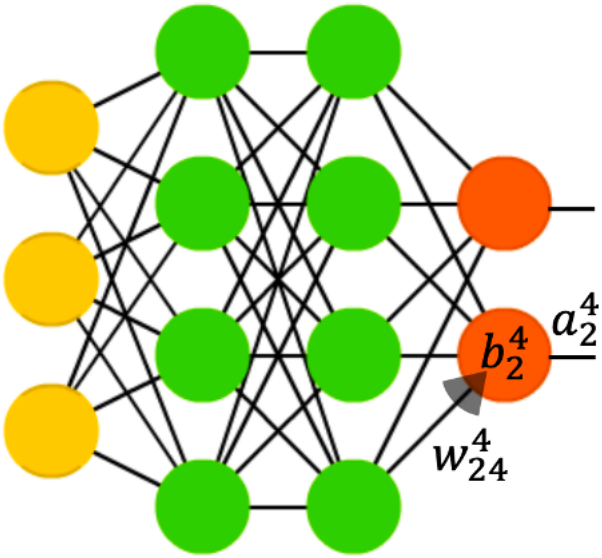


Figura 1.3: Schematizzazione di una rete neurale.

1.3 Addestramento di una rete neurale

Per addestramento di una rete si intende la capacità di generalizzare dalla propria esperienza, ovvero essere in grado di valutare correttamente nuovi esempi dopo aver fatto esperienza su un insieme di dati comunemente chiamato insieme di addestramento (training set). L'apprendimento può essere di due tipi:

- supervisionato: i dati a disposizione contengono degli esempi di input con i corrispondenti output desiderati;
- non supervisionato: i dati a disposizione contengono solo degli esempi non etichettati e a partire da questi deve trovare un certo tipo di struttura nei dati.

Addestrare una rete feed-forward significa minimizzare la funzione di perdita sommata su tutti gli esempi che abbiamo a disposizione. È un esempio di apprendimento supervisionato e si fa ricorrendo alla backpropagation. Durante l'addestramento mandiamo i dati appartenenti all'insieme di addestramento x alla rete e calcoliamo l'uscita a che dipende dagli input x, dai pesi w e dai bias b. Per guidare l'apprendimento dobbiamo quantificare di quanto $a(x)$ si discosta rispetto all'uscita desiderata $y(x)$. Utilizziamo quindi una funzione di perdita (o funzione obiettivo o di costo). La funzione di perdita più semplice è l'errore quadratico medio (MSE) definita come

$$\mathcal{L}_{MSE}(w,b) = \frac{1}{N}\sum_x (y-a)^2 \tag{1.5}$$

Formalmente scriviamo

$$\min_{w,b} = \mathcal{L}(w,b) \tag{1.6}$$

Minimizzare una funzione è un problema di ottimizzazione matematica. In un problema di minimizzazione quando la funzione è convessa, se c'è un minimo locale interno, al contempo è anche un minimo assoluto. La funzione che abbiamo non è assolutamente convessa, quindi pensare di risolvere questo problema eguagliando a zero il gradiente non ci dà garanzia che la soluzione che troviamo sia un minimo assoluto. Se anche provassimo a calcolare il gradiente di questa funzione, la complessità sarebbe enormemente grande perché abbiamo a che fare con funzioni che dipendono da molti parametri. Gli algoritmi per l'addestramento provano a risolvere questo problema con un certo numero di approssimazioni. L'algoritmo che utilizziamo è la discesa del gradiente. La discesa del gradiente è un metodo iterativo del primo ordine. Consiste nello spostarsi lungo la direzione opposta rispetto a quella del gradiente a passi proporzionali ad esso.

$$(w', b') = (w, b) - \eta \nabla \mathcal{L}(w, b) \tag{1.7}$$

$$\nabla \mathcal{L}(w, b) = \left(\frac{\partial \mathcal{L}}{\partial w_1} \frac{\partial \mathcal{L}}{\partial w_2} \dots \frac{\partial \mathcal{L}}{\partial b_1} \dots \right)^\top \tag{1.8}$$

Per prima cosa si sceglie un punto di partenza. Nella pratica, quello che si fa è di inizializzare i pesi e i bias a dei valori casuali. In quel punto calcoliamo il gradiente della funzione. Il gradiente punta nella direzione di massimo incremento della funzione, quindi ci dobbiamo spostare lungo la direzione opposta. L'algoritmo aggiorna in modo iterativo i pesi e i bias e ci si sposta sempre nel verso opposto al gradiente della funzione di perdita. Una volta che abbiamo raggiunto il punto di minimo, l'algoritmo termina.

Algoritmo 1 Discesa del gradiente

Inizializzare $\boldsymbol{x}_{\text{new}}$, $\eta > 0$, $\varepsilon > 0$
ripeti
  $\boldsymbol{x}_{\text{old}} = \boldsymbol{x}_{\text{new}}$
  $\boldsymbol{x}_{\text{new}} = \boldsymbol{x}_{\text{old}} - \eta \nabla f(\boldsymbol{x}_{\text{old}})$
finché $\|\boldsymbol{x}_{\text{new}} - \boldsymbol{x}_{\text{old}}\| < \varepsilon$

ε è la tolleranza sulla convergenza, ovvero l'algoritmo termina quando la differenza tra la x aggiornata e quella precedente è idealmente piccola. η è chiamato step size. Nell'ambito delle reti neurali η si chiama *tasso di apprendimento* (learning rate) e controlla di quanto ci stiamo spostando. Più η è grande e più il nuovo punto sarà lontano dal precedente. Se da un lato potremmo immaginare che avere un η grande ci permetta di rendere l'algoritmo più veloce, d'altro canto può succedere che il passo sia troppo grande e quindi oltrepassare il punto di minimo. La funzione di perdita $\mathcal{L}$ è una media delle perdite $\mathcal{L}_x$ dei singoli input di addestramento x. Il gradiente della funzione di perdita è

$$\nabla \mathcal{L} = \frac{1}{N} \sum_x \mathcal{L}_x \tag{1.9}$$

In generale, la discesa del gradiente è lenta. La funzione che vogliamo minimizzare è una somma di milioni (o miliardi) di termini e calcolare tutti i gradienti per tutti i pesi e i bias è infattibile. Per questo motivo si usa una variazione della discesa del gradiente che si chiama *discesa stocastica del gradiente* (SGD).

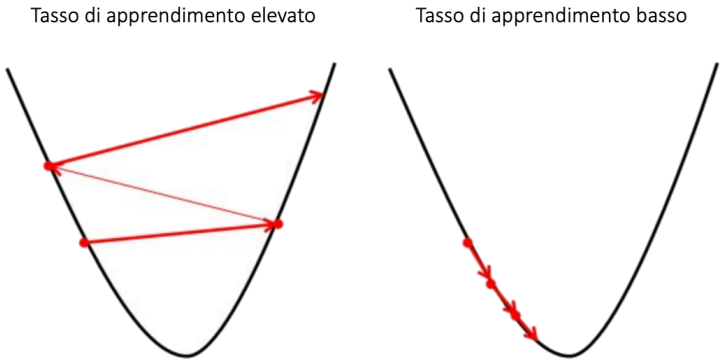


Figura 1.4: Avere un tasso di apprendimento troppo elevato potrebbe farci saltare il punto di minimo, mentre invece avere un tasso di apprendimento troppo basso potrebbe farci convergere troppo lentamente.

Invece di andare a calcolare $\nabla \mathcal{L}_x$ per tutti gli input dell'insieme di apprendimento, lo calcoliamo per una piccola selezione casuale di m campioni, chiamata mini-batch e facciamo un'approssimazione

$$\nabla \mathcal{L} \approx \frac{1}{m} \sum_{j=1}^{m} \nabla \mathcal{L}_{x_j} \tag{1.10}$$

Andiamo avanti finché non otteniamo un errore sull'insieme di addestramento che sia soddisfacente. Ad ogni iterazione, prendiamo un altro mini-batch fino a quando non esauriamo l'insieme di dati iniziale (epoca).

Algoritmo 2 Algoritmo SGD

Inizializzare $\boldsymbol{W}$, $\boldsymbol{b}$
ripeti
  Selezionare un mini-batch casuale
  Calcolare 1.10
  $\boldsymbol{W} = \boldsymbol{W} - \eta \nabla \mathcal{L}(\boldsymbol{W}, \boldsymbol{b}), \boldsymbol{b} = \boldsymbol{b} - \eta \nabla \mathcal{L}(\boldsymbol{W}, \boldsymbol{b})$
finché controllo convergenza

Algoritmo 3 Pseudocode of the gradient method

input: f, x_0
$k = 0$
while condition (stopping condition) **do**
  $p_k = -\nabla f(x_k)$
  α_k backtracking procedure
  $x_{k+1} = x_k + \alpha_k p_k$
  $k = k + 1$
end while
return x_k

Il metodo del momento, descritto per la prima volta da Hinton [6], è un'estensione della discesa del gradiente che aggiunge una sorta di inerzia a una direzione al fine di superare i minimi locali. Questo metodo coinvolge due termini: la velocità v e l'attrito (inverso) μ. La regola di aggiornamento dei pesi (bias) viene sostituita da

$$v' = \mu v - \eta \nabla \mathcal{L} \tag{1.11}$$

$$w' = w + v' \tag{1.12}$$

Quando $\mu = 1$ (nessun attrito), la velocità può aumentare rapidamente nelle iterazioni successive se la direzione del gradiente rimane costante, ma questo potrebbe portare a superare il minimo una volta raggiunto. Scegliendo $\mu \in (0, 1)$ possiamo beneficiare della possibilità di aumentare la velocità senza il rischio di superare il minimo. Un'ulteriore estensione della SGD è l'ADAM [7] (Adaptive Moment Estimation). In questo algoritmo di ottimizzazione si divide il tasso di apprendimento di un parametro per una media mobile dei gradienti recenti di quel parametro. ADAM include nella media anche i secondi momenti del gradiente.

1.4 Backpropagation

Fin ad adesso non abbiamo parlato di come calcolare il gradiente. Calcolare il gradiente è un'operazione onerosa. Quello che si fa è di utilizzare l'algoritmo della backpropagation. Sebbene fosse stato introdotto negli anni '70, questo algoritmo ha guadagnato notorietà grazie a un articolo scritto da David Rumelhart, Geoffrey Hinton e Ronald Williams nel 1986. La backpropagation trasporta il gradiente attraverso la rete e calcola le derivate parziali dei pesi w_i e dei bias b_i in qualsiasi strato. È essenzialmente un'applicazione della regola della catena. La backpropagation è altamente versatile e non dipende dalla specifica funzione di perdita o dal tipo di discesa del gradiente utilizzati. La backpropagation si compone di tre passaggi fondamentali:

1. passaggio in avanti: a partire dall'ingresso x_i, calcoliamo le uscite nei vari strati nascosti e l'uscita dello strato finale;
2. passaggio all'indietro: apportiamo aggiustamenti ai pesi della rete per ridurre l'errore tra l'uscita effettiva e l'uscita desiderata;
3. ripetiamo il passo 2, aggiornando iterativamente i pesi della rete fino a quando il modello non ha eseguito un numero sufficiente di iterazioni (o epoche).

Per illustrare il processo in modo più dettagliato, prendiamo come esempio l'uso di una funzione sigmoidea come funzione di attivazione. Nello strato di uscita possiamo subito calcolare

$$\frac{\partial \mathcal{L}}{\partial a_j^L} \tag{1.13}$$

La funzione di perdita deve essere differenziabile. Procedendo verso sinistra rispetto ai neuroni dello strato di uscita

$$\frac{\partial \mathcal{L}}{\partial z_j^L} = \frac{\partial \mathcal{L}}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L}\bigg|_{a_j^L = \sigma(z_j^L)} = \frac{\partial \mathcal{L}}{\partial a_j^L} \sigma'(z_j^L) \triangleq \delta_j^L \tag{1.14}$$

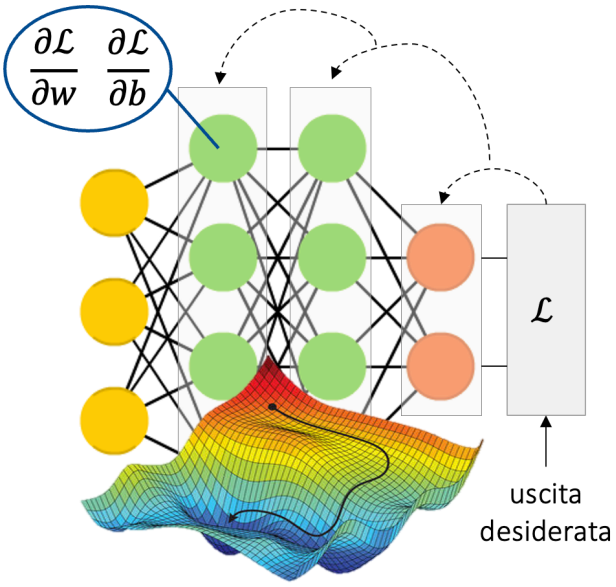


Figura 1.5: Regola della catena

Chiamiamo δ_j^L l'*errore nello strato di uscita*. Usando il prodotto di Hadamard $\odot$, la forma vettoriale è

$$\boldsymbol{\delta}^L = \nabla_{\boldsymbol{a}} \mathcal{L} \odot \sigma'(\boldsymbol{z}^L) \tag{1.15}$$

Diamo un'interpretazione δ_j^L. Il termine $\nabla_{\boldsymbol{a}} \mathcal{L}$ ci fornisce una misura di quanto velocemente la funzione di perdita sta cambiando in risposta all'uscita della rete. Se la funzione di perdita $\mathcal{L}$ è poco sensibile alle variazioni dell'uscita di un particolare neurone di uscita, l'errore relativo associato a quel neurone sarà piccolo.

Il termine $\sigma'(\boldsymbol{z}^l)$ rappresenta una misura di quanto velocemente i neuroni di uscita stanno sparando. Se un particolare neurone di uscita ha o una bassa attivazione (≈ 0) o un'alta attivazione (≈ 1), l'errore relativo a quel neurone sarà circa 0. In tale caso, diciamo che il neurone è *saturo*.

La derivata della funzione di costo rispetto ai pesi dei neuroni di uscita è

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^L} = \frac{\partial \mathcal{L}}{\partial z_j^L} \frac{\partial z_j^L}{\partial w_{jk}^L} \bigg|_{z_j^L = \sum_k w_{jk}^L a_k^{L-1} + b_j^L} = \delta_j^L a_k^{L-1} \tag{1.16}$$

La derivata della funzione di costo rispetto ai bias dei neuroni di uscita

$$\frac{\partial \mathcal{L}}{\partial b_j^L} = \frac{\partial \mathcal{L}}{\partial z_j^L} \frac{\partial z_j^L}{\partial b_j^L} \bigg|_{z_j^L = \sum_k w_{jk}^L a_k^{L-1} + b_j^L} = \delta_j^L \cdot 1 \tag{1.17}$$

In realtà, questi risultati valgono per qualsiasi strato l. Diamo un'interpretazione a

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^l} = \delta_j^l a_k^{l-1} \tag{1.18}$$

Questa equazione può essere riscritta con una notazione meno ricca di indici

$$\frac{\partial \mathcal{L}}{\partial w} = a_{\text{in}} \delta_{\text{out}} \tag{1.19}$$

Quando $a_{\text{in}} \approx 0$, anche il termine $\frac{\partial \mathcal{L}}{\partial w}$ sarà piccolo. In altre parole, i pesi uscenti dai neuroni che hanno un'attivazione bassa imparano lentamente. Procedendo verso sinistra per propagare all'indietro l'errore

$$\delta_j^l = \frac{\partial \mathcal{L}}{\partial z_j^l} = \sum_k \frac{\partial \mathcal{L}}{\partial z_k^{l+1}} \frac{\partial z_k^{l+1}}{\partial z_j^l} = \sum_k \delta_k^{l+1} \frac{\partial z_k^{l+1}}{\partial z_j^l} \tag{1.20}$$

con

$$z_k^{l+1} = \sum_j w_{kj}^{l+1} a_j^l + b_k^{l+1} = \sum_j w_{kj}^{l+1} \sigma(z_j^l) + b_k^{l+1} \tag{1.21}$$

$$\delta_j^l = \sum \delta_k^{l+1} w_{kj}^{l+1} \sigma'(z_j^l) \tag{1.22}$$

che in forma vettoriale può essere riscritta come

$$\boldsymbol{\delta}^l = ((\boldsymbol{W}^{l+1})^\top \boldsymbol{\delta}^{l+1}) \odot \sigma'(\boldsymbol{z}^l) \tag{1.23}$$

Supponiamo di conoscere l'errore δ^{l+1} allo strato $l+1$. Quando moltiplichiamo δ^{l+1} per $(\boldsymbol{W}^{l+1})^\top$ stiamo facendo avanzare l'errore all'indietro attraverso la rete ottenendo una specie di misura dell'errore all'uscita dello strato l. Il prodotto $\odot \sigma'(z^L)$ fa avanzare l'errore all'indietro attraverso la funzione di attivazione nello strato l, fornendoci l'errore nell'input pesato allo strato l.

Capitolo 2

Algoritmo Forward-Forward

Contrariamente a quello che è lo stato dell'arte attuale, Hinton ha proposto un algoritmo alternativo alla backpropagation chiamato *algoritmo Forward-Forward* [8]. L'algoritmo Forward-Forward (FF) sostituisce il passaggio in avanti e il passaggio all'indietro della backpropagation con due passaggi in avanti. La spinta di Hinton a cercare un nuovo algoritmo per l'addestramento delle reti neurali artificiali è scaturita dalla idea che la backpropagation rappresenti un modello implausibile del processo di apprendimento. Non esistono prove convinventi che la corteccia cerebrale propaghi esplicitamente le derivate della funzione di costo o memorizzi le attività neurali per il passaggio all'indietro.

La backpropagation come metodo per apprendere dati sequenziali appare particolarmente improbabile. Immaginiamo il caso di un agente che interagisce con il mondo reale. Per gestire il flusso di input sensoriale senza interruzioni, il cervello deve inviare i dati sensoriali in pipeline e richiede una procedura di apprendimento che possa operare in tempo reale (on-the-fly). La backpropagation non consente a un singolo agente di farlo senza interruzioni: deve compiere una serie di azioni e poi prendersi un breve periodo di pausa mentre propaga all'indietro le informazioni sugli ultimi passaggi temporali.

Un'ulteriore limitazione della backpropagation è che richiede una conoscenza perfetta della computazione eseguita nel passaggio in avanti al fine di calcolare le derivate. Se inseriamo una scatola nera nel passaggio in avanti, non è più possibile eseguire la backpropagation a meno che non apprendiamo un modello differenziabile della scatola nera. L'obiettivo dell'algoritmo Forward-Forward è di superare entrambi questi problemi.

È improbabile che sostituirà la backpropagation per applicazioni in cui non ci sono limiti nella potenza di calcolo. Le due aree in cui l'algoritmo Forward-Forward potrebbe essere superiore alla backpropagation sono come modello di apprendimento e come tecnica per utilizzare hardware analogico a basso consumo energetico.

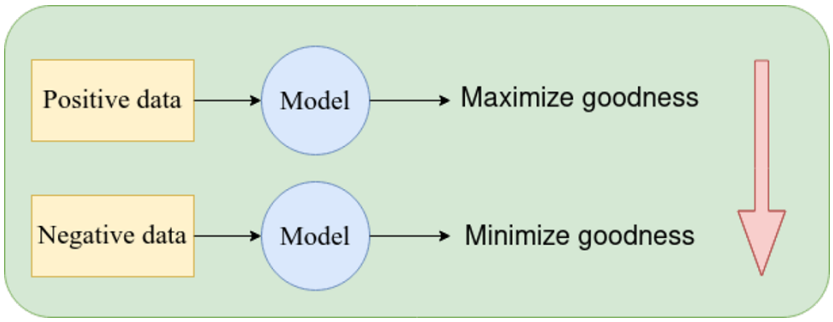


Figura 2.1: Modello algoritmo FF

2.1 Come (non) apprende il nostro cervello

Ci sono diversi motivi per cui la backpropagation non rappresenta un modello di apprendimento valido.

- Nel passaggio all'indietro si impiegano gli stessi pesi usati nel passaggio in avanti per calcolare il gradiente. Dal punto di vista biologico è impossibile per la rete conoscere durante il passaggio all'indietro i pesi utilizzati nel passaggio in avanti. Questo problema è noto come "problema del trasporto dei pesi" [9].
- L'aggiornamento di un peso nelle rete neurale dipende dai calcoli dei neuroni precedenti. D'altra parte, la modifica della forza delle connessioni nelle sinapsi biologiche avviene in risposta a segnali locali, cioè provenienti dai neuroni circostanti [10].
- Per aggiornare i pesi di uno strato è necessario completare il passaggio in avanti e attendere che il passaggio all'indietro raggiunga lo strato considerato. Questo problema è noto come il "problema del blocco degli aggiornamenti" [11].

Se la corteccia non utilizza la backpropagation, deve esistere un altro modo per ottenere i gradienti. Alcuni ricercatori stanno cercando di sviluppare metodi di addestramento che siano più coerenti con i meccanismi di apprendimento osservati nel cervello umano. L'obiettivo è creare algoritmi di apprendimento che siano biologicamente plausibili, cioè che assomiglino di più al modo in cui il cervello apprende. L'algoritmo Forward-Forward trae una certa ispirazione dal processo di apprendimento biologico che avviene nella corteccia: i neuroni vengono eccitati in risposta all'osservazione di immagini e alle relative etichette corrette.

2.2 I dati negativi

L'algoritmo Forward-Forward è una procedura di apprendimento che trae ispirazione dalla macchina di Boltzmann [12]. Gli strati della rete feed-forward sono addestrati in maniera greedy, uno strato alla volta. L'addestramento consiste in due passaggi in avanti che operano nello stesso modo, ma su dati differenti:

1. passaggio positivo: opera sui dati reali e regola i pesi in modo da aumentare la bontà in ogni strato nascosto;

2. passaggio negativo: opera su ciò che chiamiamo "dati negativi" e regola i pesi per diminuire la bontà in ogni strato nascosto.

I dati reali sono quelli dell'insieme di addestramento. Cosa dovremmo utillizare come dati negativi non è molto chiaro e probabilmente dipende dalle situazioni. In prima battuta possiamo immaginarli come una sorta di dati fittizi. La bontà è un altro concetto che non viene esplicitamente definito. Ci sono molti modi in cui potremmo definirla. In questo contesto, la bontà di ogni strato nascosto può essere definita come la somma al quadrato delle attività neurali. Si utilizza questa misura della bontà perché le derivate di una funzione quadratica sono lineari e facili da calcolare rispetto ad altre funzioni più complesse.

L'obiettivo specifico è di rendere la bontà maggiore di una soglia θ per i dati reali e minore della soglia per i dati negativi. In maniera più specifica, si vuole che la rete classifichi correttamente i dati positivi e i dati negativi tenendo conto che la probabilità che un vettore di ingresso sia positivo è data da

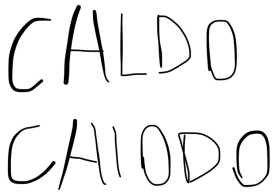
$$p(\text{positive}) = \sigma\left(\sum_j a_j^2 - \theta\right), \tag{2.1}$$

dove a_j è il valore dell'attivazione del j-esimo neurone prima della normalizzazione. La funzione sigmoidea essenzialmente comprime tutto in un intervallo tra 0 e 1. La sua uscita può essere interpretata come una misura della probabilità che il dato in esame appartenga ai dati positivi o meno. In pratica, quanto più l'uscita della sigmoide si avvicina a 1, tanto maggiore è la probabilità che il dato sia positivo, mentre un valore più vicino a 0 suggerisce che il dato sia negativo. Hinton sostiene che la rete ha la capacità di generare internamente i dati negativi utilizzando connessioni top-down oppure possiamo dare in ingresso alla rete dati che sono già noti essere negativi.

È facile vedere che un singolo strato nascosto può essere addestrato facendo sì che la somma delle attivazioni al quadrato sia elevata per i dati positivi e bassa per i dati negativi. Tuttavia, se le attività del primo strato nascosto vengono utilizzate come ingresso per il secondo strato nascosto, è banale distinguere i dati positivi dai dati negativi semplicemente utilizzando la lunghezza del vettore di attività. Per evitare questo, l'algoritmo normalizza la lunghezza del vettore prima di utilizzarlo come input per lo strato successivo. Ciò rimuove tutte le informazioni utilizzate per determinare la "bontà" nel primo strato nascosto e costringe lo strato successivo a utilizzare l'orientamento relativo delle attività delle unità nascoste.

2.3 Rete neurale ricorrente

Il limite principale dell'algoritmo Forward-Forward rispetto alla backpropagation è che quello che viene appreso negli strati successivi non può influenzare ciò che viene appreso negli strati precedenti. Vediamo un esempio che Hinton ha effettivamente fornito in una conferenza in cui parlava di questo.



Sappiamo che nella prima parola la seconda lettera è una A, mentre nella seconda parola abbiamo una H, ma si è cercato di disegnarle in modo che fossero le più simili possibili. Riusciamo comunque a distinguerle e la ragione è che quando espandiamo il contesto e guardiamo l'intera parola, le caratteristiche ad alto livello ci consentono di avere più segnale da ciò che vediamo. La chiave per superare questa limitazione dell'algoritmo Forward-Forward sta nel trattare un'immagine statica come un video piuttosto noioso che viene elaborato da una rete neurale ricorrente a più strati. Ciò che rende questa approccio interessante è la sua somiglianza con il modo in cui gli esseri umani guardano. Infatti, noi osserviamo costantemente un flusso costante di immagini. La variante ricorrente dell'algoritmo è basata su GLOM [13]. Tuttavia, poiché si tratta di una rete ricorrente, richiede passaggi multipli, rallentando l'addestramento. L'algoritmo Forward-Forward va avanti nel tempo sia per i dati positivi che per quelli negativi, ma il vettore di attività in ogni strato è determinato dai vettori di attività normalizzati sia nel livello sopra che in quello sotto al passo temporale precedente. Man mano che il tempo passa c'è un flusso di informazioni verso l'alto e un flusso di informazioni verso il basso.

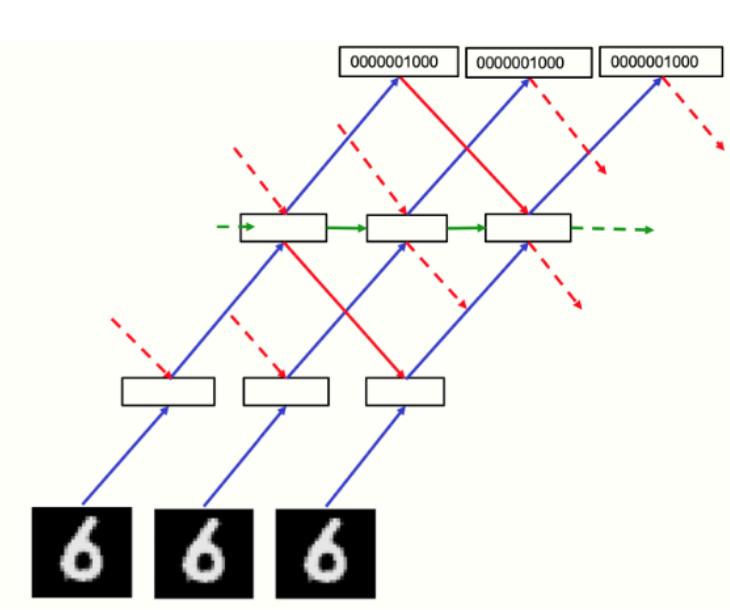


Figura 2.2: Rete neurale ricorrente

2.4 L'attenzione top-down

Nella rete ricorrente l'obiettivo è ottenere una buona concordanza tra l'input proveniente dallo strato superiore e l'input proveniente dallo strato inferiore per i dati positivi e una cattiva concordanza per i dati negativi. In una rete progettata per elaborare immagini un vantaggio significativo potrebbe derivare dalla capacità di avere una visione locale. L'input dall'alto sarà determinato da una regione più ampia dell'immagine e sarà il risultato di più fasi di elaborazione, quindi può essere visto come una previsione contestuale di ciò che dovrebbe essere prodotto dall'input dal basso che è basato su una regione più locale dell'immagine. Quando questi input si combinano è possibile ottenere una migliore comprensione dell'immagine nel suo complesso. Se l'input sta cambiando nel tempo, l'input dall'alto sarà basato su dati di input più vecchi, quindi dovrà imparare a prevedere le rappresentazioni dell'input dal basso. Se invertiamo il segno della funzione di costo e miriamo quindi a ottenere una bassa attività al quadrato per i dati positivi, l'input dall'alto dovrebbe imparare ad annullare l'input dal basso sui dati positivi, il che assomiglia molto a una codifica predittiva. La codifica predittiva è una teoria della funzione cerebrale e si basa sull'idea che il cervello cerca costantemente di fare previsioni sulle informazioni che riceve dall'ambiente. Queste previsioni sono confrontate con le informazioni sensoriali effettive per calcolare gli errori di previsione. Gli errori vengono poi utilizzati per migliorare le previsioni future. L'idea di apprendere utilizzando una previsione contestuale come segnale di insegnamento (teaching signal) per l'estrazione di caratteristiche locali esiste da molto tempo, ma è stato difficile farla funzionare sulle reti neurali utilizzando il contesto spaziale anziché quello temporale.

2.5 SymBa: Symmetric Backpropagation

SymBa [14] è un'estensione dell'algoritmo Forward-Forward che migliora la convergenza affrontando il problema dei gradienti asimmetrici a causa delle direzioni di convergenza contrastanti per dati positivi e negativi. L'algoritmo affronta questo problema garantendo che entrambi i gradienti convergano nella stessa direzione risultando in una convergenza migliorata ed efficiente durante l'addestramento. La funzione di costo è la Noise Contrastive Estimation (NSE):

$$\mathcal{L}_{\text{pos}} = \log(1 + e^{\theta - G_{\text{pos}}}), \mathcal{L}_{\text{neg}} = \log(1 + e^{G_{\text{neg}} - \theta}) \tag{2.2}$$

dove

$$G_{\text{pos}} = \sum_j y_{\text{pos},j}^2, G_{\text{neg}} = \sum_j y_{\text{neg},j}^2. \tag{2.3}$$

Quando il dominio della bontà G è l'insieme di tutti i numeri reali, la formula sopra è simmetrica rispetto a G. Tuttavia, poiché G deve essere maggiore di 0, la funzione di costo positiva e la funzione di costo negativa hanno comportamenti di convergenza diversi rallentando di fatto la convergenza. Infatti:

$$\nabla \mathcal{L}_{\text{pos}} = -\frac{2e^{\theta - G_{\text{pos}}}}{1 + e^{\theta - G_{\text{pos}}}} \sum_j y_{\text{pos},j} \nabla y_{\text{pos},j} \tag{2.4}$$

$$\nabla \mathcal{L}_{\text{neg}} = \frac{2e^{G_{\text{neg}}-\theta}}{1+e^{G_{\text{neg}}-\theta}} \sum_j y_{\text{neg},j} \nabla y_{\text{neg},j} \tag{2.5}$$

Considerando che l'obiettivo dell'algoritmo Forward-Forward è di massimizzare la bontà per i dati positivi e minimizzarla per i dati negativi, ne segue che $G_{\text{pos}} \to \infty$ e $G_{\text{neg}} \to 0$.

$$\nabla \mathcal{L}_{\text{pos}} \approx 0 \tag{2.6}$$

$$\nabla \mathcal{L}_{\text{neg}} \approx \frac{2e^{-\theta}}{1+e^{-\theta}} \sum_j y_{\text{neg},j} \nabla y_{\text{neg},j} \tag{2.7}$$

Questo mostra che il gradiente per dati positivi e negativi si comporta in modo diverso durante la fase finale del processo di addestramento. Per affrontare la natura asimmetrica dell'implementazione originale, si propone una funzione di costo alternativa:

$$\Delta = G_{\text{pos}} - G_{\text{neg}} \tag{2.8}$$

$$\mathcal{L}_{\text{SymBa}} = \log(1 + e^{-\alpha\Delta})) \tag{2.9}$$

dove α è semplicemente un fattore di scala.

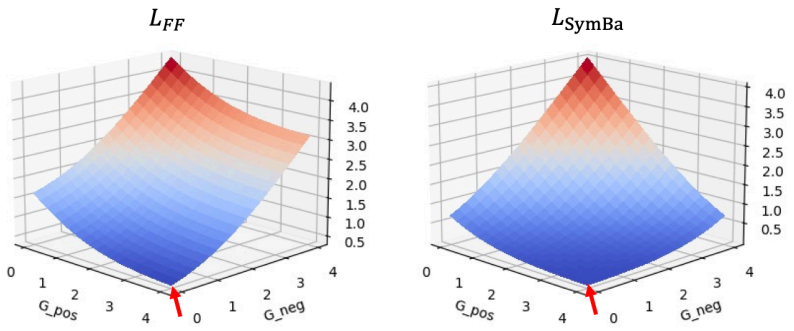


Figura 2.3: Comparazione tra L_{FF} e L_{SymBa}

Nell'articolo Hinton suggerisce di aggiungere le etichette direttamente nell'immagine di input con una codifica one-hot. L'accuratezza può essere valutata misurando la bontà di ciascuna etichetta e selezionando quella con il punteggio più alto determinato da $\text{argmax}_y G(x, y)$. Tuttavia, le informazioni di input vengono compromesse sovrapponendo le etichette all'immagine. Poiché le dimensioni della codifica one-hot sono determinate dal numero di classi, una parte significativa delle immagini viene rimossa man mano che ne aumenta il numero. Ad esempio, nel dataset CIFAR-100 con 100 classi, ogni etichetta copre approssimativamente il 9.77% dell'immagine. Per affrontare questo problema viene proposto un metodo alternativo per incorporare le etichette nelle immagini di input chiamato "Intrinsic Class Patterns" (ICP). Invece di sovrapporre la codifica one-hot, si genera un pattern discreto casuale per ogni classe non soggetto a modifiche durante l'addestramento. Questo approccio evita di coprire direttamente le immagini, preservando così le informazioni originali.

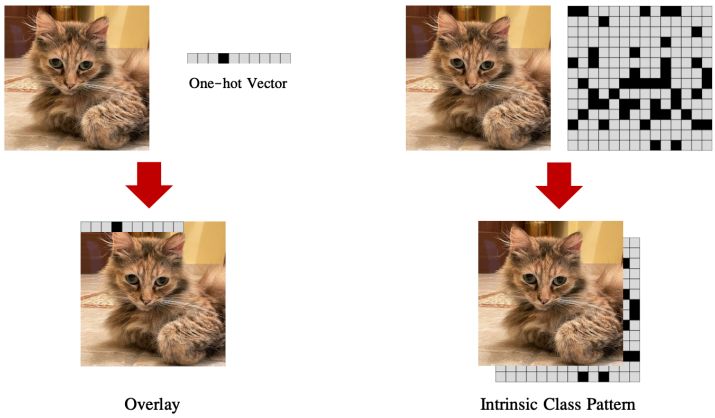


Figura 2.4: Intrinsic Class Pattern (ICP)

2.6 Forward-Forward collaborativo

"Layer collaboration in the Forward-Forward Algorithm" [15] apporta un ulteriore miglioramento all'algoritmo Forward-Forward. Analizzando la collaborazione tra gli strati dei modelli basati sull'algoritmo Forward-Forward, si è notato che tali modelli sono limitati nella loro capacità di trasferire informazioni durante l'addestramento e di conseguenza mostrano una scarsa collaborazione. La collaborazione tra gli strati è cruciale nelle reti neurali profonde perché consente alla rete di imparare ed estrarre caratteristiche dai dati in modo gerarchico. L'algoritmo Forward-Forward permette la comunicazione tra gli strati solo attraverso il passaggio in avanti perché ciascun livello tiene conto solo dell'output del suo predecessore. In questo lavoro si scopre che la collaborazione solo in avanti non è sufficiente per sviluppare una rappresentazione complessa dei dati. In concreto, si propone di considerare la bontà globale della rete per l'ottimizzazione di ciascun strato.

Algoritmo 4 Forward-Forward

Input: $\theta, f, S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$
for $i \leftarrow 1, k$ **do**
  ripeti
    Campione (x, y) da S
    $p \leftarrow \left(\sigma\left(\sum_j a_j^2 - \theta\right)\right)$
    $w_i \leftarrow w_i - \frac{\partial p}{\partial w_i}$
  finché Convergenza
end for

Algoritmo 5 FF collaborativo

Input: $\theta, f, S = \{(x_1, y_1), \ldots, (x_m, y_m)\}$
ripeti
  for $i \leftarrow 1, k$ **do**
    Campione (x, y) da S
    $\gamma \leftarrow \sum_{t \neq l} {a_j^l}^2$
    $p \leftarrow \left(\sigma\left(\sum_j a_j^2 + \gamma - \theta\right)\right)$
    $w_i \leftarrow w_i - \frac{\partial p}{\partial w_i}$
  end for
finché Convergenza

La soglia θ nell'algoritmo Forward-Forward controlla l'entità dell'apprendimento. Formalmente, si propone che la probabilità per i dati positivi sia

$$p(\text{positive}) = \sigma\left(\sum_j a_j^2 + \gamma - \theta\right) \tag{2.10}$$

dove la costante γ è una somma dei valori di bontà provenienti da diversi strati. Ad esempio

$$\gamma = \sum_{t \neq l} {a_j^l}^2 \quad \text{oppure} \quad \gamma_{<t} = \sum_{t<l} {a_j^l}^2 \tag{2.11}$$

La modifica proposta porta al seguente passo di aggiornamento modificato rispetto a w_i:

$$\Delta w_i = \frac{\partial \log\left(\sigma\left(\sum_j a_j^2 + \gamma - \theta\right)\right)}{\partial w_i} \tag{2.12}$$

In modo intuitivo, ciò assicura che lo strato l-esimo sia consapevole dei suoi strati successivi e che i suoi parametri siano addestrati di conseguenza. Viene suggerito anche di modificare la procedura di addestramento. Il classico algoritmo Forward-Forward addestra ogni livello fino alla convergenza. In modo alternativo si suggerisce di alternare gli aggiornamenti degli strati. La combinazione della nuova funzione di costo e della procedura di addestramento modificata dà luogo a un algoritmo Forward-Forward collaborativo.

2.7 Sonno

Hinton sostiene che sarebbe molto più semplice implementare l'algoritmo Forward-Forward in un cervello se i dati positivi fossero elaborati durante lo stato di veglia e i dati negativi fossero creati dalla rete stessa e elaborati durante il sonno. Crick ipotizzò che la fase REM del sonno sia un "apprendimento inverso" che rimuove le informazioni irrilevanti acquisite durante la veglia [16]. In una bozza precedente di questo articolo Hinton aveva riportato che era possibile effettuare molteplici aggiornamenti sui dati positivi seguiti da molteplici aggiornamenti sui dati negativi, generati dalla rete stessa, con una perdita molto limitata delle prestazioni. Utilizzando la somma delle attività al quadrato come bontà, l'algoritmo funziona solo se il tasso di apprendimento è molto basso e il momento è estremamente alto. Sempre secondo Hinton, è probabilmente la questione più importante ancora aperta riguardo all'algoritmo Forward-Forward come possibile modello biologico. Se fosse possibile separare le fasi positive e negative, sarebbe interessante vedere se l'eliminazione degli aggiornamenti nella fase negativa per un certo periodo porti a effetti che mimano quelli causati dalla privazione del sonno. In un articolo recente [17] è stata esplorata la separazione dei due passaggi in avanti. Si è mostrato che le dimensioni del divario tra la fase di sonno e veglia influenzano le capacità di apprendimento dell'algoritmo e che una corretta scelta dei dati negativi può mitigare gli effetti devastanti della privazione del sonno.

2.8 I vantaggi per l'hardware analogico

Nella parte conclusiva dell'articolo Hinton esprime le sue idee più radicali riguardo all'algoritmo Forward-Forward e al futuro dell'IA. L'autore inizia dicendo che le reti neurali, in ultima analisi, si riducono a una serie di moltiplicazioni tra matrici. Dal punto di vista computazionale moltiplicare due numeri binari a n bit richiede $O(n^2)$ operazioni, un costo computazionale significativo. Implementare questa operazione in modo analogico sarebbe più efficiente. I pesi sarebbero

rappresentati come conduttanze poiché c'è una chiara analogia tra la capacità di far passare la corrente e le connessioni tra i neuroni, mentre invece le attività sarebbero implementate come tensioni. Tuttavia, Hinton nota che la backpropagation costituisce un ostacolo alla possibilità di poter utilizzare l'analogico. L'uso di due passaggi avanti potrebbe consentire di superare le limitazioni dei computer digitali.

Il punto di massima radicalità emerge quando Hinton si apre alla possibilità di rinunciare al paradigma di Von Neumann. Da sempre siamo abituati a immaginare la divisione tra hardware e software nei computer. I computer general purpose sono stati pensati con questa filosofia supponendo che l'unica strada per fare dei programmi fosse quello di specificare in maniera pedissequa tutte le operazioni da svolgere. Il deep learning ha già aperto nuovi percorsi, eliminando la necessità di specificare ogni singola operazione in modo dettagliato. In particolare, Hinton critica la comunità per essersi aggrappata all'idea che il software dovrebbe essere separato dall'hardware. Se si è quindi disposti a rinunciare all'immortalità della conoscenza, si potrebbero ottenere dei dispositivi ad alto rendimento energetico. I computer che rispettano la visione di Hinton dovrebbero essere "mortali", simili al concetto di mortalità che possiedono i cervelli biologici. Invece di sviluppare una rete neurale che sia indipendente dall'hardware in cui si trova, si può progettare una rete che si adatti all'hardware e che in qualche modo ne sfrutti le proprietà. Uno dei problemi sarebbe l'impossibilità di copiare i valori dei parametri su un diverso hardware, ma esiste un metodo biologico per farlo che si chiama distillazione [18]: il nuovo hardware viene addestrato a fornire le stesse risposte (approssimativamente) di quello vecchio. Infine, servirebbe un algoritmo che possa funzionare in hardware e l'algoritmo Forward-Forward potrebbe rispondere a questa esigenza. Nonostante ciò, in situazioni in cui non ci sono limiti alle risorse energetiche, la backpropagation rappresenta la strada da seguire e che potrebbe in futuro portarci a raggiungere traguardi inimmaginabili.

Capitolo 3

Esempio di applicazione dell'algoritmo FF

Nell'articolo viene mostrato che l'algoritmo funziona in due esempi che fanno uso di reti neurali relativamente piccole contenenti poche milioni di connessioni. Negli esempi si utilizza la base di dati (dataset) MNIST di cifre scritte a mano. Di queste, 50 000 immagini vengono utilizzate per l'addestramento e 10 000 per la validazione durante la fase di ricerca dei migliori iperparametri. Il set di test contiene 10 000 immagini che vengono utilizzate per calcolare l'errore di test. MNIST è stato studiato a fondo e si conoscono le prestazioni su semplici reti neurali addestrate con la backpropagation. Questo rende MNIST un ottimo banco di prova per testare nuovi algoritmi. Le CNN ben progettate con pochi strati nascosti di solito raggiungono un errore di test di circa 0.6%. Nel caso della versione "permutation-invariant" in cui la rete neurale non riceve informazioni sulla disposizione spaziale dei pixel, le reti neurali feed-forward con pochi strati nascosti di unità lineari rettificate (ReLU) raggiungono tipicamente un errore di test dell'1.4%. Queste reti richiedono circa 20 epoche per addestrarsi. L'errore di test dell'1.4% può essere ulteriormente ridotto all'1.1% utilizzando regolarizzazioni come il dropout (che rende l'addestramento più lento) o il label smoothing (che rende l'addestramento più veloce). In sintesi:

- CNN con pochi strati nascosti: 0.6% di errore di test.
- Reti neurali feed-forward con ReLU: 1.4% di errore di test.
- Utilizzo di regolarizzazioni come dropout o label smoothing: 1.1% di errore di test.

3.1 Apprendimento non supervisionato

Andiamo a vedere come utilizzare l'algoritmo Forward-Forward in un contesto di apprendimento non supervisionato. Un modo comune per utilizzare l'apprendimento contrastivo è imparare prima a trasformare i vettori di input in vettori di rappresentazione senza utilizzare alcuna informazione sulle etichette. Successivamente, si cerca di apprendere una trasformazione lineare di questi vettori di rappresentazione in vettori di logit. I logit sono spesso associati a valori reali,

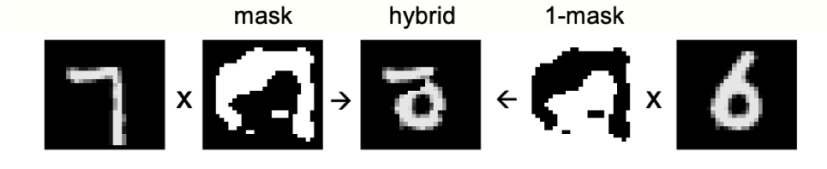


Figura 3.1: Dati negativi generati utilizzando una maschera

positivi o negativi, che rappresentano quanto ciascuna etichetta potrebbe essere associata all'input. I logit vengono poi utilizzati in una funzione softmax per ottenere una distribuzione di probabilità sulle etichette. La funzione softmax normalizza i logit, cioè li converte in valori tra 0 e 1 in modo che la somma di tutti i valori nella distribuzione sia uguale a 1. L'algoritmo Forward-Forward può essere utilizzato per eseguire questo tipo di apprendimento non supervisionato utilizzando vettori di dati reali come esempi positivi e vettori di dati corrotti come esempi negativi. L'apprendimento della trasformazione lineare verso i logit è supervisionato, ma non coinvolge l'addestramento di strati nascosti, quindi non richiede la backpropagation. Esistono molteplici modi diversi per corrompere i dati. Per garantire che l'algoritmo Forward-Forward si concentri sulle correlazioni a lungo raggio, è necessario creare dati negativi che abbiano correlazioni a lungo raggio molto diverse, ma correlazioni a corto raggio molto simili. Ciò può essere fatto creando una maschera contenente regioni di uni e zeri. Successivamente, creiamo immagini ibride sommando insieme l'immagine di una cifra moltiplicata per la maschera e un'altra immagine di una cifra moltiplicata per l'inverso della maschera. Questo crea una sorta di fusione tra le due cifre. La maschera può essere creata partendo da un'immagine binaria casuale e successivamente sfocando ripetutamente l'immagine con un filtro della forma $[1/4, 1/2, 1/4]$ sia in direzione orizzontale che verticale. Dopo ripetute sfocature, l'immagine viene quindi sogliata a 0.5. Ciò significa che i pixel con un valore inferiore a 0.5 diventano 0, mentre quelli con un valore maggiore o uguale a 0.5 diventano 1. Il motivo per cui non usiamo semplicemente dati casuali è che ciò che la rete apprende con il passaggio in avanti dipenderà molto dai dati positivi e negativi. Se i dati negativi sono soltanto dati casuali, sarà molto facile per l'algoritmo identificare le differenze.

3.2 Apprendimento supervisionato

Un modo per per utilizzare l'algoritmo FF in un compito di apprendimento supervisionato è includere l'etichetta nell'input. I dati positivi consistono in un'immagine con l'etichetta corretta, mentre i dati negativi consistono in un'immagine con l'etichetta errata. Le immagini MNIST hanno un bordo nero di 1 pixel su tutti i lati e questa caratteristica può essere sfruttata per includere l'etichetta. Infatti, per implementare questo approccio, Hinton propone di sostituire i primi 10 pixel dell'immagine con una codifica one-hot. Invece di utilizzare una rappresentazione continua per le etichette (come numeri da 0 a 9), ciascuna etichetta è convertita in un vettore binario in cui solo uno dei bit è impostato su 1, indicando la classe corrispondente. Ad esempio, l'etichetta "5" potrebbe essere rappresentata come $[0, 0, 0, 0, 0, 1, 0, 0, 0, 0]$ in una codifica one-hot. La

sostituzione dei primi 10 pixel con questa rappresentazione dell'etichetta consente di visualizzare in modo diretto ciò che il primo strato nascosto del modello apprende. Una rete con 4 strati nascosti, ciascuno contenente 2000 unità ReLU e connettività completa tra i vari strati, ottiene un errore dell'1.36% dopo 60 epoche. La backpropagation richiede circa 20 epoche per ottenere una prestazione simile. Raddoppiando il tasso di apprendimento dell'algoritmo FF e addestrando per 40 epoche invece di 60, si ottiene un errore di test leggermente peggiore, ovvero 1.46% invece di 1.36%. La rete può essere testata in tre diversi modi:

- metodo veloce: forniamo alla rete un'immagine insieme a un'etichetta neutra e addestriamo un classificatore lineare che converte le attività di tutti gli strati nascosti nei logit di un softmax (subottimo);
- metodo lento: diamo alla rete la stessa immagine più volte, una per ogni possibile etichetta e osserviamo quale etichetta dà la massima bontà negli strati nascosti (ottimo);
- metodo ibrido: utilizziamo prima il metodo veloce per ottenere un insieme ristretto di etichette possibili. Successivamente, verifichiamo la bontà di ciascuna etichetta nell'insieme ristretto usando il metodo lento.

Un'etichetta neutra è composta da dieci pixel, ciascuno di valore 0.1. Tutte le classi hanno la stessa probabilità "neutra" di 0.1. Le attività nascoste in tutti i livelli tranne il primo vengono quindi utilizzate come input per un softmax che è stato allenato durante l'addestramento. Questo è un modo rapido, ma non è l'ottimo per classificare un'immagine. L'alternativa è di mandare in ingresso alle rete l'immagine con un'etichetta specifica come parte dell'input e accumulare i punteggi in tutti gli strati tranne il primo. Dopo averlo fatto per ciascuna etichetta separatamente, viene scelta l'etichetta con il punteggio complessivo più alto. Al fine di scegliere di generare dati negativi con etichette che hanno il punteggio più basso, il classificatore softmax lineare viene addestrato con etichette neutrali contemporaneamente alla rete e utilizzato per selezionare le etichette sbagliate da usare per i dati negativi. Questo ha fatto sì che l'addestramento richiedesse circa un terzo delle epoche. Possiamo aumentare i dati (data augmentation) di addestramento sfocando le immagini di massimo due pixel in ogni direzione per ottenere 25 spostamenti diversi per ciascuna immagine. Se la stessa rete viene addestrata per 500 epoche con questi dati aumentati, si ottiene un errore dello 0.64% sul test che è simile a quello di una rete neurale convoluzionale addestrata con backpropagation. Otteniamo anche interessanti campi recettivi (receptive field) nel primo strato nascosto.

3.3 L'algoritmo

Vogliamo addestrare una rete in un problema di apprendimento supervisionato usando il dataset MNSIT. Facciamo riferimento all'implementazione ufficiale dell'algoritmo FF reperibile sul sito personale di Hinton.

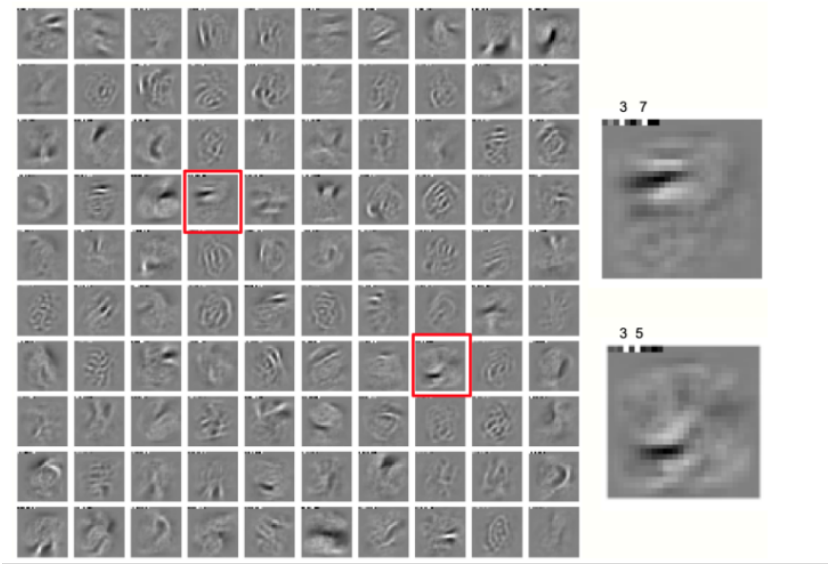
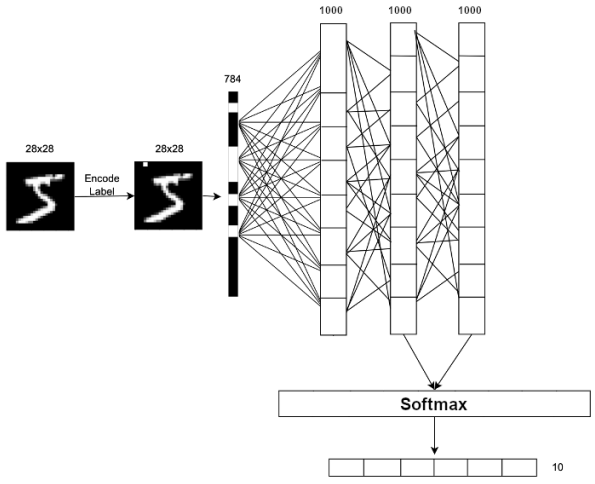


Figura 3.2: Campi recettivi nel primo strato nascosto

3.3.1 Struttura della rete

La rete è formata da cinque strati. Abbiamo uno strato di input con 784 neuroni (28×28) e tre strati nascosti, ciascuno composto da 1000 unità ReLU. Lo strato finale è un softmax con 10 neuroni, uno per ogni classe. Facciamo uso della discesa del gradiente stocastica come ottimizzatore con un tasso di apprendimento di 0.01. Al raggiungere della metà delle epoche facciamo decrescere il tasso di apprendimento linearmente.



3.3.2 Inizializzazione e iperparametri

Il dataset viene suddiviso in 500 batch, ognuno composto da 100 campioni. Si sceglie di avere 1000 neuroni per ogni strato della rete. Nella sua implementazione Hinton imposta la soglia θ pari al numero di neuroni di ogni strato, avendo in questo modo un rapporto di 1:1 tra il numero di neuroni e la soglia.

3.3.3 Ciclo di addestramento

L'algoritmo può essere schematizzato nei seguenti passaggi:

1. Passaggio in avanti con i dati positivi attraverso la rete.
2. Calcoliamo il gradiente dei dati positivi rispetto ai pesi e ai bias.
3. Addestriamo lo strato softmax.
4. Generiamo i dati negativi.
5. Passaggio in avanti con i dati negativi attraverso la rete.
6. Calcoliamo il gradiente dei dati negativi rispetto ai pesi e ai bias.
7. Aggiorniamo i parametri della rete utilizzando il gradiente dei dati positivi e negativi.

Il cuore dell'algoritmo è il ciclo di addestramento che itera attraverso le epoche.

$$a_j^l = \text{reLU}(z_j^l) \tag{3.1}$$

Durante ogni epoca, i mini-batch vengono utilizzati per calcolare i gradienti e aggiornare i pesi. Il gradiente della funzione di costo rispetto a z_j^l è

$$\frac{\partial \mathcal{L}_{\text{pos}}}{\partial z_j^l} = (1 - p(\text{positive}))a_j^l \triangleq \delta_{j,\text{pos}}^l \tag{3.2}$$

Si vuole, infatti, che la probabilità che un dato in ingresso sia positivo quando in ingresso abbiamo dati positivi sia 1. Di conseguenza:

$$\frac{\partial \mathcal{L}_{\text{pos}}}{\partial w_{jk}^l} = \delta_{j,\text{pos}}^l a_k^{l-1} \tag{3.3}$$

$$\frac{\partial \mathcal{L}_{\text{pos}}}{\partial b_j^l} = \delta_{j,\text{pos}}^l \tag{3.4}$$

Dobbiamo calcolare la probabilità mediante uno strato softmax. La funzione di costo è log-verosimiglianza (likelihood) negativa:

$$C = -\log(y) \tag{3.5}$$

Si considerano solo gli ultimi tre strati nascosti (nel nostro caso consideriamo solo gli strati tre e quattro). Includere il primo strato nascosto come parte dell'input per il classificatore lineare peggiora le prestazioni Vengono generati dati negativi per l'addestramento e il processo sopra descritto viene ripetuto per calcolare i gradienti negativi. Nella generazione dei dati negativi i logit associati

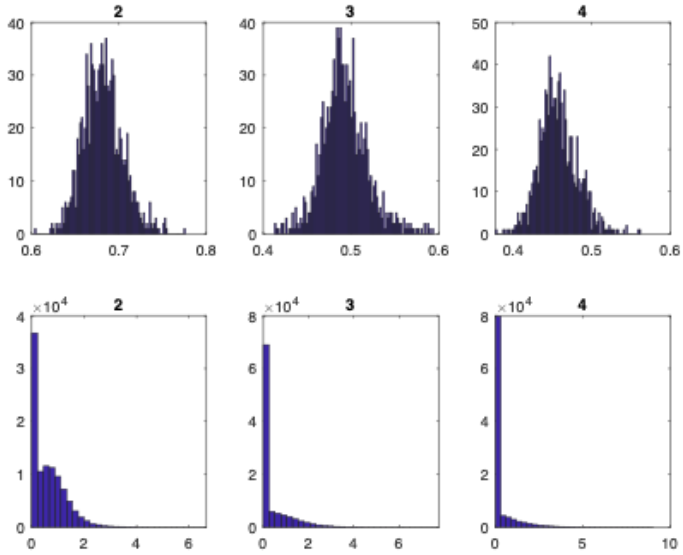


Figura 3.3: Attività e le attività medie degli strati nascosti della rete l dopo 100 epoche. Sulla prima riga ci sono le attività medie e sulla seconda riga le attività non mediate

alle etichette corrette vengono diminuiti drasticamente per evitare che vengano scelte. Per i dati negativi:

$$\frac{\partial \mathcal{L}_{\text{neg}}}{\partial z_j^l} = (0 - p(\text{positive}))a_j^l \triangleq \delta_{j,\text{neg}}^l \tag{3.6}$$

Si vuole che la probabilità che un dato in ingresso sia positivo quando in ingresso abbiamo dati negativi sia 0.

3.3.4 Normalizzazione

Nell'algoritmo FF si fa uso di due diverse normalizzazioni. La "normalizzazione peer" consiste nell'incoraggiare l'attività media di un'unità a coincidere con l'attività media di tutte le unità nello strato. Nella pratica fingiamo che ci sia un gradiente anche quando le unità nascoste sono spente. La forza di questa regressione è regolata da un parametro λ_{mean} in modo che le attività medie siano simili, ma non troppo simili. L'algortmo Forward-Forward utilizza anche una versione molto semplice di normalizzazione degli strati che non comporta la sottrazione della media prima di dividere per la lunghezza del vettore di attività. In altre parole, le attività vengono solo divise per la deviazione standard. La normalizzazione fa in modo in modo che la somma dei quadrati degli elementi delle righe abbia una media pari a 1:

$$\hat{a}_{ij} = \frac{a_{ij}}{\sqrt{\sigma_i^2 + \varepsilon}} \tag{3.7}$$

$$\sigma_i^2 = \frac{1}{n} \sum_{j=1}^{n} a_{ij}^2 \tag{3.8}$$

dove $1 \leq i \leq m$ e $1 \leq j \leq n$. Nella pratica si aggiunge una quantità ε con un valore molto piccolo al denominatore per evitare la divisione per zero.

3.3.5 Monitoraggio e test

Durante l'addestramento vengono stampati periodicamente delle metriche di accuratezza e un grafico a istogramma che rappresenta le attività (medie e non) degli strati nascosti della rete. Infine, viene creato un grafico che mostra l'andamento della funzione di costo durante le epoche di addestramento. Dopo 100 epoche di addestramento si ottiene un tasso di errore di 1.67%. Per concludere, si intende fare un confronto delle prestazioni dell'algoritmo Forward-Forward con la backpropagation. Il modello è stato addestrato utilizzando 50 000 campioni del dataset di MNIST per costituire il set di addestramento, è stata eseguita la validazione su 10 000 campioni e infine è stato condotto un test finale su ulteriori 10 000 campioni. È importante notare che l'algoritmo di backpropagation è stato implementato apportando modifiche alla versione Forward-Forward sviluppata da Hinton.

algoritmo	tempo di addestramento	accuratezza (%)	flop (M)
FF	17.73 minuti	98.14	7.608
BP	15.37 minuti	97.98	7.588

Dall'analisi della tabella sopra riportata emerge che la backpropagation manifesta una maggiore velocità rispetto all'algoritmo Forward-Forward (FF), ma sembra avere un'accuratezza inferiore. Ulteriori miglioramenti potrebbero essere ottenuti attraverso una selezione più accurata degli iperparametri o implementando l'ottimizzatore ADAM.

Appendice A

Macchine di Boltzmann

La macchina di Boltzmann è una rete di neuroni binari con connessioni a coppie che hanno lo stesso peso in entrambe le direzioni. La classe più generale a cui appartiene questa rete è quella dei modelli basati su energia (EBM). La macchina di Boltzmann ha un'energia globale definita per l'intera rete. L'energia globale E è simile a quella delle reti di Hopfield:

$$E = -\left(\sum_{i<j} w_{ij} s_i s_j + \sum_i \theta_i s_i\right) \tag{A.1}$$

dove

- w_{ij} è la forza di connessione tra l'unità j e l'unità i;
- s_i è lo stato dell'unità i (0 o 1);
- θ_i è il bias dell'unità i.

Quando è in esecuzione liberamente, una macchina di Boltzmann aggiorna ripetutamente ciascun neurone impostandolo nello stato "on" con una probabilità data da:

$$p_{i=\text{on}} = \frac{1}{1 + \exp(-\frac{\Delta E_i}{T})} \tag{A.2}$$

dove T è la temperatura del sistema. Questa relazione è la fonte della funzione logistica. Il processo di addestramento della macchina di Boltzmann coinvolge due fasi: positiva e negativa.

1. Fase positiva: le unità visibili sono vincolate (clamped) a un vettore di stato binario campionato dall'insieme di addestramento.
2. Fase negativa: la rete opera liberamente, ovvero le unità visibili hanno il loro stato determinato dai dati esterni, mentre le unità nascoste sono libere di variare.

L'obiettivo è fare in modo che la distribuzione dei vettori binari sui neuroni visibili, quando la rete è in esecuzione liberamente, corrisponda alla distribuzione dei dati. Questo metodo di apprendimento è biologicamente plausibile poiché richiede solo informazioni "locali".

Bibliografia

[1] Warren S McCulloch e Walter Pitts. «A logical calculus of the ideas immanent in nervous activity». In: *The bulletin of mathematical biophysics* 5 (1943), pp. 115–133.

[2] Frank Rosenblatt. «The perceptron: a probabilistic model for information storage and organization in the brain.» In: *Psychological review* 65.6 (1958), p. 386.

[3] Yann LeCun, Yoshua Bengio e Geoffrey Hinton. «Deep learning». In: *nature* 521.7553 (2015), pp. 436–444.

[4] George Cybenko. «Approximation by superpositions of a sigmoidal function». In: *Mathematics of control, signals and systems* 2.4 (1989), pp. 303–314.

[5] B.D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996. ISBN: 9780521460866. URL: `https://books.google.it/books?id=2SzT2p8vP1oC`.

[6] David E Rumelhart, Geoffrey E Hinton e Ronald J Williams. «Learning representations by back-propagating errors». In: *nature* 323.6088 (1986), pp. 533–536.

[7] Diederik P Kingma e Jimmy Ba. «Adam: A method for stochastic optimization». In: *arXiv preprint arXiv:1412.6980* (2014).

[8] Geoffrey Hinton. «The forward-forward algorithm: Some preliminary investigations». In: *arXiv preprint arXiv:2212.13345* (2022).

[9] Stephen Grossberg. «Competitive learning: From interactive activation to adaptive resonance». In: *Cognitive science* 11.1 (1987), pp. 23–63.

[10] Giorgia Dellaferrera e Gabriel Kreiman. «Error-driven input modulation: solving the credit assignment problem without a backward pass». In: *International Conference on Machine Learning*. PMLR. 2022, pp. 4937–4955.

[11] James CR Whittington e Rafal Bogacz. «Theories of error back-propagation in the brain». In: *Trends in cognitive sciences* 23.3 (2019), pp. 235–250.

[12] Geoffrey E Hinton, Terrence J Sejnowski et al. «Learning and relearning in Boltzmann machines». In: *Parallel distributed processing: Explorations in the microstructure of cognition* 1.282-317 (1986), p. 2.

[13] Geoffrey Hinton. «How to represent part-whole hierarchies in a neural network». In: *Neural Computation* 35.3 (2023), pp. 413–452.

[14] Heung-Chang Lee e Jeonggeun Song. «SymBa: Symmetric Backpropagation-Free Contrastive Learning with Forward-Forward Algorithm for Optimizing Convergence». In: *arXiv preprint arXiv:2303.08418* (2023).

[15] Guy Lorberbom et al. «Layer Collaboration in the Forward-Forward Algorithm». In: *arXiv preprint arXiv:2305.12393* (2023).

[16] Francis Crick e Graeme Mitchison. «The function of dream sleep». In: *Nature* 304.5922 (1983), pp. 111–114.

[17] Mircea-Tudor Lic e David Dinucu-Jianu. «Sleep Deprivation in the Forward-Forward Algorithm». In: *arXiv preprint arXiv:2310.18647* (2023).

[18] Geoffrey Hinton, Oriol Vinyals e Jeff Dean. «Distilling the knowledge in a neural network». In: *arXiv preprint arXiv:1503.02531* (2015).

[19] I. Goodfellow, Y. Bengio e A. Courville. *Deep Learning*. Adaptive Computation and Machine Learning series. MIT Press, 2016. ISBN: 9780262035613. URL: `https://books.google.it/books?id=Np9SDQAAQBAJ`.

[20] Alessandro Bria. *Lectures of Machine and Deep Learning*. 2021.

[21] Alessio Zappone. *Lezioni di Trasmissione ed elaborazione delle immagini*. 2021.

[22] Michael A Nielsen. *Neural networks and deep learning*. Vol. 25. Determination press San Francisco, CA, USA, 2015.