

ALMA MATER STUDIORUM · UNIVERSITÀ DI BOLOGNA

---

---

Scuola di Scienze  
Dipartimento di Fisica e Astronomia  
Corso di Laurea in Fisica

MAGNETIC RESONANCE  
FINGERPRINTING CON RETI  
NEURALI A VALORI COMPLESSI

Relatore:  
Dott. Enrico Giampieri

Presentata da:  
Davide Sangiorgi

Anno Accademico 2018/2019

# Abstract

In questo documento cerco un metodo per migliorare le prestazioni del MRF (Magnetic Resonance Fingerprinting), una tecnica di risonanza magnetica quantitativa. Il problema é quello di diminuire il tempo di calcolo necessario per determinare i parametri tissutali relativi alla risonanza magnetica effettuata. Il metodo proposto é quello dell'utilizzo di reti neurali a valori complessi con input il segnale di risonanza magnetica e con output i valori relativi ai parametri che si vogliono studiare.

Dopo aver chiarito il concetto di risonanza magnetica, di MRF ed i problemi ad essi associati, introduco le reti neurali: l'architettura, la dinamica e l'apprendimento relativi ad esse. Discuto a seguire i problemi relativi all'introduzione dei numeri complessi nel modello di rete neurale e anche i vantaggi che le reti neurali a valori complessi possono portare, non solo rispetto ai metodi tradizionali, ma anche rispetto a reti neurali a valori reali.

Analizzo inoltre delle tecniche utili a migliorare la generalizzazione e rendere le reti neurali a valori complessi una soluzione ancora piú concreta. Studio quindi i miglioramenti introdotti dagli ensemble di reti neurali e dall'applicazione di funzioni d'attivazione stocastiche, che introducono del rumore gaussiano all'interno del modello.

# Risonanza magnetica

L'imaging da risonanza magnetica (MRI) é una tecnica utilizzata principalmente in campo medico per produrre immagini di sezioni del corpo umano. La MRI é basata sui principi della risonanza magnetica nucleare (NMR), una tecnica spettroscopica utilizzata per ottenere informazioni di tipo microscopico, chimico e fisico sulle molecole. La MRI é nata inizialmente come tecnica di imaging tomografico, cioé capace di produrre l'immagine di un sottile strato del corpo umano a partire dal segnale NMR. Ciascuna sezione é composta di vari elementi tridimensionali detti voxel, il cui volume é di circa  $3mm^3$ . L'immagine di risonanza magnetica di una sezione risulterá composta da un insieme di elementi bidimensionali chiamati pixel, la cui intensitá é proporzionale all'intensitá del segnale NMR del voxel corrispondente.

Il segnale di risonanza magnetica é generato dalla rotazione del vettore magnetizzazione dei protoni, in particolare dei protoni dell'idrogeno che compone l' $H_2O$ , molecola comune negli organismi viventi. Normalmente i vettori magnetizzazione sono orientati in maniera casuale, ma se sottoposti a campo magnetico esterno stazionario  $B_0$ , subiscono lo stesso momento torcente. All'equilibrio, il vettore di magnetizzazione risultante é posizionato lungo la direzione del campo magnetico statico  $B_0$  (che é per convenzione lungo l'asse  $z$ ) ed é chiamato magnetizzazione all'equilibrio  $M_0$ . Come appena accennato, in questa configurazione, la componente  $z$  del vettore di magnetizzazione  $M_Z$  é uguale a  $M_0$ .  $M_Z$  é conosciuta come magnetizzazione longitudinale. Inoltre, sempre per effetto di  $B_0$ , l'asse di ciascun protone precede attorno alla direzione di  $B_0$  con frequenza che é caratteristica di ogni elemento atomico e viene denominata frequenza di *Larmor*.

I protoni utilizzati per produrre immagini RM sono quelli dell'idrogeno che, naturalmente, abbondano nei tessuti viventi, in particolare per via delle molecole di acqua. Per mettere in risonanza i protoni dell'idrogeno si invia un'onda radio con frequenza pari alla frequenza di Larmor per l'idrogeno. Inviando l'impulso di radiofrequenza (RF) si determina la sincronizzazione dei protoni nella stessa fase di precessione generando un vettore di magnetizzazione trasversale, che ruota nel piano  $xy$ .

Una volta cessato l'impulso RF si verifica la progressiva desincronizzazione della precessione dei protoni, con conseguente decadimento della magnetizzazione trasversale ed il ritorno al livello energetico inferiore da parte dei protoni che avevano subito l'inversione  $180^\circ$ . In ambedue i casi si verifica il fenomeno chiamato rilassamento, durante il quale si

generano impulsi dati dall'oscillazione della magnetizzazione e misurabili tramite la bobina rice-trasmittente (la stessa bobina che in precedenza ha generato l'impulso RF). Il rilassamento avviene con due costanti di tempo distinte. Il tempo di rilassamento spin-reticolo  $T_1$ , che é relativo al ritorno all'equilibrio della magnetizzazione longitudinale. L'equazione che descrive questo fenomeno in funzione del tempo t é:

$$M_Z = M_0(1 - e^{-t/T_1}) \quad (1)$$

Se il vettore di magnetizzazione ha subito un impulso RF  $180^\circ$ , gradualmente ritornerá alla sua posizione di equilibrio sull'asse  $+z$  ad una velocitá regolata dal  $T_1$ , seguendo l'andamento:

$$M_Z = M_0(1 - 2e^{-t/T_1}) \quad (2)$$

Dopo l'impulso RF la componente della magnetizzazione sul piano  $xy$  comincia a perdere fase poiché ognuno dei pacchetti di spin che la costituiscono é sottoposto ad un campo magnetico leggermente diverso e ruota ad una propria frequenza di Larmor. Piú trascorre il tempo, maggiore é la differenza di fase e minore la componente sul piano  $xy$  della magnetizzazione:

$$M_{xy} = M_{xy0}e^{-t/T_2} \quad (3)$$

I fattori che contribuiscono al decadimento della magnetizzazione trasversale sono le interazioni molecolari (che portano ad un effetto molecolare detto  $T_2$  puro) e le variazioni del campo magnetico statico  $B_0$  (che portano ad un effetto detto  $T_{2_{disomog}}$  di disomogeneitá di campo). La combinazione di questi due fattori é quella che realmente si verifica nel decadimento della magnetizzazione trasversale.

La costante di tempo "combinata" é chiamata  $T_2$  star ed é contraddistinta dal simbolo  $T_2^*$ . La relazione tra il  $T_2$  derivante da processi molecolari e quello dovuto a disomogeneitá di campo magnetico é la seguente:

$$1/T_2^* = 1/T_2 + 1/T_{2_{disomog}} \quad (4)$$

Il ritorno all'equilibrio del vettore di magnetizzazione di un sistema di spin che ha assorbito un impulso RF genera un segnale che puó essere rivelato. La rotazione del vettore di magnetizzazione trasversale attorno alla direzione del campo magnetico statico  $B_0$  induce una corrente elettrica nella bobina posizionata attorno all'asse X. Riportando in un grafico la corrente in funzione del tempo si ottiene un'onda sinusoidale che decadra naturalmente secondo la costante di tempo  $T_2^*$  dovuta alla perdita di fase dei pacchetti di spin. Il segnale originato dal "libero" decadimento della magnetizzazione trasversale é chiamato FID (Free Induction Decay).

Esponendo il sistema ad una serie di impulsi RF di opportuna intensitá e tempo di attivazione (sequenza di impulsi) é possibile produrre un segnale NMR che abbia delle specifiche caratteristiche e che permetta quindi di ottenere immagini pesate in  $T_1$  o in  $T_2$ .

## 0.1 introduzione al MRF

La risonanza magnetica ha lo svantaggio di essere lenta rispetto ad altri strumenti diagnostici ed è generalmente qualitativa: il principale mezzo di informazione utilizzato per caratterizzare una patologia è il contrasto tra i tessuti, piuttosto che le misurazioni assolute dei singoli tessuti. Sebbene queste informazioni si siano dimostrate estremamente utili per la diagnosi, la prognosi e la valutazione terapeutica, la mancanza di quantificazione limita la valutazione obiettiva, porta a una variabilità nell'interpretazione e potenzialmente limita l'utilità della tecnologia in alcuni scenari clinici.

Per superare questa limitazione sono stati compiuti sforzi significativi nello sviluppo di approcci quantitativi in grado di misurare le proprietà dei tessuti come i tempi di rilassamento  $T_1$  e  $T_2$ . La quantificazione delle proprietà dei tessuti consente ai medici di distinguere meglio tra tessuto sano e patologico in senso assoluto, rende più semplice il confronto oggettivo di esami diversi e potrebbe essere più rappresentativo dei cambiamenti rispetto all'imaging ponderato standard.

Queste tecniche si basano sull'acquisizione di più immagini, analizzando un solo parametro per volta, ognuna quindi con uno specifico parametro di acquisizione che varia, mentre le altre sono mantenute costanti. Inoltre, la magnetizzazione deve ripristinare lo stesso stato iniziale per ogni ciclo. La necessità di mantenere costanti tutti i parametri della sequenza tranne uno e la necessità di mantenere il segnale costante hanno reso questi approcci estremamente inefficienti a causa del tempo di scansione prolungato (ancora maggiore del tempo necessario per l'imaging qualitativo) e quindi non adatti ad un ambiente clinico. Sono state sviluppate tecniche alternative più rapide, tra le quali quella che descriverò a breve.

Come appena accennato, la maggior parte degli approcci MR attuali consente la misurazione delle proprietà dei tessuti, ma sono relativamente lenti e generalmente forniscano una sola proprietà alla volta. Inoltre, la caratterizzazione delle patologie dipende spesso non solo da una singola proprietà, ma da una combinazione di proprietà da valutare insieme. In tempi recenti sono state quindi proposte diverse tecniche per abbreviare i tempi di acquisizione e per fornire al contempo misure combinate di  $T_1$  e  $T_2$ . La tecnica che approfondisco è il Magnetic Resonance Fingerprinting (MRF).

Il MRF è un potente strumento diagnostico, prognostico e di valutazione della terapia che, grazie alla sua natura versatile rispetto ad altre modalità di imaging, consente di sondare diversi parametri contemporaneamente. Tuttavia permangono importanti ostacoli all'adozione clinica, in particolare una necessità di una quantificazione al contempo rapida e accurata. Il MRF mira a fornire misurazioni simultanee di più parametri come  $T_1$ ,  $T_2$  densità di spin relativa, disomogeneità  $B_0$  ecc., utilizzando un'unica acquisizione, efficiente in termini di tempo; ne si riassumono i passaggi fondamentali in figura 1.

Invece di eseguire un'acquisizione con tutti i parametri costanti tranne uno, il MRF si affida a parametri di acquisizione deliberatamente variabili in modo pseudocasuale cosicché ciascun tessuto generi un'evoluzione del segnale unica. È possibile simulare le

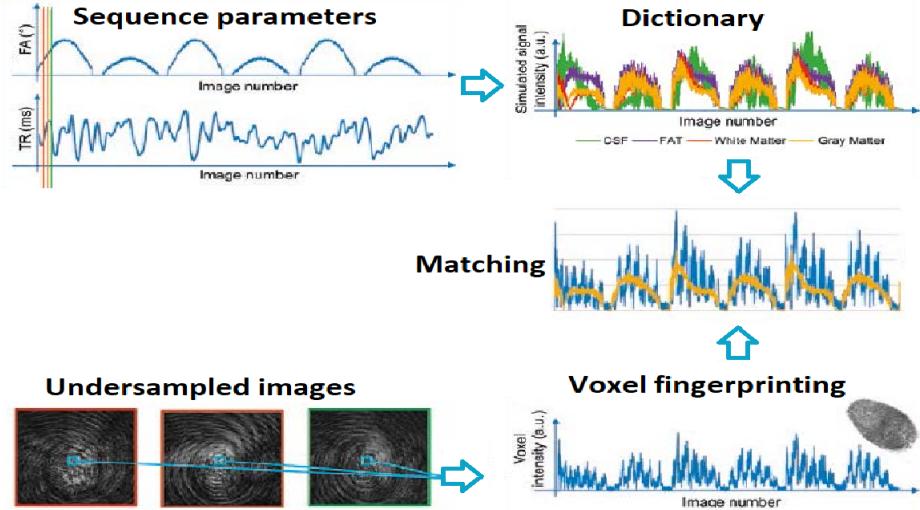


Figura 1: In alto é mostrata la generazione del dizionario, mentre in basso si ha la misura acquisita. Viene poi effettuato il matching per determinare quale segnale del dizionario si avvicina a quello effettivamente misurato.

evoluzioni del segnale utilizzando diversi modelli fisici per un’ampia varietá di combinazioni di parametri tissutali; queste simulazioni sono poi raccolte in un database chiamato dizionario. Dopo l’acquisizione viene utilizzato un algoritmo di riconoscimento del modello per trovare la voce del dizionario che meglio rappresenta l’evoluzione del segnale acquisita da ciascun voxel. I parametri utilizzati per simulare la migliore corrispondenza vengono quindi assegnati al voxel stesso.

Questo processo é analogo a quello di identificazione delle impronte digitali utilizzato dagli esperti forensi per identificare le persone di interesse: l’evoluzione del segnale acquisito é unica per ciascun tessuto e puó essere vista come l’impronta digitale raccolta che deve essere identificata. Il dizionario é equivalente al database in cui tutte le impronte digitali sono memorizzate, insieme a tutte le informazioni relative a ciascuna persona: contiene tutte le evoluzioni fisiologicamente possibili del segnale che possono essere osservate dall’acquisizione e che consente di riconoscere il tessuto all’interno di ogni voxel. Nel caso forense, ogni impronta digitale indica l’identificazione con una caratteristica della persona associata e quindi peso, altezza, data di nascita... Analogamente, nel caso del MRF, ogni impronta digitale del dizionario corrisponde a determinati valori di  $T_1$ ,  $T_2$ , densitá di spin relativa,  $B_0$ , diffusione, etc... Di conseguenza il MRF, come il processo di identificazione forense delle impronte digitali, é efficace solo quando é disponibile un database sufficientemente grande da contenere tutti i potenziali candidati, ma la differenza principale é data dal tempo a disposizione. Cercando, nel caso clinico, di accorciare quest’ultimo non é possibile simulare tutti i casi possibili e vengono selezionati quelli di maggiore interesse per trovare un compromesso tra precisione e velocitá di ricostruzione.

In MRF il dizionario é generato su un computer utilizzando algoritmi che simulano il comportamento di spin durante l’acquisizione e quindi prevedono l’evoluzione realistica del segnale. Solitamente, per simulare i vari effetti della sequenza di acquisizione sugli spin, dato un insieme di parametri tissutali di interesse, vengono utilizzate le equazioni di Bloch. Le informazioni che possono essere recuperate con MRF sono quindi correlate al modo e agli effetti fisici simulati.

Un aspetto critico del dizionario é la sua dimensione: per garantire l’identificazione di ogni possibile parametro tissutale presente nell’acquisizione é necessario simulare un’ampia combinazione di  $T_1$ ,  $T_2$  (e gli altri parametri che si vogliono studiare). Un dizionario TrueFISP standard porta un totale di 363624 combinazioni possibili e include i valori dei parametri che si trovano comunemente nel corpo umano. Il calcolo di un tale dizionario per 1000 punti temporali richiede circa 2.5 minuti su un computer standard utilizzando uno script basato su C++ e raggiunge i 2.5GB di dimensione. Un ulteriore aumento delle dimensioni del dizionario e/o risoluzione aumenterebbe l’accuratezza delle mappe ottenute a spese però di un aumento dei tempi di ricostruzione e dei requisiti di memoria.

La simulazione di un’acquisizione FISP viene calcolata in modo diverso rispetto a quella sopra descritta e il processo di simulazione tramite equazioni di Bloch puó richiedere piú tempo. La sequenza di FISP é meno sensibile agli effetti di risonanza rispetto all’acquisizione TrueFISP, quindi il dizionario corrispondente include solo i tempi di rilassamento  $T_1$  e  $T_2$  come parametri di interesse. Ció comporta solo 18838 voci di dizionario che possono essere calcolate in circa 8 minuti su un computer standard e genera un dizionario di circa 1.2GB. Indipendentemente dalla sequenza utilizzata, il dizionario viene calcolato una sola volta, prima dell’acquisizione.

Dopo l’acquisizione dei dati, l’impronta digitale di ciascun voxel viene normalizzata e confrontata con tutte le voci del dizionario, anch’esse normalizzate, per identificare il tessuto di un dato voxel. La voce del dizionario che meglio corrisponde all’impronta digitale acquisita é considerata una corrispondenza positiva, il che significa che il tessuto rappresentato nel voxel é stato identificato. Tutti i parametri noti relativi a quel segnale possono quindi essere recuperati dal dizionario e assegnati al voxel. L’unicitá dei diversi componenti del segnale e l’accuratezza con cui viene simulato il dizionario sono due componenti cruciali per la corretta stima dei parametri del tessuto.

Sono stati sviluppati diversi metodi per confrontare la misurazione con i dati del dizionario. La versione piú semplice della corrispondenza viene eseguita prendendo il prodotto scalare tra il segnale voxel e ciascun segnale di impronta digitale simulato. La voce che restituisce il valore piú alto é considerata quella che rappresenta al meglio le proprietá del tessuto e i rispettivi parametri sono assegnati al voxel. É stato dimostrato che il prodotto scalare é un’operazione robusta ed é in grado di classificare correttamente i tessuti anche in caso di sottocampionamento o anche in presenza di una qualitá limitata del segnale. La corrispondenza diretta con il prodotto scalare é accurata, ma possono essere necessari fino a 160 secondi per abbinare una porzione 2D con dimensione 128x128,

con 1000 punti temporali con un dizionario che conta 363624 voci. Allo stesso modo ci vogliono circa 30 secondi per abbinare un’immagine 2D con 256x256 voxel, 1000 punti temporali e 18838 voci del dizionario per una ricostruzione FISP.

La corrispondenza puó essere potenzialmente accelerata comprimendo il dizionario, riducendo cosí il numero totale di confronti che devono essere eseguiti. É stato dimostrato che la decomposizione a valore singolare (SVD) puó essere applicata per comprimere il dizionario nella dimensione temporale e ridurre il tempo di corrispondenza di un fattore 3.4 per un dizionario TrueFISP e fino a 4.8 volte per un dizionario FISP. In questo approccio, il dizionario viene proiettato in un sottospazio di dimensione inferiore attraversato dai primi 25-200 vettori singolari ottenuti dall’SVD. L’impronta digitale acquisita viene proiettata sullo stesso sottospazio e la corrispondenza viene eseguita utilizzando il segnale proiettato e il dizionario compresso. In questo modo si rduce il numero di calcoli e di conseguenza anche il tempo di calcolo finale nonostante l’operazione aggiuntiva di proiezione dei dati nel sottospazio.

Un approccio alternativo per ridurre i tempi di calcolo per il matching consiste nel ridurre la dimensione della combinazione di parametri. É stato sviluppato un algoritmo di corrispondenza dei gruppi in cui le voci del dizionario con forti correlazioni sono raggruppate insieme e viene generato un nuovo segnale che rappresenta al meglio il gruppo. L’abbinamento é quindi suddiviso in due fasi: all’inizio l’impronta digitale acquisita viene abbinata al segnale rappresentativo di ciascun gruppo e vengono presi in considerazione solo i gruppi che restituiscono la massima correlazione. Quindi la corrispondenza viene utilizzata per ritrovare il miglior matching tra l’impronta digitale e le voci del dizionario associate ai gruppi prima selezionati. Questo algoritmo aumenta la velocitá di calcolo della corrispondenza di un ordine di grandezza rispetto alla compressione SVD e due ordini di grandezza rispetto alla corrispondenza diretta senza perdita significativa nella qualitá.

Per evitare di limitare a priori la precisione riducendo la dimensione del dizionario, si possono utilizzare le reti neurali. In questo documento analizzo quindi la possibile applicazione di reti neurali a valori complessi al problema del matching in MRF. Prima di affrontare questo argomento, introduco il concetto di rete neurale e cerco di avvalorare la tesi secondo cui, introducendo i numeri complessi all’interno del modello di rete, ne si migliori l’efficacia, l’accuratezza e la resistenza al rumore nei problemi definiti nel dominio complesso, proprio come l’analisi di segnale NMR.

# Capitolo 1

## Reti neurali

Il caso piú semplice di rete neurale é il percettrone mostrato in figura 1.1, introdotto da Rosenblatt nel 1958 [1]. Con esso, senza ancora introdurre alcuna funzione di attivazione (di cui spiego il ruolo nella sezione successiva), si possono già rappresentare tutte le funzioni che sono linearmente separabili come l'AND, l'OR, il NAND o lo XOR. In figura 1.2 viene mostrata invece la rete neurale, dove il percettrone é l'elemento base e ricorsivo della rete.

L'idea della struttura nasce dalla ricerca di creare un algoritmo che lavori in modo analogo al cervello umano, costituito da unitá fondamentali chiamate neuroni, fortemente interconnessi tra loro. La potenza del modello non risiede tanto nell'attività del singolo neurone, in quanto svolge un calcolo semplice, ma risiede principalmente nella configurazione delle connessioni tra neuroni e nel potere di generalizzazione di questo modello, capace quindi di adattarsi al cambiamento di input senza dover riprogettare i criteri di output.

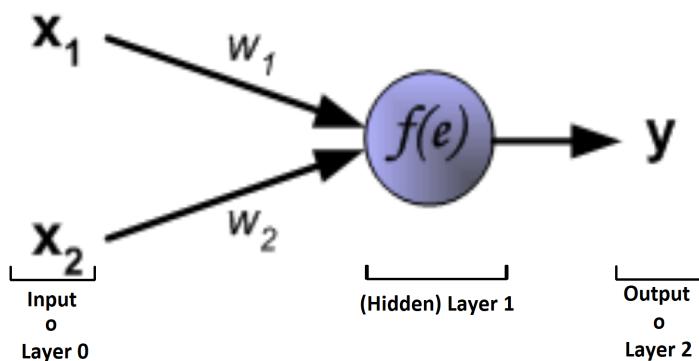


Figura 1.1: Percettrone con input composto da due neuroni  $x_1$  e  $x_2$ , un unico neurone nel layer centrale e un output  $y$ . Il valore del neurone del layer 1 é dato da  $f(e)$  con  $e$  dato dalla somma pesata tra  $x_1$  e  $x_2$  e i pesi che sono rispettivamente  $w_1$  e  $w_2$ .

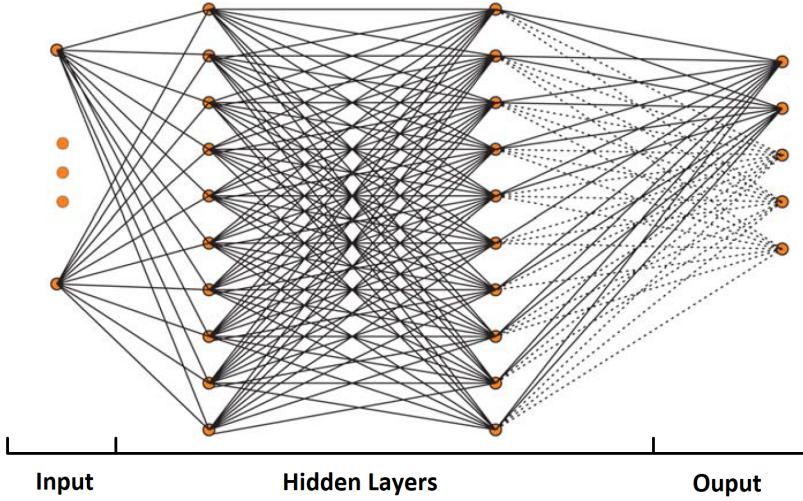


Figura 1.2: Struttura di una rete neurale. Ogni neurone definito dal cerchio arancione si comporta in modo analogo al percettrone visto in precedenza, con la differenza che l’ “output” viene nuovamente combinato, come se fosse uno dei neuroni di input del layer successivo. L’operazione si ripete per il numero di layer della rete, fino a giungere all’output. Continuando l’analoga con la figura 1.1, i pesi  $w_i$  non sono segnati esplicitamente, ma ne viene associato uno ad ognuno dei collegamenti tra neuroni graficati con le linee nere e così vale anche per le funzioni d’attivazione associate ai cerchi arancioni: solitamente viene utilizzata la stessa per tutti i neuroni, ma ci sono esempi di lavori in cui si è preferito applicare una funzione d’attivazione diversa, per esempio, ai neuroni che costituiscono l’output rispetto a tutti gli altri.

## 1.1 Struttura

Partendo dal caso più semplice mostrato in figura 1.1 si visualizzano gli elementi fondamentali che costituiscono una rete neurale. L’input  $x$  è composto da un determinato numero di “neuroni” ed è da essi che il segnale, che chiamerò  $o$ , comincia a propagarsi. Ogni unità che costituisce l’input è collegato con il neurone del layer centrale, propagando quindi il segnale. Ad ogni collegamento si associa un peso ( $w_1$  e  $w_2$  in figura 1.1) che determina l’influenza del “neurone” precedente nella propagazione del segnale e una eventuale funzione di attivazione ( $f(e)$  in figura 1.1) viene applicata al segnale finale per introdurre la non linearità nel sistema, definendo quindi il valore finale del percettrone del layer centrale. Nel caso del percettrone tale valore può essere direttamente l’output a meno di eventuali fattori moltiplicativi, altrimenti, nelle reti come in figura 1.2 tale processo viene poi ripetuto di layer in layer fino ad arrivare all’output della rete.

Testi diversi suggeriscono funzioni d’attivazione diverse, a seconda del problema affrontato. Attualmente non sembra esistere una funzione d’attivazione che prevalga in

modo assoluto per prestazioni, ma la ReLU descritta a breve in equazione 1.6 pare essere una delle piú utilizzate. Mostro comunque altre tra le funzioni di attivazione piú incontrate:

- Threshold Logic Units, cioé funzioni a gradino:

$$f(x) = \begin{cases} 0 & \text{se } x < \alpha \\ 1 & \text{se } x \geq \alpha \end{cases} \quad (1.1)$$

con  $\alpha$  che si aggiunge ai parametri della rete;

- Funzioni lineari:

$$f(x) = \alpha x + \beta; \quad (1.2)$$

con  $\alpha$  e  $\beta$  che si aggiungono ai parametri della rete;

- Sigmoidi:

$$f(x) = \frac{1}{1 + e^{-x}}; \quad (1.3)$$

- tangenti iperboliche

$$f(x) = \tanh(x) \quad (1.4)$$

- gaussiane

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \quad (1.5)$$

con  $\sigma$  e  $\mu$  che si aggiungono ai parametri della rete;

- Rectifier Linear Units (ReLU)

$$f(x) = \text{ReLU}(x) = \max(0; x); \quad (1.6)$$

- Leaky ReLU

$$f(x) = \text{LeakyReLU}(x) = \begin{cases} \lambda x & \text{se } x < 0 \\ x & \text{se } x \geq 0 \end{cases}; \quad (1.7)$$

con  $\lambda$  solitamente  $\approx 1$  che si aggiunge ai parametri della rete;

Passando all'approssimare funzioni piú complesse, si introduce la rete neurale in figura 1.2, in cui ogni elemento di ogni layer si comporta in modo analogo al percettrone.

## 1.2 Dinamica

La dinamica di una rete neurale specifica come il segnale si propaghi lungo la rete. Il numero di neuroni del primo layer definisce la dimensione del vettore  $x$  dell'input per la determinata rete neurale. Definiamo poi  $x_i$  l' $i$ -esimo elemento dell'input. Il segnale viene propagato attraverso la somma pesata dei valori dell'input (i pesi sono i parametri principali che la rete dovrà apprendere).

$$a_j = \sum_{i=1}^n w_i o_i \quad (1.8)$$

con  $j$  che indica che stiamo considerando il  $j$ -esimo neurone del layer successivo e  $o_i$  (che in questo caso è l'input) la componente  $i$ -esima del segnale che si sta propagando dal layer precedente. Il valore finale del  $j$ -esimo neurone del nuovo layer è dato quindi da  $f(a_j)$  con  $f$  la funzione di attivazione citata in precedenza. Il segnale  $o$  si propaga nella maniera appena descritta un layer dopo l'altro finché i termini  $o_k$  non definiscono l'output della rete.

## 1.3 Apprendimento

La definizione di una rete neurale si completa introducendo quello che è l'aspetto principale di questo algoritmo: l'apprendimento. La rete neurale apprende dagli esempi contenuti nel set di training  $\tau = \{z_1, \dots, z_n\}$ . Il set di training differisce a seconda che l'apprendimento sia supervisionato:  $\tau = \{(x_1, \hat{y}_1), \dots, (x_n, \hat{y}_n)\}$  o non supervisionato:  $\tau = \{x_1, \dots, x_n\}$ . Nel caso di apprendimento supervisionato si ha a disposizione un numero di casi etichettati, di conseguenza, una volta dati in analisi alla rete, si riesce a quantificare l'errore della rete confrontando l'output con l'etichetta a disposizione. In questo caso (che è anche quello che viene approfondito di seguito) si riesce ad agire in modo diretto sui parametri della rete. L'apprendimento non supervisionato prevede invece che le informazioni inserite all'interno della rete non siano codificate, dovrà essere essa stessa, quindi, a catalogare tutte le informazioni in proprio possesso e organizzarle.

L'apprendimento è un processo di modifiche progressive effettuate ai parametri della rete (principalmente i pesi, ma, come visto poco fa, anche le funzioni di attivazione possono avere parametri da includere nell'apprendimento). Lontano dall'essere un metodo di memorizzazione dei dati del set di training, si suppone che le regole imparate in questo modo riescano a generalizzare il problema, permettendo alla rete di catalogare anche input mai visti in precedenza.

Il processo di modifica dei parametri, nel caso di apprendimento supervisionato, avviene attraverso la back-propagation. Descrivo di seguito come avviene.

Dato l'output  $y_i$  della rete, si definisce una funzione di costo, per esempio:

$$C(\tau, w) = C(W) = \frac{1}{2} \sum_{i=1}^n (\hat{y}_i - y_i)^2. \quad (1.9)$$

Per un peso generico, la correzione applicata durante la back-propagation é data da:

$$w' = w + \Delta w \quad (1.10)$$

$$\Delta w = -\eta \frac{\partial C}{\partial w} \quad (1.11)$$

con  $\eta$  tasso di apprendimento da fissare a priori.

Sia  $l$  il numero di layer della rete, chiamiamo  $L_0$  l'input,  $L_l$  l'output e  $L_1, \dots, L_{l-1}$  gli hidden layer. Con  $k \in L_j$  indichiamo il  $k$ -esimo neurone del layer  $L_j$ . L'obiettivo attuale é quello di calcolare  $\Delta w = -\eta \frac{\partial C}{\partial w}$  per un peso  $w$  generico della rete. Consideriamo il peso  $w_{ij}$ , con  $j \in L_{l-1}$  e  $i \in L_l$ , allora in un hidden layer si ha:

$$\begin{aligned} \frac{\partial C}{\partial w_{jk}} &= \frac{\partial}{\partial w_{jk}} \left\{ \frac{1}{2} \sum_{i=1}^m (\hat{y}_i - y_i)^2 \right\} \\ &= \frac{\partial}{\partial o_j} \left\{ \frac{1}{2} \sum_{i=1}^m (\hat{y}_i - y_i)^2 \right\} \frac{\partial o_j}{\partial w_{jk}} \\ &= \left\{ \frac{1}{2} \sum_{i=1}^m \frac{\partial}{\partial o_j} (\hat{y}_i - y_i)^2 \right\} \frac{\partial o_j}{\partial w_{jk}} \\ &= \left\{ - \sum_{i=1}^m (\hat{y}_i - y_i) \frac{\partial y_i}{\partial o_j} \right\} \frac{\partial o_j}{\partial w_{jk}} \end{aligned} \quad (1.12)$$

$$\begin{aligned} \frac{\partial y_i}{\partial o_j} &= \frac{\partial f_i(a_i)}{\partial a_i} \frac{\partial a_i}{\partial o_j} \\ &= \dot{f}_i(a_i) \frac{\partial}{\partial o_j} \sum_j w_{ij} o_{ij} \\ &= \dot{f}_i(a_i) w_{ij} \end{aligned} \quad (1.13)$$

$$\begin{aligned} \Rightarrow \frac{\partial C}{\partial w_{jk}} &= \left\{ - \sum_{i=1}^m (\hat{y}_i - y_i) \dot{f}_i(a_i) w_{ij} \right\} \frac{\partial o_j}{\partial w_{jk}} \\ &= - \left( \sum_{i=1}^m w_{ij} \delta_i \right) \frac{\partial f_j(a_j)}{\partial a_j} \frac{\partial a_j}{\partial w_{jk}} \\ &= - \left( \sum_{i=1}^m w_{ij} \delta_i \right) \dot{f}_j(a_j) o_k \end{aligned} \quad (1.14)$$

$$\Rightarrow \Delta w_{jk} = -\eta \frac{\partial C}{\partial w_{jk}} = \eta \left( \sum_{i=1}^m w_{ij} \delta_i \right) \dot{f}_j(a_j) o_k \quad (1.15)$$

con  $\delta_i = (\hat{y}_i - y_i) \dot{f}_i(a_i)$ .

Definendo:

$$\delta_j = \begin{cases} (\hat{y}_j - y_j) \dot{f}_j(a_j) & \text{se } j \in L_l \\ \left( \sum_{i \in L_{k+1}} w_{ij} \delta_i \right) \dot{f}_j(a_j) & \text{se } j \in L_k \text{ dove } k = l-1, \dots, 0 \end{cases} \quad (1.16)$$

possiamo allora generalizzare la correzione da applicare ad un peso di un layer qualsiasi come:

$$\Delta w_{jk} = \eta \delta_j o_k \quad (1.17)$$

Nella seguente immagine si può visualizzare il procedimento appena descritto.

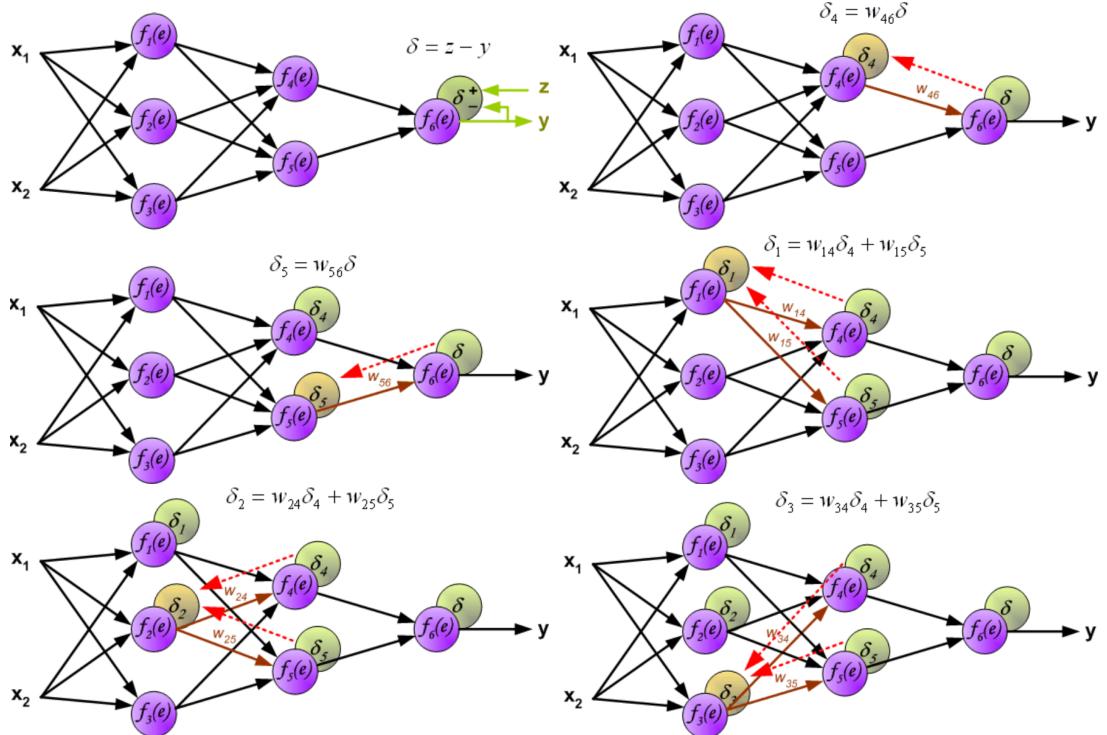


Figura 1.3: Utilizzando la nomenclatura introdotta in questo capitolo si visualizzano i valori di  $\delta_j$  dovuti alla back propagation, rispettando l'effettivo ordine in cui vengono calcolati.

Volendo il più delle volte definire la percentuale di successo della rete e avendo solitamente un set finito  $\{(x_1, \hat{y}_1), \dots, (x_N, \hat{y}_N)\}$ , si sceglie il set di training con  $n < N$  per poter utilizzare gli esempi rimanenti come set di convalida o set di test  $\zeta =$

$\{(x_{n+1}, \hat{y}_{n+1}), \dots, (x_N, \hat{y}_N)\}$ . Applicando la rete su questi nuovi input si riesce quindi a stimare l'efficacia della rete neurale.

# Reti neurali a valori complessi

Le reti neurali a valori complessi sono applicate in sempre piú circostanze. Vengono utilizzate per esempio per rilevare difetti nei metalli analizzando le risposte dei materiali agli ultrasuoni [2], nella “voice processing” [3] e nell’analisi dei segnali sonar per la stima di informazioni su un determinato paesaggio [4].

## 1.4 Storia

Uno dei primi esempi di rete neurale fu il perceptron di Rosenbatt (1958) [1]. Di seguito fu creata ADELINe (Widrow & Hoff, 1960) [5], composta da un singolo layer con un singolo neurone e utilizzava il *LMS* (least mean square) insieme ad un algoritmo di discesa stocastica del gradiente per la configurazione dei pesi. Ne descriviamo adesso le caratteristiche. Siano  $x \in R^N$  gli input e  $w \in R^N$  i pesi; dal modello abbiamo:

$$y = x \cdot w. \quad (1.18)$$

Dato l’output desiderato  $\hat{y} \in R$  possiamo definire la funzione errore  $e$  e la *loss function*  $L$ . L’obiettivo é quello di trovare il valore dei pesi che minimizzi  $L$ .

$$e = \hat{y} - y \quad (1.19)$$

$$L = e^2 \quad (1.20)$$

$$w \leftarrow \arg \min(L). \quad (1.21)$$

L’algoritmo *LMS* ottimizza  $L$  usando il gradiente discendente:

$$\nabla L_w := -2ex \quad (1.22)$$

$$w \leftarrow w + \eta ex \quad (1.23)$$

dove  $\eta$  é il tasso di apprendimento.

Widrow, McCool and Ball (1975) [6] hanno esteso l’algoritmo *LMS* al dominio complesso fornendo la derivazione delle parti reali e immaginarie. Brandwood (1983) [7] generalizzó la teoria applicando il gradiente al numero complesso, senza separarlo in parte reale e parte immaginaria attraverso il gradiente di Wirtinger (1927) [8].

Aggiorniamo quindi il problema nel dominio complesso:

$$L = e\bar{e} \quad (1.24)$$

$$\nabla L_w := -2e\bar{x} \quad (1.25)$$

$$w \leftarrow w + \eta e\bar{x}. \quad (1.26)$$

Nonostante i risultati di Brandwood, fino a non molto tempo fa la letteratura non applicava il calcolo di Wirtinger, a favore della derivazione separata della parte reale da quella immaginaria.

## 1.5 Struttura

Esaminiamo un modello *feedforward* avente un unico hidden layer; abbiamo di conseguenza:

$$h = f(W^{(h)}x + b^{(h)}) \quad (1.27)$$

$$y = W^{(o)}x + b^{(o)} \quad (1.28)$$

con  $\theta = \{W^{(h)}, W^{(o)}, b^{(h)}, b^{(o)}\}$  che rappresenta i parametri del modello. Definendo  $M$  il numero dei nodi dell'input e  $N$  il numero dei nodi dell'output, avremo:

$$\begin{array}{ll} W^{(h)} \in R^{M \times M} & \text{or} \quad W^{(o)} \in C^{M \times M} \\ b^{(h)} \in R^M & \text{or} \quad b^{(o)} \in C^M \\ W^{(o)} \in R^{N \times M} & \text{or} \quad W^{(o)} \in C^{N \times M} \\ b^{(o)} \in R^N & \text{or} \quad b^{(o)} \in C^N \end{array} \quad (1.29)$$

In tutti i casi gli input, i pesi e gli output appartengono allo stesso dominio numerico.

## 1.6 Funzione d'attivazione

Bisogna ricordare che la funzione d'attivazione deve essere una funzione  $f(x) : C \rightarrow C$  e un aspetto fondamentale che interessa la back-propagation è la derivabilità. Tenendo conto di ciò, riporto di seguito le funzioni d'attivazione che ho trovato essere quelle più utilizzate per reti neurali a valori complessi.

**Identità** Permette una modellizzazione lineare

$$f^{(-)}(x) := x \quad (1.30)$$

**Tangente Iperbolica** È una funzione sigmoidale e differenziabile. Permette una non linearità ampiamente utilizzata per l'apprendimento delle reti neurali

$$f^{(\sigma)} := \tanh(x) \quad (1.31)$$

**Split reale-immaginario** Applica la tangente iperbolica separatamente alla parte reale e alla parte immaginaria:

$$f^{(ri)}(x) := \tanh(\operatorname{Re} x) + i \tanh(\operatorname{Im} x) \quad (1.32)$$

**Split ampiezza-fase** Applica la tangente iperbolica al modulo del numero complesso, senza modificarne la fase (questa funzione non è differenziabile nel campo complesso)

$$f^{(ap)}(x) := \tanh(|x|) e^{i \arg x} \quad (1.33)$$

**ModReLU** Arjosky nel 2015 propose una variazione alla classica ReLU utilizzata nelle reti neurali reali, definita come segue:

$$\operatorname{ModReLU}(z) = \operatorname{ReLU}(|z| + b) e^{i\theta_z} = \begin{cases} (|z| + b) \frac{z}{|z|} & \text{se } |z| + b \geq 0 \\ 0 & \text{altrimenti} \end{cases} \quad (1.34)$$

dove  $\theta_z$  è la fase di  $z$  e  $b$  il bias, apprendibile dalla rete neurale, e necessario per creare una *dead zone* di raggio  $b$  attorno all'origine dove il neurone è inattivo. Tale funzione non soddisfa le equazioni di Cauchy-Riemann e quindi non è olomorfa.

**CReLU** Consiste in una applicazione della funzione ReLU individualmente alla parte reale e alla parte immaginaria:

$$\operatorname{CReLU}(z) = \operatorname{ReLU}(\operatorname{Re}(z)) + i \operatorname{ReLU}(\operatorname{Im}(z)). \quad (1.35)$$

Questa soddisfa le condizioni di Cauchy-Riemann solo se sia la parte reale che la parte immaginaria sono contemporaneamente strettamente positive o strettamente negative, quindi nell'intervallo  $\theta_z \in (0, \frac{\pi}{2})$  oppure  $\theta_z \in (\pi, \frac{3\pi}{2})$ .

**zReLU** Proposta nel 2016 da Guberman e basata anch'essa sulla ReLU, è definita come:

$$z\operatorname{ReLU}(z) = \begin{cases} z & \text{se } \theta_z \in [0, \frac{\pi}{2}] \\ 0 & \text{altrimenti} \end{cases} \quad (1.36)$$

## 1.7 Back propagation

### 1.7.1 Loss function

La maggior parte della letteratura attuale utilizza come funzione di costo la *mean squared error*. Dato un target  $\hat{y}$  e l'output ottenuto  $y$ , entrambi in  $C^N$  e l'errore:

$$e := \hat{y} - y \quad (1.37)$$

la *complex mean squared loss function* é definita come segue:

$$\begin{aligned} L(e) &= \sum_{i=0}^{N-1} |e_i|^2 \\ &= \sum_{i=0}^{N-1} e_i \bar{e}_i. \end{aligned} \tag{1.38}$$

La 1.38 é una funzione a valori reali scalari non negativi, che tende a zero insieme al modulo dell'errore. Savitha, Suresh e Sundararajan [9] proposero di sostituire l'errore 1.37 con:

$$e := \log(\hat{y}) - \log(y). \tag{1.39}$$

La *loss function* diventerebbe quindi:

$$L(e) = (\log |\hat{y}_i| - \log |y_i|)^2 + (\arg(\hat{y}_i) - \arg(y_i))^2 \tag{1.40}$$

L'equazione 1.40 ha la proprietá di rappresentare esplicitamente l'ampiezza e la fase. Le equazioni 1.37 e 1.39 possono essere *error function* appropriate per reti neurali complesse.

### 1.7.2 Il gradiente complesso ed il calcolo di Wirtinger

Wirtinger (1927) [8] forní un formalismo che rese il calcolo della derivata di funzioni con valori complessi meno oneroso rispetto a funzioni olomorfe e non analitiche, agendo integralmente nel campo complesso. Nonostante tale apparente comoditá, solo recentemente si ricominciò ad utilizzare il calcolo di Wirtinger per la *back-propagation* di reti neurali complesse.

Definiamo:

$$\begin{aligned} f(z) &:= f(z, \bar{z}) \\ &= g(x, y) \\ &= u(x, y) + iv(x, y) \end{aligned} \tag{1.41}$$

con  $z \in C$ ,  $x, y \in R$  e  $z = x + iy$ .

Usando la prima definizione avremo le derivate in  $z$  e  $\bar{z}$  date da:

$$\frac{\partial f}{\partial z} \Bigg|_{\bar{z} \text{ costante}} \tag{1.42}$$

$$\frac{\partial f}{\partial \bar{z}} \Bigg|_{z \text{ costante}} \tag{1.43}$$

le quali, espresse in funzioe di  $x$  e  $y$ , diventano:

$$\frac{\partial f}{\partial z} = \frac{1}{2} \left( \frac{\partial f}{\partial x} - i \frac{\partial f}{\partial y} \right) \quad (1.44)$$

$$\frac{\partial f}{\partial \bar{z}} = \frac{1}{2} \left( \frac{\partial f}{\partial x} + i \frac{\partial f}{\partial y} \right). \quad (1.45)$$

Si osservi che la derivata parziale rispetto a  $\bar{z}$  é nulla per ogni funzione olomorfa. Richiamiamo le condizioni di esistenza di Cauchy-Riemann per la derivata complessa della funzione  $f(z, \bar{z})$  presa in considerazione:

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \quad (1.46)$$

$$\frac{\partial v}{\partial x} = -\frac{\partial u}{\partial y}. \quad (1.47)$$

Se applichiamo le condizioni di Cauchy-Riemann alla derivata di  $f$  in 1.45 notiamo che effettivamente essa si annulla. Le funzioni olomorfe quindi non dipendono esplicitamente da  $\bar{z}$ . Brandwood dimostró che l'annullarsi di 1.44 o 1.45 per una generica  $f : C \rightarrow R$  é condizione sufficiente e necessaria affinché  $f$  abbia un punto stazionario. Per estensione se  $f : C^N \rightarrow R$  é una funzione a valori reali di un certo vettore  $z = [z_0, z_1, \dots, z_{N-1}]^T \in C^N$  e definiamo il gradiente e il gradiente coniugato come:

$$\frac{\partial}{\partial z} := \left[ \frac{\partial}{\partial z_0}, \frac{\partial}{\partial z_1}, \dots, \frac{\partial}{\partial z_{N-1}} \right] \quad (1.48)$$

$$\frac{\partial}{\partial \bar{z}} := \left[ \frac{\partial}{\partial \bar{z}_0}, \frac{\partial}{\partial \bar{z}_1}, \dots, \frac{\partial}{\partial \bar{z}_{N-1}} \right] \quad (1.49)$$

allora  $\frac{\partial f}{\partial z} = 0$  o  $\frac{\partial f}{\partial \bar{z}} = 0$  sono condizioni sufficienti e necessarie per determinare un punto di stazionarietá.

Se  $f$  é una funzione di un vettore complesso  $z$ , la sua derivata totale é:

$$df = \frac{\partial f}{\partial z} dz + \frac{\partial f}{\partial \bar{z}} d\bar{z}. \quad (1.50)$$

Se  $f$  é reale allora avremmo:

$$df = 2Re \left\{ \frac{\partial f}{\partial z} dz \right\}. \quad (1.51)$$

Definendo ora l'operatore gradiente come:

$$\begin{aligned} \nabla_z &:= \left( \frac{\partial}{\partial \bar{z}} \right)^T \\ &= \left( \frac{\partial}{\partial z} \right)^* \end{aligned} \quad (1.52)$$

Puó essere dimostrato, usando la disuguaglianza di Cauchy-Scharz, che  $f$  ha il maggior tasso di cambiamento lungo il gradiente. Grazie a queste definizioni possiamo costruire una *cost function* reale con argomenti complessi anche se alcuni elementi della funzione non sono olomorfi. In generale, date le funzioni arbitrarie  $f$  e  $g$ , combiniamo gli jacobiani come segue

$$J_{f \circ g} = J_f J_g + J_f^{(c)} \overline{J_g^{(c)}} \quad (1.53)$$

$$J_{f \circ g}^{(c)} = J_f J_g^{(c)} + J_f^{(c)} \overline{J_g}. \quad (1.54)$$

Supponiamo di avere una funzione composta  $(f \circ g \circ h)(z, \bar{z})$ , con  $f$  la *cost function* reale,  $g$  una funzione complessa non olomorfa e  $h$  una funzione olomorfa. Tenendo a mente che  $f$  é una funzione a valori reali di variabile complessa e che  $h$  é olomorfa ( $\frac{\partial h}{\partial \bar{z}} = 0$ ), applichiamo la *chain rule*:

$$J_f = \frac{\partial f}{\partial g} \quad (1.55)$$

$$J_f^{(c)} = \frac{\partial f}{\partial \bar{g}} \quad (1.56)$$

$$J_{f \circ g} = J_f \frac{\partial g}{\partial h} + J_f^{(c)} \overline{\left( \frac{\partial g}{\partial \bar{h}} \right)} \quad (1.57)$$

$$J_{f \circ g \circ h} = J_{f \circ g} \frac{\partial h}{\partial z} \quad (1.58)$$

$$\nabla_z f = (J_{f \circ g \circ h})^* \quad (1.59)$$

Il Calcolo di Wirtinger rende un po' piú semplice costruire un grafico computazionale per reti complesse aventi composizioni miste di operazioni olomorfe e non olomorfe.

## 1.8 Confronto con rete neurale a valori reali

Le reti neurali a valori complessi non sono un nuovo concetto, tuttavia, l'uso di modelli a valore reale é stato spesso favorito rispetto a modelli a valore complesso a causa delle difficoltà nella formazione e nelle prestazioni. Infatti, come riprenderò piú avanti, le reti neurali complesse non sempre convengono se comparate alle reti neurali a valori reali, soprattutto in problemi naturalmente nel dominio reale e non complesso. In tal caso infatti introducendo i numeri complessi si ottengono risultati non molto distanti, ma con una complessità di calcolo maggiore.

Il numero di parametri reali é una metrica per quantificare la capacità di una rete di approssimare funzioni strutturalmente complesse. Con troppi parametri il modello tende a overfittare i dati del training mentre con troppi pochi tende ad underfittare. Spesso la letteratura però ignora il numero di parametri, finendo col confrontare reti di dimensioni

notevolmente diverse. Bisogna garantire che queste architetture siano comparabili per dimensioni e capacità del modello. Per definire la dimensione della rete si può considerare il numero di parametri reali. Di conseguenza, per costruire reti comparabili o si imposta un numero fisso di neuroni a valore reale oppure si fissa il numero di parametri reali per ogni rete.

Una conseguenza della rappresentazione di un numero complesso come  $a + ib$  (utilizzando quindi due numeri reali) è dato dal fatto che, a parità di “neuroni”, il numero di parametri  $p_C$  della rete complessa diventa il doppio del numero di parametri  $p_R$  di una rete a valori reali. Per garantire che modelli abbiano le stesse capacità, il numero di parametri a valore reale per layer deve essere uguale o almeno il più simile possibile. In questo modo le differenze di prestazione sono causate dall’introduzione di numeri complessi come parametri e non da una differenza di capacità.

Si consideri il numero di parametri in un layer fully connected nel caso reale e nel caso complesso. Sia  $d$  la dimensione dell’input e  $m$  il numero di neuroni di un layer, di conseguenza il numero di parametri nel caso reale e nell’caso complesso è:

$$p_R = (d \times m) + m, \quad p_C = 2(d \times m) + 2m \quad (1.60)$$

Per una rete neurale con  $k$  hidden layers e output di dimensione  $c$ , il numero di parametri reali (senza contare i bias) diventa:

$$p_R = d \times m + k(m \times m) + m \times c, \quad (1.61)$$

$$p_C = 2[d \times m + k(m \times m) + m \times c] \quad (1.62)$$

A prima vista la progettazione di architetture di reti neurali multistrato comparabili, ovvero con lo stesso numero di parametri a valore reale in ogni layer pare banale. Tuttavia, dimezzare il numero di neuroni in ogni stato non permette il raggiungimento della comparabilità dei modelli. Si puó affrontare questo problema scegliendo delle architetture con un numero di hidden layers  $k$  e un numero di neuroni per layer che puó alternarsi tra  $m$  e  $m/2$ .

Si consideri per esempio le dimensioni degli output e dei pesi con  $k = 4$ . Per il caso reale si ha quindi:

$$\begin{aligned} & \text{Input layer} & \text{Hidden layer} \\ (1 \times d) \overbrace{(d \times m_1)}^{} & \rightarrow (1 \times m_1) \overbrace{(m_1 \times m_2)}^{} \\ & \text{Hidden layer} & \text{Hidden layer} \\ \rightarrow (1 \times m_2) \overbrace{(m_2 \times m_3)}^{} & \rightarrow (1 \times m_3) \overbrace{(m_3 \times m_4)}^{} \\ & \text{Hidden layer} & \text{Hidden layer} \\ \rightarrow (1 \times m_4) \overbrace{(m_4 \times m_5)}^{} & \rightarrow (1 \times m_5) \overbrace{(m_5 \times c)}^{} \\ & \text{Model output} \\ \rightarrow \overbrace{(1 \times c)}^{} & \end{aligned} \quad (1.63)$$

Mentre per il caso complesso:

$$\begin{aligned}
 (1 \times n)(n \times m_1/2) &\rightarrow (1 \times m_1/2)(m_1/2 \times m_2) \\
 \rightarrow (1 \times m_2)(m_2 \times m_3/2) &\rightarrow (1 \times m_3/2)(m_3/2 \times m_4) \\
 \rightarrow (1 \times m_4)(m_4 \times m_5/2) &\rightarrow (1 \times m_5/2)(m_5/2 \times c) \\
 \rightarrow (1 \times c)
 \end{aligned} \tag{1.64}$$

Un ulteriore approccio é quello di lavorare piú in generale con un budget di parametri. Quindi dato un numero massimo di parametri reali  $p_R$ , si definisce una rete neurale a valori reali o complessi con un numero di hidden layer  $k \geq 0$  tale da rimanere nel budget. I  $k$  hidden layer hanno lo stesso numero di neuroni, reali o complessi che siano :  $m_R = m_C$ . Il numero di neuroni nell'ultimo layer é definito dal numero di classi  $c$ .

$$m_R = \begin{cases} -\frac{n+c}{2k} + \sqrt{\frac{n+c}{2k}^2 + \frac{p_R}{k}}, & \text{se } k > 0 \\ \frac{p_R}{n+c}, & \text{altrimenti} \end{cases} \tag{1.65}$$

$$m_C = \begin{cases} -\frac{n+c}{2k} + \sqrt{\frac{n+c}{2k}^2 + \frac{p_R}{k}}, & \text{se } k > 0 \\ \frac{2(p_R)}{n+c}, & \text{altrimenti} \end{cases} \tag{1.66}$$

Confrontando reti neurali reali e complesse di capacità simile, i modelli complessi hanno prestazioni solitamente uguali o leggermente peggiori rispetto ai modelli a valore reale, se vengono considerati compiti di classificazione nel dominio reale. Tuttavia le reti neurali a valore complesso sono state applicate con successo a una varietá di compiti, in particolare nell'elaborazione del segnale in cui i dati di input hanno una interpretazione fisica nel dominio complesso[2] [3] [4] [10] [11] [12].

Per confrontare reti neurali a valori reali con quelle a valori complessi, Nils Mönning e Suresh Manandhar in *Evaluation of Complex–Valued on Real–Valued Classification Tasks* [13] applicano entrambe le reti a problemi di classificazione comuni. Gli esperimenti eseguiti sono due:

- MLP con numero di hidden layers  $k = 0, 2, 4, 8$ , dimensione degli hidden layer fissa per le reti a valori reali e alternante (64 o 32 neuroni) per le reti a valori complessi. Non é stato definito un budget di parametri. Il test é stato fatto sul problema di classificazione MINST, CIFAR-10, CIFAR-100 e Reuters.
- MLP con budget fisso di 500000 parametri reali. Le dimensioni sono quindi variabili e saranno indicate di volta in volta. Vengono scelti gli stessi problemi di classificazione citati per il caso precedente. Il numero di neuroni dell'ultimo layer é definito dalle equazioni 1.65 e 1.66 e viene arrotondato per eccesso.

É stata utilizzata l'inizializzazione dei pesi discussa da Trabelsi et al. [12] per tutti gli esperimenti. Per ridurre l'impatto dell'inizializzazione, ogni modello é stato addestrato 10 volte, con training composte da 100 epochs.

Le tabelle 1.1,1.2,1.3,1.4 mostrano i risultati per reti neurali con dimensione variabile e budget illimitato per i parametri (esperimento 1). Le tabelle 1.5,1.6,1.7,1.8 mostrano invece i risultati dell'esperimento 2.

Hidden layers $k$	Real parameters $p_R$	Activation function $\phi$	MNIST	
			R	C
k=0	50816	<i>identity</i>	0.9282	0.9509
		<i>tanh</i>	0.9761	0.9551
		<i>ReLU</i>	0.9780	0.9710
		$ z ^2$	0.9789	0.9609
		$ z $	0.9770	0.9746
		<i>identity</i>	0.9274	0.9482
k=2	59008	<i>tanh</i>	0.9795	0.8923
		<i>ReLU</i>	0.9804	0.9742
		$ z ^2$	0.9713	0.6573
		$ z $	0.9804	0.9755
		<i>identity</i>	0.9509	0.9468
		<i>tanh</i>	0.9802	0.2112
k=4	67200	<i>ReLU</i>	0.9816	0.9768
		$ z ^2$	0.8600	0.2572
		$ z $	0.9789	0.9738
		<i>identity</i>	0.9242	0.1771
		<i>tanh</i>	0.9796	0.1596
		<i>ReLU</i>	0.9798	0.9760
k=8	83584	$ z ^2$	0.0980	0.0980
		$ z $	0.9794	0.1032

Tabella 1.1: Tratto da [13]. Test di accuratezza di reti con  $k + 2$  layer, ognuno con 64 neuroni (alternando 32 e 64 neuroni nei layer della rete a valori complessi) e un output con  $c = 10$  neuroni. Esperimento 1.

Hidden layers $k$	Real parameters $p_R$	Activation function $\phi$	Reuters	
			R	C
k=0	642944	<i>identity</i>	0.8116	0.7936
		<i>tanh</i>	0.8117	0.7912
		<i>ReLU</i>	0.8081	0.7934
		$ z ^2$	0.8050	0.7885
		$ z $	0.8068	0.7992
		<i>identity</i>	0.8005	0.7836
k=2	651136	<i>tanh</i>	0.7978	0.7320
		<i>ReLU</i>	0.7921	0.7854
		$ z ^2$	0.7725	0.6874
		$ z $	0.7996	0.7823
k=4	659328	<i>identity</i>	0.7925	0.7787
		<i>tanh</i>	0.7814	0.4199
		<i>ReLU</i>	0.7734	0.7671
		$ z ^2$	0.5895	0.0650
k=8	675712	$ z $	0.7863	0.7694
		<i>identity</i>	0.7929	0.7796
		<i>tanh</i>	0.7542	0.1861
		<i>ReLU</i>	0.7555	0.7676
		$ z ^2$	0.0053	0.0053
		$ z $	0.7671	0.7524

Tabella 1.2: Tratto da [13]. Test di accuratezza di reti con  $k + 2$  layer, ognuno con 64 neuroni (alternando 32 e 64 neuroni nei layer della rete a valori complessi) e un output con  $c = 46$  neuroni. Esperimento 1.

Hidden layers $k$	Real parameters $p_R$	Activation function $\phi$	MNIST	
			R	C
k=0	394496	<i>identity</i>	0.4044	0.1063
		<i>tanh</i>	0.4885	0.1431
		<i>ReLU</i>	0.4902	0.4408
		$ z ^2$	0.5206	0.1000
		$ z $	0.5256	0.1720
		<i>identity</i>	0.4039	0.1000
k=2	427264	<i>tanh</i>	0.5049	0.1672
		<i>ReLU</i>	0.5188	0.4960
		$ z ^2$	0.1451	0.1361
		$ z $	0.5294	0.1000
k=4	460032	<i>identity</i>	0.4049	0.1000
		<i>tanh</i>	0.4983	0.1549
		<i>ReLU</i>	0.8445	0.6810
		$ z ^2$	0.1000	0.1000
k=8	525568	$ z $	0.5273	0.1000
		<i>identity</i>	0.4005	0.1027
		<i>tanh</i>	0.4943	0.1365
		<i>ReLU</i>	0.5072	0.4939
		$ z ^2$	0.1000	0.1000
		$ z $	0.5276	0.1000

Tabella 1.3: Tratto da [13]. Test di accuratezza di reti con  $k + 2$  layer, ognuno con 128 neuroni (alternando 128 e 64 neuroni nei layer della rete a valori complessi) e un output con  $c = 10$  neuroni. Esperimento 1.

Hidden layers $k$	Real parameters $p_R$	Activation function $\phi$	CIFAR-100	
			R	C
k=0	406016	<i>identity</i>	0.1758	0.0182
		<i>tanh</i>	0.2174	0.0142
		<i>ReLU</i>	0.1973	0.1793
		$ z ^2$	0.2314	0.0158
		$ z $	0.2423	0.0235
		<i>identity</i>	0.1720	0.0100
k=2	438784	<i>tanh</i>	0.2314	0.0146
		<i>ReLU</i>	0.2400	0.2123
		$ z ^2$	0.0143	0.0123
		$ z $	0.2411	0.0100
k=4	471552	<i>identity</i>	0.1685	0.0100
		<i>tanh</i>	0.2178	0.0157
		<i>ReLU</i>	0.2283	0.2059
		$ z ^2$	0.0109	0.0100
k=8	537088	$ z $	0.2313	0.0100
		<i>identity</i>	0.1677	0.0100
		<i>tanh</i>	0.2000	0.0130
		<i>ReLU</i>	0.2111	0.1956
		$ z ^2$	0.0100	0.0100
		$ z $	0.2223	0.0100

Tabella 1.4: Tratto da [13]. Test di accuratezza di reti con  $k + 2$  layer, ognuno con 64 neuroni (alternando 32 e 64 neuroni nei layer della rete a valori complessi) e un output con  $c = 100$  neuroni. Esperimento 1

Hidden layers $k$		Units	$p_R$	Activation function $\phi$	MNIST	
					R	C
k=0	162	81		<i>identity</i>	0.4335	0.1006
				<i>tanh</i>	0.5032	0.1676
				<i>ReLU</i>	0.5007	0.4554
				$ z ^2$	0.5179	0.1006
				$ z $	0.5263	0.2381
				<i>identity</i>	0.4069	0.1000
k=2	148	77		<i>tanh</i>	0.5205	0.1673
				<i>ReLU</i>	0.5269	0.4963
				$ z ^2$	0.1395	0.1273
				$ z $	0.5315	0.1000
				<i>identity</i>	0.4052	0.1000
				<i>tanh</i>	0.5218	0.1475
k=4	138	74		<i>ReLU</i>	0.5203	0.4975
				$ z ^2$	0.1065	0.1010
				$ z $	0.5234	0.1000
				<i>identity</i>	0.4050	0.1003
				<i>tanh</i>	0.5162	0.1396
				<i>ReLU</i>	0.5088	0.4926
k=8	123	69		$ z ^2$	0.1000	0.1000
				$ z $	0.5194	0.1000

Tabella 1.5: Tratto da [13]. Test di accuratezza di reti con  $k + 2$  layer, con un budget di 500000 parametri reali. L'output è composto da  $c = 10$  neuroni. Esperimento 2.

Hidden layers $k$		Units $p_R$	$m_C$	Activation function $\phi$	Reuters	
					R	C
k=0	50	25	<i>identity</i>	0.8072	0.7970	
			<i>tanh</i>	0.8112	0.7832	
			<i>ReLU</i>	0.8054	0.7925	
			$ z ^2$	0.8037	0.7929	
			$ z $	0.8059	0.7912	
			<i>identity</i>	0.7992	0.7809	
k=2	49	25	<i>tanh</i>	0.7952	0.7289	
			<i>ReLU</i>	0.7898	0.7751	
			$ z ^2$	0.7778	0.6887	
			$ z $	0.7716	0.7911	
			<i>identity</i>	0.7636	0.7854	
			<i>tanh</i>	0.7996	0.4550	
k=4	4925	25	<i>ReLU</i>	0.7658	0.7676	
			$ z ^2$	0.5823	0.0289	
			$ z $	0.7809	0.7573	
			<i>identity</i>	0.7760	0.7663	
			<i>tanh</i>	0.7449	0.1799	
			<i>ReLU</i>	0.7182	0.7484	
k=8	48	24	$ z ^2$	0.0053	0.0053	
			$ z $	0.7449	0.7302	

Tabella 1.6: Tratto da [13]. Test di accuratezza di reti con  $k + 2$  layer, con un budget di 500000 parametri reali. L'output è composto da  $c = 46$  neuroni. Esperimento 2.

Hidden layers $k$		Units $p_R$	$m_C$	Activation function $\phi$	CIFAR-10	
	$m_R$				R	C
k=0	630	315		<i>identity</i>	0.9269	0.9464
				<i>tanh</i>	0.9843	0.9467
				<i>ReLU</i>	0.9846	0.9828
				$ z ^2$	0.9843	0.9654
				$ z $	0.9857	0.9780
				<i>identity</i>	0.9261	0.9427
k=2	339	207		<i>tanh</i>	0.9852	0.6608
				<i>ReLU</i>	0.9878	0.9835
				$ z ^2$	0.9738	0.9331
				$ z $	0.9852	0.9748
				<i>identity</i>	0.9254	0.2943
				<i>tanh</i>	0.9838	0.2002
k=4	268	170		<i>ReLU</i>	0.9862	0.9825
				$ z ^2$	0.8895	0.2875
				$ z $	0.9846	0.9870
				<i>identity</i>	0.9250	0.1136
				<i>tanh</i>	0.9810	0.1682
				<i>ReLU</i>	0.9851	0.9824
k=8	205	134		$ z ^2$	0.0980	0.0980
				$ z $	0.9803	0.1135

Tabella 1.7: Tratto da [13]. Test di accuratezza di reti con  $k + 2$  layer, con un budget di 50000 parametri reali. L'output è composto da  $c = 10$  neuroni. Esperimento 2.

Hidden layers $k$		Units $p_R$	$m_C$	Activation function $\phi$	CIFAR-100	
	$m_R$				R	C
k=0	158	79	$ReLU$	<i>identity</i>	0.2807	0.0314
				<i>tanh</i>	0.2308	0.0193
				$ z ^2$	0.2364	0.0124
				$ z $	0.2439	0.0279
				<i>identity</i>	0.1723	0.0100
			$tanh$	<i>tanh</i>	0.2440	0.0203
				$ReLU$	0.2481	0.2224
				$ z ^2$	0.0155	0.0151
				$ z $	0.2453	0.0100
k=2	144	75	$ReLU$	<i>identity</i>	0.1727	0.0100
				<i>tanh</i>	0.2397	0.0150
				$ z ^2$	0.0122	0.0100
				$ z $	0.2390	0.0100
			$tanh$	<i>identity</i>	0.1706	0.0100
				$ReLU$	0.2381	0.2147
				$ z ^2$	0.0122	0.0100
				$ z $	0.2390	0.0100
k=4	135	72	$ReLU$	<i>identity</i>	0.1706	0.0100
				<i>tanh</i>	0.2209	0.0164
				$ z ^2$	0.0100	0.0100
			$tanh$	$ReLU$	0.2167	0.2027
				$ z $	0.2191	0.0100
k=8	121	67	$ReLU$	<i>identity</i>	0.1706	0.0100
				<i>tanh</i>	0.2209	0.0164
				$ z ^2$	0.0100	0.0100

Tabella 1.8: Tratto da [13]. Test di accuratezza di reti con  $k + 2$  layer, con un budget di 50000 parametri reali. L'output è composto da  $c = 100$  neuroni. Esperimento 2.

Confrontando i diversi risultati ottenuti con le funzioni di attivazione si nota come la funzione identità, ma anche soprattutto la tangente iperbolica, risultino meno performanti nelle reti a valore complesso piuttosto che in quelle a valore reale, mentre la ReLU mantiene una accuratezza molto simile indipendentemente dal modello.

Questi dati confermano l'affermazione precedente per la quale le reti neurali a valori complessi hanno prestazioni simili o leggermente inferiori nei campi di applicazione nel dominio reale.

È difficile trovare un confronto così accurato come quello precedente per problemi nel dominio complesso, ma sono presenti esempi in cui la rete neurale a valori complessi è risultata più efficiente di quella a valori reali. Riporto di seguito i dati trovati, facendo riferimento nelle descrizioni delle tabelle all'esperimento e all'articolo preso in considerazione. Per comodità riporto solo i dati relativi a reti neurali a valori reali e a valori complessi, tralasciando quelli relativi ad altri modelli.

In [12] vengono confrontate reti neurali a valori reali e a valori complessi sul problema di trascrizione musicale MusicNet. In particolare la rete neurale a valori reali tratta il segnale separando la componente reale da quella immaginaria. I risultati di tale esperimento sono riportati in tabella 1.9.

ARCHITECTURE	FS	PARAMS	AP
DEEP, REAL	11kHz	10.0M	69.6%
DEEP, COMPLEX	11kHz	8.8M	<b>72.9%</b>

Tabella 1.9: Tratto da [12]. Dati relativi all'esperimento MusicNet. FS é il rate di campionamento; params indica il numero di parametri e AP indica la media dei risultati ottenuti.

Sempre in [12] viene portato avanti un esperimento sullo “speech spectrum prediction”. Il dataset utilizzato é il TIMIT. Come in precedenza per le reti reali il segnale é stato scomposto in parte reale e parte immaginaria. I risultati sono mostrati in tabella 1.10

MODEL	#PARAMS	MSE (VALIDATION)	MSE (TEST)
CONV-LSTM	$\approx 88K$	11.10	12.18
CCONV-LSTM	$\approx 88K$	<b>10.78</b>	<b>11.90</b>

Tabella 1.10: Tratto da [12]. Per i vari modelli di rete vengono confrontati il numero di parametri e l'errore quadratico medio (MSE), relativo sia al training che al test. In grassetto i risultati migliori. Per CNN si intendono le reti neurali convoluzionari, analoghe a quelle descritte, ma che prevedono una struttura iniziale aggiuntiva.

In [14] il problema affrontato é quello dell'imaging computazionale. In tabella 1.11 vengono riportati i risultati ottenuti in funzione del modello e del SVR (rapporto segnale rumore) applicato.

Methods	RMSE, -10dB	RMSE, -5dB	RMSE, 0dB	RMSE, 5dB	RMSE, 10dB	Time needs
RV-CNN	0.0456	0.0308	0.0262	0.0251	0.0248	0.083s
CV-CNN	<b>0.0434</b>	<b>0.0289</b>	<b>0.0255</b>	<b>0.0247</b>	<b>0.0245</b>	0.071s

Tabella 1.11: Tratto da [14]. Confronto tra la media degli errori reti su un totale di 100 esperimenti.

In [15] le reti neurali sono confrontate sul dataset di immagini MNIST e la rete neurale a valori reali ha input di dimensione doppia in quanto riceve la parte reale e la parte

Model	Test Accuracy
Real-valued NN	(93.39±0.10)%
CVNN (ModReLU)	(95.92 ± 0.18)%
CVNN (Proposed split-KAF)	<b>(97.21 ± 0.34)%</b>

Tabella 1.12: Tratto da [15]. Confronto tra reti neurali a valori reali e a valori complessi sul dataset MNIST. Nel caso citato gli autori hanno testato una particolare funzione d’attivazione proposta da loro e presente nell’ultima riga. Di interesse é però il vantaggio che comunque pare avere la rete neurale complessa (seconda riga) su quella reale.

immaginaria del segnale come due elementi distinti dell’input. I risultati sono mostrati in tabella 1.12.

Da notare come effettivamente il dominio dell’input analizzato dalle reti neurali nei casi appena citati sia sempre di natura ondulatoria e di conseguenza complessa. Per questo motivo, come già anticipato in precedenza, in questo documento si propone di applicare le reti neurali a valori complessi per il MRI fingerprinting, che verrá trattato nella sezione 1.10 piú approfonditamente.

Per completezza voglio citare un ultimo esempio, un ultimo confronto, significativo non tanto per i risultati relativi all’accuratezza, ma per i risultati mostrati relativi alla generalizzazione e alla convergenza dei modelli.

In [11] vengono utilizzate reti neurali convoluzionali a valori complessi per l’identificazione di cellule fluorescenti come in figura 1.4.



Figura 1.4: Tratto da [11]. Immagini simulate al computer di cellule fluorescenti che la rete neurale convoluzionale a valori complessi deve identificare.

I risultati ottenuti sono molto buoni e promettenti: infatti anche se l’accuratezza sul set di test é praticamente uguale tra rete a valori reali e rete a valori complessi (tabella ??), le reti a valori complessi soffrono meno di overfitting.

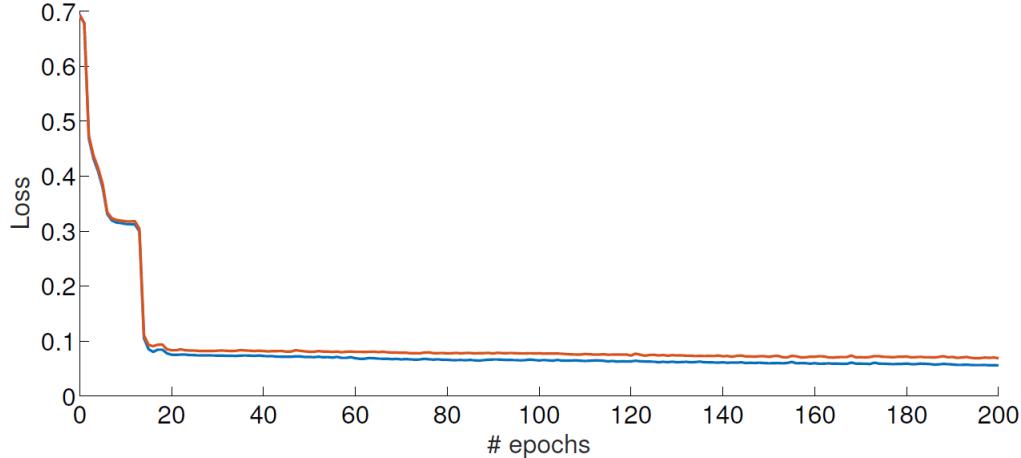
	Train loss	Train accuracy	Test loss	Test Accuracy
Complex network	0.056	97.4%	0.069	97.3%
Real network	0.007	99.8%	0.145	97.5%

Tabella 1.13: Tratto da [11]. Risultati del modello reale e di quello complesso nel set di training e di test. Si noti come il test loss della rete reale sia decisamente maggiore rispetto a quello della rete complessa. Maggiore è il test loss, minore è la capacità di generalizzazione della rete.

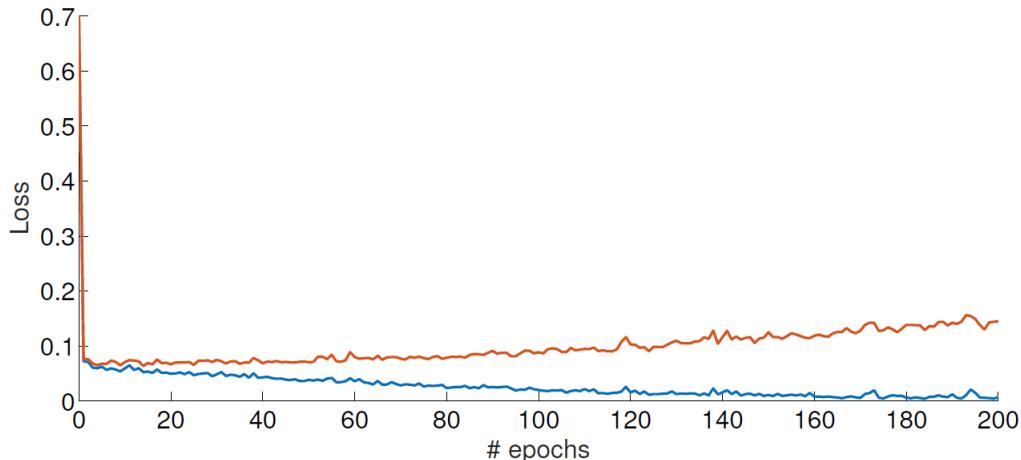
Ciò viene mostrato ancora meglio nel confronto effettuato nei grafici ?? e ?? della convergenza delle due reti in funzione del numero delle epochs (per ogni epoch viene effettuato un determinato numero di cicli di train e di test per determinare l'accuratezza).

Bisogna tenere a mente che in [11] viene esplicitata la scelta di confrontare solo i risultati migliori e suppongo che anche per gli altri dati ottenuti sia stato fatto lo stesso. Però in [11] si ha la possibilità di capire cosa significhi questa scelta: in figura ?? viene mostrato come solo 4 volte su 20 la rete riesce a raggiungere la convergenza desiderata.

Per poter migliorare questo aspetto, di seguito trattato metodi per poter migliorare la resistenza all'overfitting, ma, ancor più importante, migliorare la generalizzazione e la ricerca del minimo globale della rete neurale.



(a) Tratto da [11]. Convergenza della rete neurale a valori complessi.



(b) Tratto da [11]. Convergenza della rete neurale a valori reali.

Figura 1.5: Convergenza delle reti reali e complesse in funzione dei progressi dell'algoritmo. Una epoch é il numero di iterazioni in cui il numero totale di esempi scelti é uguale alla dimensione dell'insieme di addestramento, in questo caso una epoch é di 100 iterazioni. Nella linea blu, la funzione di loss del training e nella linea rossa quella del test. Il modello reale soffre di overfitting, mentre il complesso sembra soffrirne meno.

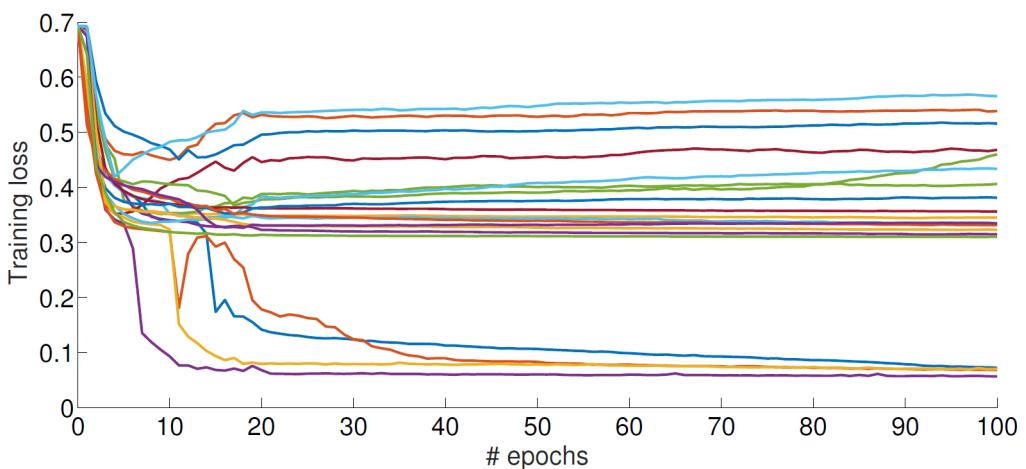


Figura 1.6: Tratto da [11]. Training ripetuto della rete complessa. Ogni riga rappresenta il training loss in tutte le epochs di una singola prova. Il tasso di apprendimento è ridotto dopo 20 epochs, per una convergenza ottimale. Non tutte le prove sono convergenti e, tra queste, la maggior parte non ha raggiunto il minimo globale.

# Ottimizzazione

## 1.9 ensemble di reti neurali

Una rete di dimensioni finite raramente apprende completamente un particolare mapping e può generalizzare male. Aumentare la dimensione o il numero di hidden layer però, il più delle volte, non porta a nessun miglioramento [16] anzi, aumentando la dimensionalità del problema aumenta la possibilità che la rete si stabilizzi lontana dal minimo globale.

Molti ricercatori hanno dimostrato che la semplice combinazione degli output di molti classificatori può generare previsioni più accurate di quelle di qualsiasi classificatore [17] [18]. In particolare, la combinazione di reti neurali addestrate separatamente (comunemente indicato come ensemble di reti neurali) si è dimostrata particolarmente efficace [19] [20] [21] [22] [23]. Figura 1.7 illustra la struttura generale di un ensemble di reti neurali. Ogni rete nell'ensemble è per prima cosa addestrata e poi, per ogni esempio del training, l'output predetto di ognuna delle reti è combinato dal judge per produrre l'output dell'ensemble.

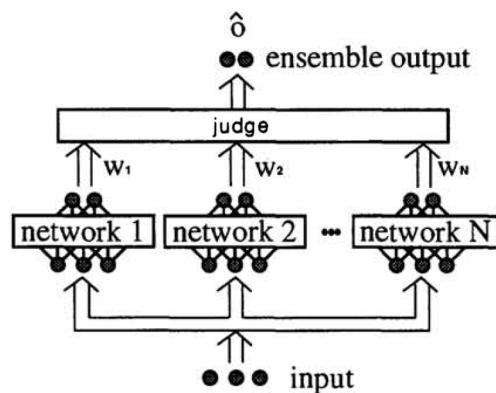


Figura 1.7: Struttura di un ensemble di reti neurali. In basso l'input in comune per tutte le reti. Gli output di queste ultime viene poi combinato dal judge attraverso i pesi  $w_1, \dots, w_N$ .

L'output dell'ensemble puó anche essere utilizzato come ulteriore training restituendolo alle singole reti. Infatti, nel caso in cui nell'ensemble vi siano una o piú reti poco efficienti e non si hanno a disposizione piú esempi per il training, si puó utilizzare l'output dell'ensemble come output desiderato nelle singole reti per continuare l'addestramento [24].

L'errore di una singola rete dipende dalle dimensioni dell'hidden layer e dalla durata del training. Tuttavia in generale l'errore non é una funzione decrescente della dimensione dell'hidden layer. In figure ?? e ?? si mostra l'errore di una singola rete rispetto alla dimensione dell'hidden layer riscontrato da [24] per diversi step di training.

D'altra parte la miglior capacità di apprendimento di un ensemble dipende, come accennato in precedenza, dalla sinergia tra le singole reti e non dalle maggiori dimensioni rispetto alla singola rete. In [24] vengono confrontate le prestazioni di ogni singola rete con le prestazioni di un ensemble di 5 reti neurali, con stessa struttura della singola.

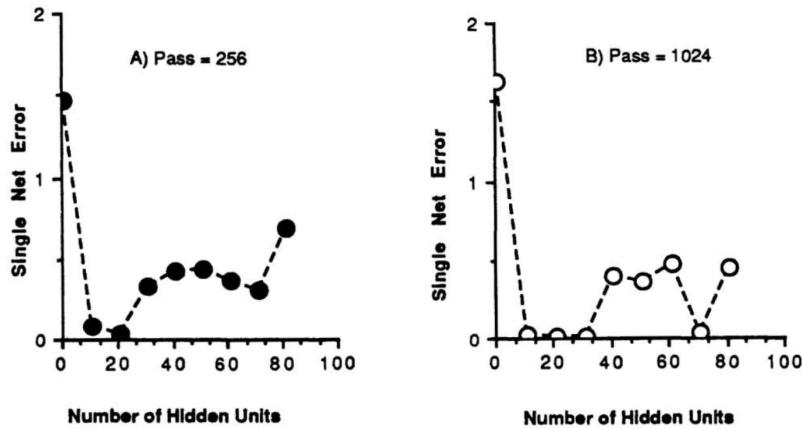


Figura 1.8: Tratto da [24]. Errore della singola rete in funzione del numero di hidden layer, in dimostrazione del fatto che la relazione non é lineare.

Vengono confrontati gli errori medi dei due sistemi. Figura ?? e ?? mostrano il vantaggio dell'ensemble rispetto agli errori della rete singola rispettivamente per 256 e 1024 step del training. Nei casi estremi in cui l'apprendimento sia nullo o quasi totale, l'ensemble non mostra particolari vantaggi, tuttavia si ha un notevole vantaggio se l'apprendimento é solo parziale.

### 1.9.1 judge

Dato un singolo input, le reti neurali dell'ensemble daranno presumibilmente degli output  $Y_k$  diversi, che diventano l'input del *judge*. Quest'ultimo combinandoli ottiene il risultato

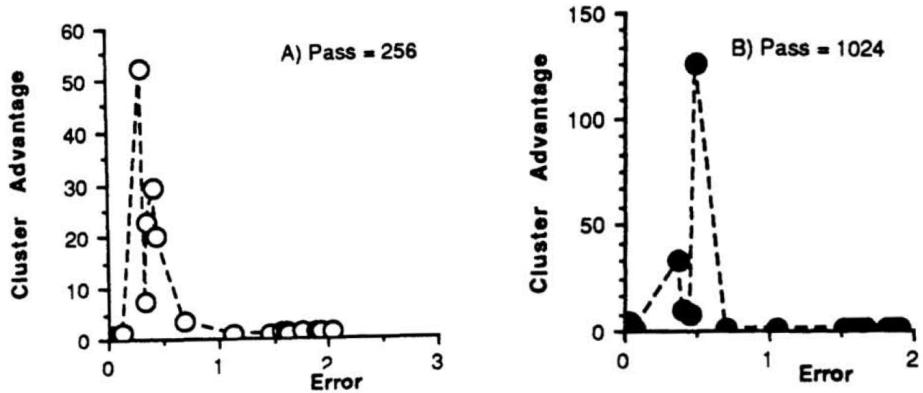


Figura 1.9: Tratto da [24]. Vantaggio dell’ensemble in funzione dell’errore. A) dopo 256 step del training e B) dopo 1024 step del training.

finale  $Y$ . Esistono diversi metodi per calcolare  $Y$ , ma per semplicità consideriamo i due casi più semplici:

$$Y = \sum_{k=1}^N \frac{1}{N} Y_k \quad (1.67)$$

$$Y = \sum_{k=1}^N W_k Y_k \quad (1.68)$$

Il primo è dato da una semplice media dei  $Y_k$ , mentre nel secondo caso si considera una media pesata, con i  $W_k$  che indicano “l’affidabilità” delle singole reti neurali. Seguendo questo ragionamento i  $W_k$  vengono di volta in volta modificati come segue:

$$W_k = W_k \cdot G \cdot \frac{e}{e_k} \quad (1.69)$$

$$e = \frac{1}{N} \sum_{k=1}^N |Y - Y_k| \quad (1.70)$$

$$e_k = |Y - Y_k| \quad (1.71)$$

Con  $G$  che indica il tasso di correzione e  $e_k$  è la deviazione dell’output della singola rete dall’output dell’ensemble.

Dopo il periodo di training iniziale, si presume generalmente che la rete neurale non sia più soggetta ad ulteriore training. Tuttavia, come già anticipato, si può restituire l’output del judge alle singole reti neurali come output desiderato. William P. Lincoln e Josef Skrzypek in *Synergy of clustering multiple back propagation networks* affermano che questo procedimento migliora la resistenza al rumore.

## 1.9.2 ottimizzazione dell'ensemble

Precedenti lavori sia teorici [25] [21] che empirici [26] [22] hanno dimostrato che un ensemble efficace dovrebbe consistere non solo in reti particolarmente performanti, ma anche in reti che commettano errori in parti distinte dello spazio degli input.

Assumiamo che l'obiettivo sia quello di apprendere una funzione  $f : R^N \rightarrow R$  per la quale abbiamo  $p$  coppie input-output per il training con  $Y_k = f(x_k)$  e  $k = 1, \dots, p$ . L'ensemble consiste in  $N$  reti e chiamiamo  $V^\alpha(x)$  l'output della rete  $\alpha$  relativo all'input  $x$ . Definiamo nel modo seguente il valore ottenuto dal judge:

$$\bar{V}(x) = \sum_{\alpha} w_{\alpha} V^{\alpha}(x). \quad (1.72)$$

Consideriamo i pesi  $w_{\alpha}$  come l'affidabilità della rete  $\alpha$  e di conseguenza li vincoliamo ad essere positivi e normalizzati, cioè  $\sum_{\alpha} w_{\alpha} = 1$ . L'*ambiguità* su un input  $x$  di un singolo membro dell'ensemble è definito come  $a^{\alpha}(x) = (V^{\alpha}(x) - \bar{V}(x))^2$ . La *ambiguità dell'ensemble* su un input  $x$  è data da:

$$\bar{a}(x) = \sum_{\alpha} w_{\alpha} a^{\alpha}(x) = \sum_{\alpha} w_{\alpha} (V^{\alpha}(x) - \bar{V}(x))^2. \quad (1.73)$$

È quindi una varianza e misura il disaccordo delle reti sull'input  $x$ . L'errore quadratico della rete  $\alpha$  e dell'ensemble sono rispettivamente:

$$e^{\alpha}(x) = (f(x) - V^{\alpha}(x))^2 \quad (1.74)$$

$$e(x) = (f(x) - \bar{V}(x))^2 \quad (1.75)$$

Aggiungendo e sottraendo  $f(x)$  alla 1.73 si ottiene:

$$\bar{a}(x) = \sum_{\alpha} w_{\alpha} e^{\alpha}(x) - e(x). \quad (1.76)$$

Chiamando la media pesata degli errori delle singole reti  $\bar{e}(x) = \sum_{\alpha} w_{\alpha} e^{\alpha}(x)$  allora la 1.75 diventa:

$$e(x) = \bar{e}(x) - \bar{a}(x). \quad (1.77)$$

Queste ultime formule possono essere mediate lungo la distribuzione degli input, ottenendo le seguenti:

$$E^{\alpha} = \int dx p(x) e^{\alpha}(x) \quad (1.78)$$

$$A^{\alpha} = \int dx p(x) a^{\alpha}(x) \quad (1.79)$$

$$E = \int dx p(x) e(x). \quad (1.80)$$

Le prime due sono rispettivamente l'errore di generalizzazione e l'ambiguitá della rete  $\alpha$  ed  $E$  é l'errore di generalizzazione dell'ensemble. Dalla 1.77 otteniamo:

$$E = \bar{E} - \bar{A} \quad (1.81)$$

con  $\bar{E} = \sum_{\alpha} w_{\alpha} E^{\alpha}$  la media pesata degli errori generalizzati delle singole reti e  $\bar{A} = \sum_{\alpha} w_{\alpha} A^{\alpha}$  la media pesata delle ambiguitá delle singole reti.

Questa equazione separa l'errore di generalizzazione in un termine che dipende dagli errori di generalizzazione delle singole reti e da un altro contenente le correlazioni tra le reti. Inoltre, il termine di correlazione  $A$  puó essere stimato interamente da dati non etichettati, non é richiesta alcuna conoscenza della funzione da approssimare. Il termine “senza etichetta” é tratto dai problemi di classificazione e in questo contesto si riferisce a un input  $x$  per il quale non si conosce il valore  $f(x)$  della funzione target. Se l'ensemble é fortemente distorto, l'ambiguitá sará piccola, perché le reti implementano funzioni molto simili e concordano quindi gli input anche al di fuori del training. Pertanto l'errore di generalizzazione sará sostanzialmente uguale alla media degli errori delle singole reti. Se, d'altra parte, c'é una grande varianza, l'ambiguitá é alta e in questo caso l'errore di generalizzazione sará piú piccolo dell'errore di generalizzazione medio. Vediamo immediatamente che l'errore di generalizzazione dell'ensemble é sempre minore della media pesata degli errori degli ensemble:  $E < \bar{E}$ . In particolare, per pesi uniformi:

$$E \leq \frac{1}{N} \sum_{\alpha} E^{\alpha} \quad (1.82)$$

Dalla 1.81 si ricava che aumentare l'efficienza dell'ensemble significa aumentare l'ambiguitá e quindi la discordanza tra le reti singole senza aumentarne ovviamente l'errore di generalizzazione. Come aumentare l'ambiguitá dell'ensemble? Un metodo puó essere quello di utilizzare diverse tipologie di reti neurali, oppure di utilizzare set di training diversi. Inoltre, per essere in grado di stimare il primo termine in 1.81, sarebbe auspicabile una cross-validation. In [21] viene testato questo metodo per approssimare con un ensemble di reti neurali un'onda quadra in una variabile, mostrata in figura 1.10. Sono state utilizzate 5 reti con un hidden layer di 20 “neuroni”, addestrate indipendentemente le une dalle altre mediante back-propagation utilizzando 200 esempi casuali. L'ambiguitá é stata stimata da un insieme di 1000 input e i pesi associati agli output delle singole reti per la combinazione erano uniformi  $w_{\alpha} = 1/5$ . In figura 1.11 viene mostrato l'errore di generalizzazione in funzione della dimensione  $K$  dei set per la cross-validation. Innanzitutto notiamo come l'errore di generalizzazione sia lo stesso per un set di cross-validation di dimensione 40 o 0. Tuttavia bisogna considerare che sono stati scartati tutti i risultati di ensemble con due o piú reti non convergenti.

Dopo aver definito piú chiaramente il problema da affrontare, cerchiamone una soluzione. La maggior parte dei lavori é concentrata sulla combinazione degli output di reti piú performanti o ha indirizzato solo indirettamente il modo in cui generare un buon

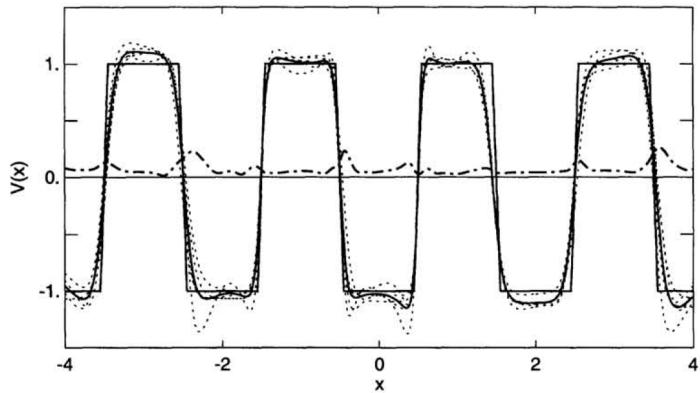


Figura 1.10: Tratto da [21]. La linea spessa e continua indica l'output dell'ensemble, mentre quelle tratteggiate gli output delle singole reti. Viene mostrata anche l'ambiguità (linea tratto-punto).

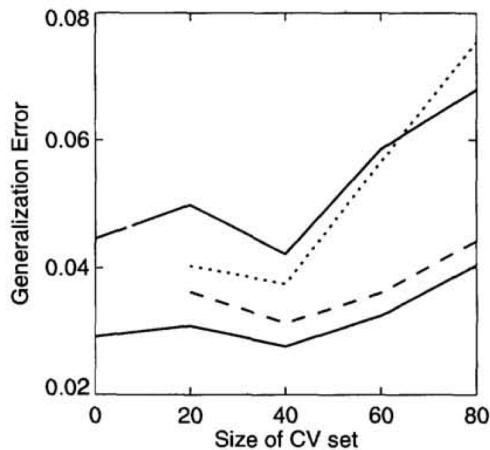


Figura 1.11: Tratto da [21]. La linea continua mostra l'errore di generalizzazione per pesi uniformi in funzione di  $K$ , dove  $K$  è la dimensione dei set di convalida. La linea tratteggiata è l'errore stimato dall'equazione 1.81. La linea tratteggiata è per i pesi ottimali stimati dall'uso degli errori di generalizzazione per le singole reti, stimate dai set di crossvalidation. La linea continua inferiore è l'errore generalizzato che si otterrebbe se i singoli errori di generalizzazione fossero conosciuti esattamente.

ensemble di reti, generando casualmente diverse topologie, impostazioni iniziali dei pesi, dei parametri o utilizzando solo una parte del set di training nella speranza di produrre reti che commettano errori per input distinti. Per trovare direttamente un ensemble ac-

curato e diversificato si possono sfruttare gli algoritmi genetici, creando una popolazione iniziale e utilizzando operatori genetici per creare continuamente nuove reti, mantenendo di volta in volta solo l'insieme di reti più performanti e al contempo più discordi tra loro.

Di seguito sintetizziamo l'algoritmo utilizzato. Viene creata una popolazione iniziale di reti neurali sottoposta a training. Di seguito si creano nuove reti partendo dalle precedenti utilizzando operazioni “genetiche” come mutazioni e crossover. La nuova popolazione viene nuovamente addestrata, ponendo particolare attenzione ad utilizzare gli esempi classificati erroneamente dalla popolazione precedente. Viene quindi associato un punteggio determinato dalla funzione di fitness:

$$Fitness_i = Accuracy_i + \lambda Diversity_i = (1 - E_i) + \lambda D_i \quad (1.83)$$

con la diversità definita come:

$$D_i = \sum [V_i(x) - \bar{V}(x)]^2. \quad (1.84)$$

Definiamo il termine di accuratezza del set di convalida della rete come  $A_i = 1 - E_i$  e utilizziamo l'equazione 1.84 per calcolare il termine di diversità  $D_i$ . Quindi normalizziamo separatamente ciascuno dei due termini. Non essendo sempre chiaro con quale valore si debba settare  $\lambda$ , solitamente ci si basa sulle seguenti regole. Il valore di  $\lambda$  non si modifica se l'errore dell'ensemble  $\hat{E}$  diminuisce mentre consideriamo nuove reti. Cambia invece se si verifica una delle seguenti: (1) se l'errore della popolazione  $\bar{E}$  non sta aumentando e la diversità  $D$  sta diminuendo, aumentiamo  $\lambda$ ; (2) se  $\bar{E}$  sta aumentando e  $\bar{D}$  non decresce, caliamo  $\lambda$ .

Il tutto si può sintetizzare come segue:

**GOAL:** creare geneticamente un ensemble di reti accurate e diversificate.

- Creare e addestrare la popolazione iniziale di reti.
- Finché non si raggiunge un limite di efficacia o di tempo:
  - Utilizzare operazioni genetiche per creare nuove reti.
  - Addestrare le nuove reti utilizzando l'equazione 1.85.
  - Misurare la diversità di ogni rete rispetto alla popolazione corrente (equazione 1.84).
  - Normalizzare i termini di accuratezza e diversità delle singole reti.
  - Calcolare la funzione di fitness in equazione 1.83.
  - Aggiornare la popolazione utilizzando le  $N$  reti con valore maggiore per la 1.83.
  - Correggere  $\lambda$  dell'equazione 1.83

- Utilizzare la popolazione attuale di reti come ensemble e combinare gli output delle reti seguendo l’equazione 1.69

Ribadisco che una rete utile all’ensemble é quella che classifica correttamente il maggior numero possibile di casi, mentre pecca principalmente laddove le altre reti classificano correttamente. Ci si preoccupa di questo aspetto durante la backpropagation moltiplicando la normale funzione di costo per un termine che misura l’errore combinato della popolazione su un determinato esempio:

$$Cost = \sum_{k \in T} \left| \frac{t(k) - \hat{o}(k)}{\hat{E}} \right|^{\frac{\lambda}{\lambda+1}} [t(k) - a(k)]^2 \quad (1.85)$$

dove  $t(k)$  é il target e  $a(k)$  é l’attivazione della rete per l’esempio  $k$  nel training set  $T$ . Si noti che, dato che la rete non é ancora un membro dell’ensemble,  $\hat{o}(k)$  e  $\hat{E}$  non dipendono dalla rete stessa; il nuovo termine é perciò una costante quando si calcolano le derivate nella back propagation. Si normalizza  $t(k) - \hat{o}(k)$  dividendo per l’errore dell’ensemble  $\hat{E}$  in modo che il valore medio del nuovo termine sia circa 1 indipendentemente dalla correttezza dell’insieme. Ció é particolarmente importante con popolazioni particolarmente accurate, poiché  $t(k) - \hat{o}(k)$  sarà vicino a 0 per la maggior parte degli esempi e la rete verrebbe addestrata solo su un numero esiguo di esempi. L’esponente  $\frac{\lambda}{\lambda+1}$  rappresenta il rapporto di importanza del termine di diversitá nella funzione di fitness. Ad esempio, se  $\lambda$  é vicino a 0, la diversitá non é considerata importante e la rete viene addestrata con la consueta funzione di costo; tuttavia se  $\lambda$  é grande, la diversitá é cosiderata importante e il nuovo termine nella funzione di costo assume maggiore importanza.

Si combinano le previsioni delle reti prendendo una somma ponderata dell’output di ciascuna rete, dove ogni peso é definito come in precedenza. Riporto di seguito i risultati ottenuti da David W. Opitz e Jude W. Shavlik [27].

L’algoritmo genetico che viene utilizzato per generare nuove topologie di rete é l’algoritmo REGENT [27]. REGENT utilizza algoritmi genetici per effettuare ricerche nello spazio delle topologie della rete neurale KN. I KN sono reti le cui topologie sono determinate a seguito della mappatura diretta di un insieme di regole di base che rappresentano ció che attualmente si conosce del problema da affrontare.

KBANN [28], per esempio, traduce un insieme di regole proposizionali in una rete neurale, quindi affina i pesi della rete risultante usando la back-propagation. L’uso di KNN consente di avere reti altamente corrette nell’ensemble; tuttavia, poiché ogni rete nell’ensemble é inizializzata con lo stesso insieme di regole specifiche del dominio, non ci si aspetta che vi sia un grande disaccordo tra le reti.

Un’alternativa da considerare é quella di generare casualmente la popolazione iniziale di topologie di rete, poiché a volte le regole specifiche del dominio non sono disponibili. É stato eseguito l’algoritmo genetico sul set di problemi MAX di NYNEX e su tre problemi del progetto genoma umano che aiutano a localizzare i geni nelle sequenze di

DNA. Ognuno di questi domini é accompagnato da una serie di regole approssimativamente corrette che descrivono ciò che attualmente é noto su tale attività [29] [27]. Gli esperimenti misurano l’errore del set di test dell’algoritmo genetico su queste attività. Ogni ensemble é composto da 20 reti e gli algoritmi REGENT e ADDEMUP (cosí é stato chiamato l’algoritmo genetico utilizzato in [30]) hanno considerato 250 reti durante la loro ricerca genetica. La tabella 1.14 presenta i risultati del caso in cui si crei casualmente la topologia delle reti.

La prima riga della tabella 1.14, la migliore rete, deriva da una rete neurale a singolo layer in cui, per ogni fold sono state addestrate 20 reti contenenti tra 0 e 100 hidden nodes (neuroni) e usato un set di validazione per scegliere la migliore. La riga successiva, il bagging, contiene i risultati dell’esecuzione dell’algoritmo di bagging di Breiman [31] su reti standard con un unico hidden layer, in cui il numero di hidden nodes viene impostato casualmente tra 0 e 100 per ogni rete. Il bagging é un “bootstrap”, cioè un metodo dell’ensemble che forma ogni rete con una diversa partizione dell’insieme di addestramento. Genera ogni partizione tracciando casualmente, con la sostituzione,  $N$  esempi dal set di addestramento, dove  $N$  é la dimensione del set di addestramento. La riga inferiore della tabella 1.14, ADDEMUP, contiene i risultati di una serie di ADDEMUP in cui la popolazione iniziale di 20 individui viene generata casualmente.

I risultati mostrano che su questi dominii la combinazione dell’output di piú reti addestrate di generalizza meglio rispetto al tentativo di scegliere la rete singola rete migliore. Mentre la tabella 1.14 mostra la potenza degli insiemi di reti neurali, la tabella 1.15 mostra la capacità di ADDEMUP di utilizzare le conoscenze a priori del problema.

La prima riga della tabella 1.15 contiene i risultati di generalizzazione dell’algoritmo KBANN, mentre la riga successiva, KBANN-bagging, contiene i risultati dell’ensemble in cui ogni singola rete é la rete KBANN addestrata su una diversa partizione dell’insieme di addestramento. Sebbene ciascuna di queste reti inizi con la stessa topologia e impostazione di peso iniziali “grande” (i pesi risultanti dalla conoscenza specifica del dominio), piccoli cambiamenti nel set di allenamento producono ancora cambiamenti significativi nelle previsioni. Si noti inoltre che su tutti i set di dati, il bagging KBANN é migliore dell’esecuzione del bagging su reti generate casualmente (bagging della tabella 1.14). La riga successiva, REGENT-combined, contiene i risultati della semplice combinazione, usando l’equazione 1.69, degli output delle reti nella popolazione finale di REGENT. ADDEMUP, l’ultima riga della tabella 1.15, differisce principalmente da REGENT-combined in due modi: (1) la sua funzione di fitness (equazione 1.83 ) tiene conto della diversità piuttosto che della sola precisione della rete e (2) allena le nuove reti enfatizzando gli esempi errati dell’ensemble attuale.

Pertanto, il confronto di ADDEMUP con REGENT-combined aiuta a testare direttamente la diversità di ADDEMUP, anche se i risultati aggiuntivi riportati in Opitz [29] mostrano che ADDEMUP ottiene gran parte del suo miglioramento dalla sua funzione di fitness.

Ci sono due ragioni principali per cui si ritiene che i risultati di ADDEMUP nel-

la tabella 1.15 siano particolarmente incoraggianti: (1) confrontando ADDEMUP con REGENT-combined, é stata esplicitamente testata la qualitá dell'euristica e ne si dimostra l'efficacia e (2) ADDEMUP é in grado di utilizzare efficacemente le conoscenze di base per ridurre l'errore delle singole reti nel suo insieme, pur essendo in grado di creare abbastanza diversitá tra di loro in modo da migliorare la qualitá complessiva dell'ensemble.

Tabella 1.14: Tratto da [27]. Standard neural networks (no domain-specific knowledge used)

	Promoters	Splice Junction	RBS	MAX
best-network	6.6%	7.8%	10.7%	37.0%
bagging	4.6%	4.5%	9.5%	35.7%
ADDEMUP	4.6%	4.9%	9.0%	34.9%

Tabella 1.15: Tratto da [27]. Knowledge-based neural networks (domain-specific knowledge used)

	Promoters	Splice Junction	RBS	MAX
KBANN	6.2%	5.3%	9.4%	35.8%
KBANN-bagging	4.2%	4.5%	8.5%	35.6%
REGENT-combined	3.9%	3.9%	8.2%	35.6%
ADDEMUP	2.9%	3.6%	7.5%	34.7%

## 1.10 Funzione d'attivazione sticastica

La funzione d'attivazione gioca un ruolo fondamentale nel training delle reti neurali in quanto introduce la non linearitá indispensabile per poter approssimare una qualsiasi funzione. Tra le diverse descritte in precedenza ci concentriamo sulla ReLU, in particolare la funzione d'attivazione che sarà descritta prende in considerazione il valore ottenuto con la precedente e ci applica un rumore stocastico. In questo modo si riesce a prevenire meglio l'overfitting senza dover intervenire su quanto gi appreso dalla rete durante il training.

Inizialmente si utilizzavano come funzione d'attivazione la sigmoide o la tangente iperbolica, in quanto continue, monotone e con dominio limitato. Il problema principale é dato dal fatto che le derivate tendono velocemente a valori molto piccoli, rallentando il processo di apprendimento. Per questo motivo la funzione ReLU é diventata particolarmente popolare e per comodit la riprendiamo di seguito:

$$\text{ReLU}(x) = \max(x). \quad (1.86)$$

La sua derivata é data da:

$$ReLU'(x) = \begin{cases} 1 & \text{se } x < 0 \\ 0 & \text{altrimenti} \end{cases} \quad (1.87)$$

La ReLU risolve in parte il problema delle derivate, tuttavia si annulla per tutti i valori negativi producendo la "morte" del neurone. Per risolvere il problema sono state introdotte delle varianti della ReLU, come la Leaky ReLU1.7.

L'idea di utilizzare una funzione di attivazione stocastica viene dal comportamento degli impulsi nervosi, i cui picchi sono disturbati da effetti bio-meccanici [32]. La perturbazione indotta risulta essere particolarmente efficace nel prevenire l'overfitting durante il training. In particolare si introduce una nuova funzione d'attivazione il cui output produce una perturbazione stocastica; si propone poi un metodo per governare la perturbazione stocastica addestrando i parametri della distribuzione di probabilitá attraverso la back-propagation e viene mostrato come questa funzione d'attivazione aumenti le performance della rete, in particolare nell'ambito di "Visual object classification".

In figura 1.12 visualizziamo l'effetti dovuto all'applicazione di questa nuova funzione d'attivazione.

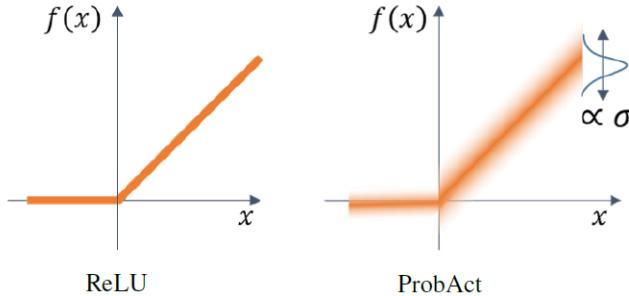


Figura 1.12: Confronto tra ReLU e funzione d'attivazione stocastica. L'effetto sfumato é stato introdotto solo per visualizzare come la funzione d'attivazione stocastica abbia come valore medio proprio  $ReLU(x)$ .

In generale ogni layer di una rete neurale produce un output dato da:

$$Y = f(w^T x), \quad (1.88)$$

Dove  $w^T$  indica il vettore dei pesi per quel layer,  $x$  il vettore dell'input e  $f$  la funzione d'attivazione, come puó essere la ReLU. Come mostrato in figura 1.12 introduciamo una variabile randomica nella ReLU per creare una funzione d' attivazione stocastica. In particolare la si puó definire come segue:

$$f(x) = \mu(x) + z \quad (1.89)$$

Dove  $\mu(x)$  é la ReLU e il tremine perturbativo  $z$  puó essere espresso come segue:

$$z = \sigma\epsilon. \quad (1.90)$$

La rete é ottimizzata usando l'apprendimento basato sul gradiente definito dal seguente teorema:

**Teorema 1.** *La propagazione del gradiente di una unitá stocastica  $h$  basata su una funzione deterministica  $g$  con input  $x$  (vettore contenente gli output dagli altri neuroni) e parametri interni  $\phi$  (pesi e bias); se il rumore  $z$  é possibile e se  $g(x, \phi, z)$  ha gradiente non nullo allora*

$$h = g(x, \phi, z)$$

La natura di  $z$  é gaussiana, in quanto  $z = \sigma\epsilon$  ed  $\epsilon$  é effettivamente una variabile gaussiana. Ció garantisce l'apprendimento attraverso il metodo basato sul gradiente.

Il parametro perturbato  $\sigma$  puó quindi essere fisso o addestrabile e definisce il range della perturbazione stocastica ed  $\epsilon$  é il valore ottenuto dalla distribuzione normale. Il vantaggio principale di fissare  $\sigma$  é quello di non aumentare il numero di parametri da dover addestrare nel training, però bisogna trovare empiricamente il valore piú corretto.

Si puó comunque scegliere di lasciare  $\sigma$  come parametro in piú da addestrare durante il training tramite back-propagation, assieme agli altri parametri. Il calcolo del gradiente per  $\sigma$  si ottiene attraverso la regola della catena. Data la funzione  $F$ , il gradiente di  $F$  rispetto a  $\sigma_{l,i}$  della  $i$ -esima unitá del  $l$ -esimo layer, é dato da:

$$\frac{\partial F}{\partial \sigma_{l,i}} = \frac{\partial F}{\partial y_{l,i}} \frac{\partial y_{l,i}}{\partial \sigma_{l,i}} \quad (1.91)$$

Con  $y_{l,i}$  l'output dell' $i$ -esima unitá del  $l$ -esimo layer. Il termine  $\frac{\partial F}{\partial y_{l,i}}$  é il gradiente propagato dal layer piú interno. Il gradiente dell' attivazione é dato da:

$$\frac{\partial y_{l,i}}{\partial \sigma_{l,i}} = \epsilon \quad (1.92)$$

Addestrare  $\sigma$  senza porre dei limiti puó provocare l'annullamento o la divergenza del parametro perturbativo con conseguente difficoltá nel training. Sfruttando le proprietá della funzione sigmoide possiamo definire  $\sigma = \alpha \text{sigmoid}(\beta k)$  per avere  $\sigma$  confinata tra 0 e  $\alpha$ . Con  $k$  il parametro da addestrare e  $\alpha$  e  $\beta$  parametri che posso essere tenuti fissi. Negli esperimenti descritti di seguito si avrá  $\alpha = 2$  e  $\beta = 5$  (valori scelti dopo un test apposito). Intuitivamente, la funzione d'attivazione stocastica aggiunge perturbazioni ad ogni passaggio tra un layer e quello successivo in modo indipendente, funzionando cosí anche come regolarizzatore della rete. Bisogna ricordare che il rumore viene poi propagato in tutti i layer successivi, quindi il rumore totale aggiunto ad un certo layer

dipende dal rumore applicato in precedenza, ma anche dai pesi. Ad esempio, anche nel caso piú semplice in cui la rete abbia due layer con un “neurone” per ogni layer, la distribuzione dell’output  $y_2$  del secondo layer differisce in base ai pesi del primo e secondo layer come segue:

$$y_2 = \mu [w_2\mu(w_1x) + w_2\sigma_1\epsilon] + \sigma_2\epsilon \quad (1.93)$$

$$= \begin{cases} \mu [w_2\mu(w_1x)] + w_2\sigma_1\epsilon + \sigma_2\epsilon & \text{se } w_2\mu(w_1x) + w_2\sigma_1\epsilon > 0; \\ \sigma_2\epsilon & \text{altrimenti} \end{cases} \quad (1.94)$$

$$\sim \begin{cases} N(\mu [w_2\mu(w_1x)], (w_2\sigma_1)^2 + \sigma_2^2) \\ N(0, \sigma_2^2). \end{cases} \quad (1.95)$$

Utilizzando l’equazione del teorema 1 , consideriamo  $g$  la funzione di introduzione del rumore nella rete, che dipende da  $z$  e da alcune trasformazioni differenziabili  $d$  sugli input  $x$  e sui parametri interni del modello. Allora possiamo scrivere l’output  $h$  come:

$$h = g(d, z) \quad (1.96)$$

Se utilizziamo l’equazione 1.96 per altri metodi di aggiunta di rumore come il dropout [33], si puó dedurre  $z$  come rumore moltiplicato subito dopo che la non linearitá é introdotta in un neurone. Nel caso descritto in questo documento, si campiona del rumore gaussiano e lo si aggiunge durante il calcolo di  $h$ . L’effetto della regolarizzazione é proporzionale alla varianza della distribuzione. Per evitare l’overfitting si puó impostare un valore di varianza elevato.

Ripropongo ora alcuni esperimenti con la funzione d’attivazione stocastica descritta in [34], applicata alla classificazione di immagini. Viene mostrato anche come tale funzione di attivazione lavori come regolarizzatore e come quindi pervenga l’overfitting. Descriviamo prima i dataset utilizzati e mostriamo di seguito i risultati:

- CIFAR-10 Dataset. 60000 immagini con 10 classi, 6000 immagini per classe e ogni immagine 32x32 pixel. 50000 immagini utilizzate per il training e 10000 per la convalida.
- CIFAR-10 Dataset 100 classi e 600 immagini per classe. 500 immagini utilizzate per il training e le rimanenti per la convalida. La risoluzione anche per queste é di 32x32 pixel.
- STL-10 Dataset. 500 immagini per ognuna delle 10 classi e 100 immagini tenute per il testing. Le immagini hanno una risoluzione di 96x96 pixel.

Per valutare le performance del metodo proposto si confronta la funzione d’attivazione *ProbAct* con: ReLU, LeakyReLU PReLU. Si utilizza una rete a 16 layer Visual Geometry Group (VGG). Non vengono utilizzati metodi di regolarizzazione ulteriori. Gli

Tabella 1.16: Tratto da [34]. Confronto dell'accuratezza ottenuta dalla rete utilizzando diverse funzioni di attivazione. Il valore ottenuto é dato dalla media su tre set di test. La funzione ProbAct é la funzione d'attivazione stocastica.

Activation function	CIFAR-10	CIFAR-100	STL-10	Train time	Test time
ReLU	86.67%	52.94%	60.80%	<b>1.00 X ReLU</b>	<b>1.00 X ReLU</b>
Leaky ReLU	86.49%	49.44%	59.16%	1.04 X ReLU	1.08 X ReLU
PReLU	86.35%	43.30%	60.01%	1.16 X ReLU	<b>1.00 X ReLU</b>
ProbAct					
Fixed ( $\sigma = 0.5$ )	88.50%	56.85%	62.30%	1.09 X ReLU	1.25 X ReLU
Fixed ( $\sigma = 1.0$ )	88.87%	<b>58.45%</b>	62.50%	1.10 X ReLU	1.27 X ReLU
Single Trainable $\sigma$	87.40%	52.87%	63.07%	1.23 X ReLU	1.30 X ReLU
Element-wise	86.40%	53.10%	61.70%	1.25 X ReLU	1.31 X ReLU
Trainable $\sigma$ (unbound)					
Element-wise	<b>88.92%</b>	55.83%	<b>64.17%</b>	1.26 X ReLU	1.33 X ReLU
Trainable $\sigma$ (bound)					

input vengono normalizzati. Le immagini tratte da STL-10 sono state ridotte di risoluzione per ottenere sempre immagini 32x32. Nel caso di  $\sigma$  wide-trainable e bounded sono stati ottenuti risultati superiori del 2.25% su CIFAR-10, del 2.89% su CIFAR-10 e del 3.37% su STL-10, comparati con la ReLU. In aggiunta il metodo proposto risulta il migliore tra quelli confrontati. In figura 1.13 é mostrato l' andamento del training di  $\sigma$ .

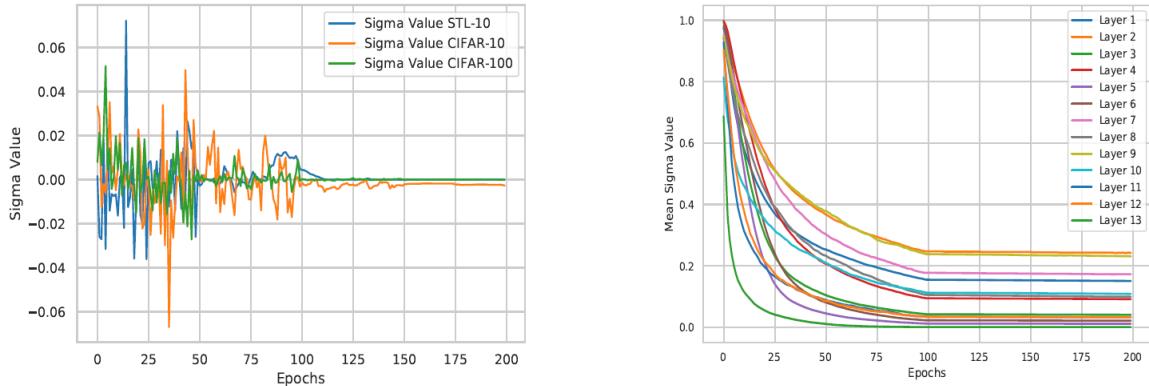


Figura 1.13: Tratto da [34]. A sinistra la transizione del singolo sigma addestrabile nell'architettura VGG-16 sui tre set di dati. A destra il valore medio dei  $\sigma$  addestrati sul set di dati CIFAR10, evidenziando i diversi valori per ogni layer.

Il training é costituito da 400 epochs, ma non vengono mostrati i dati relativi alle epochs superiori a 200 in quanto non hanno contribuito significativamente al training. Infatti giá dopo solo 100 epochs la *single trainable  $\sigma$*  tende a 0.

Figura 1.14 mostra la distribuzione in frequenza dell' elemento  $k$  dopo 400 epochs. Si osserviamo due picchi per ogni distribuzione attraverso i 3 dataset.

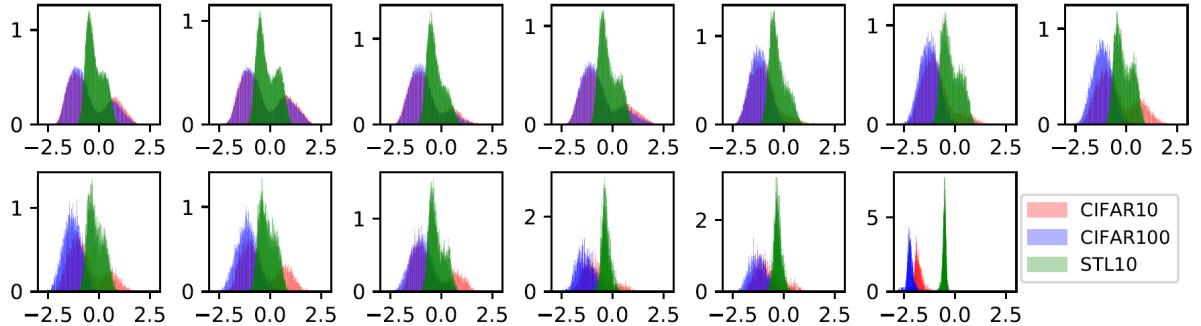


Figura 1.14: Tratto da [34]. Distribuzione in frequenza dell' elemento  $k$  dopo l'allenamento dell'architettura VGG-16 per i set di dati CIFAR-10, CIFAR-100 e STL-10 per ogni layer. L'asse X indica il valore  $k$  dopo l'allenamento e l'asse Y indica la sua frequenza. Ogni sottodisegno rappresenta un layer in cui viene applicata la funzione d'attivazione stocastica, a partire dal primo layer in alto a sinistra fino all'ultimo layer in basso a destra.

Un elevato valore di  $\sigma$  agisce come un regolarizzatore interno, migliorando la generalizzazione e prevenendo l'overfitting.

Per definire il livello di overfitting di una rete utilizziamo un termine  $\gamma$  che mostra la differenza di accuratezza tra training e test:

$$\gamma = TrainAccuracy(\%) - TestAccuracy(\%). \quad (1.97)$$

Se  $\gamma$  è piccolo, la rete in seguito al training si è generalizzata bene anche al set di test. Se invece è grande, la rete ha imparato molto bene a classificare gli esempi del training, ma non riesce a classificare casi diversi da quelli già visti.

In figura 1.15 si confrontano le abilità di generalizzazione con e senza l'utilizzo di dropout per il dataset CIFAR-100.

Il dropout viene applicato con una probabilità del 0.5%. La rete senza dropout e con funzione d'attivazione ReLU ha circa  $\gamma = 48\%$ , valore che non viene modificato di molto se introdotto il dropout. D'altra parte, fissato  $\sigma = 1$  si raggiunge  $\gamma = 43\%$  e con  $\sigma = 2$  si abbassa ulteriormente a  $\gamma = 39\%$ , sempre senza utilizzare il dropout. Introducendo anche il dropout si ottiene  $\gamma = 30\%$  con  $\sigma = 1$ .

Una funzione d'attivazione stocastica diventa particolarmente efficace nel caso in cui si abbia un dataset ridotto. Per confermare ciò sono stati confrontati i risultati di reti identiche, una con funzione d'attivazione ReLU e l'altra con funzione d'attivazione stocastica (ProbAct [34]), utilizzando solo una frazione dei dataset CIFAR-10 e CIFAR-100. I risultati ottenuti sono mostrati in tabella 1.17.

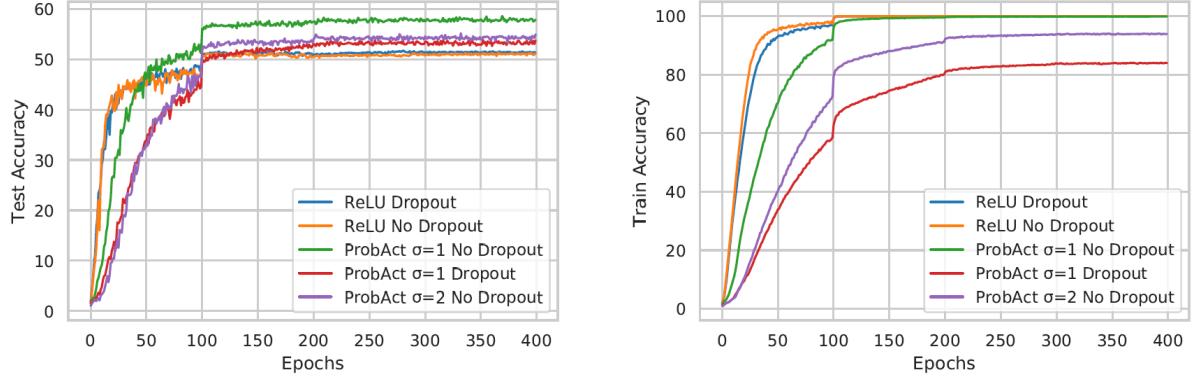


Figura 1.15: Tratto da [34]. Confronto tra ReLU e funzione d’attivazione stocastica, sia con che senza dropout, utilizzando il dataset CIFAR-100. In particolare a sinistra il confronto é effettuato nel set di test e a destra nel set di training.

Tabella 1.17: Tratto da [34]. Confronto dell’accuratezza del test tra ReLU e ProbAct su sottoinsiemi ridotti di CIFAR-10 e CIFAR-100 (50% e 25% del set di dati originale). L’accuratezza del test indicata é data dalla media su tre serie di test. Sono stati evidenziati i risultati migliori (tutti appartenenti a ProbAct).

Activation function	CIFAR-10 (50%)	CIFAR-100 (50%)	CIFAR-10 (25%)	CIFAR-100 (25%)
ReLU	82.74%	42.36%	75.62%	30.42%
ProbAct	<b>84.73%</b>	<b>46.11%</b>	<b>79.02%</b>	<b>31.67%</b>

# Applicazione delle reti neurali al problema del MRF

Le reti neurali a valori complessi sembrano essere piú performanti rispetto a quelle reali nel caso l'input da analizzare sia naturalmente esprimibile nel campo complesso 1.8. Per questo motivo si é scelto di studiare in parallelo ai possibili vantaggi e svantaggi della rete neurale a valori complessi, una possibile applicazione nel campo della risonanza magnetica, in particolare nel MRI fingerprinting.

## 1.11 MRF con reti neurali a valori reali

Recentemente il MRF é stato proposto come tecnica di imaging quantitativa per l'acquisizione simultanea di parametri tissutali come i tempi di rilassamento  $T_1$  e  $T_2$ . Sebbene l'acquisizione sia altamente accelerata, la ricostruzione soffre di lunghi tempi di calcolo: i metodi di matching usati per trovare il segnale piú simile a quello misurato devono confrontare quest'ultimo con un numero elevato di segnali simulati. Gli approcci di deep-learning possono superare questa limitazione, fornendo la mappatura diretta del segnale misurato attraverso una rete neurale.

Per evitare errori o imprecisioni, il dizionario viene in genere calcolato con una granularitá fine su una gamma molto vasta di possibili valori dei tessuti. Le dimensioni del dizionario tuttavia, aumentano in modo esponenziale all'aumentare del numero di parametri tissutali, il che puó rapidamente portare a dizionari proibitivi di grandi dimensioni, che richiedono ingenti risorse di elaborazione. Questo aumento della memoria, dell'archiviazione e dell'onere computazionale é un fattore limitante per l'adozione clinica dei metodi MRF.

Ridurre la densitá del dizionario come proposto in precedenza, non é però una soluzione convincente a questo problema perché limita a priori l'accuratezza della ricostruzione, senza considerare ancora imprecisioni dovute a fattori sperimentali.

Il lavoro matematico nella teoria delle reti neurali e algoritmi di deep-learning ha dimostrato che qualsiasi funzione di Borel puó essere rappresentata da una rete neurale con un numero finito di neuroni, che possono quindi offrire una rappresentazione compatta di funzioni complicate [35]. Sfruttando questa proprietá si puó cercare una rete

neurale che sfrutti il dizionario come set di training e/o set di test e che vada a sostituire il lento processo di matching. Infatti in termini di spazio di archiviazione sarebbe più compatta di un dizionario ed il processo di matching, una volta compiuto il training verrebbe drasticamente velocizzato.

Per verificare l'efficienza di questo modello, in [36] viene definita una rete neurale con 4 layer (uno di input, 2 hidden layer e un layer di output) utilizzando il TensorFlow [37]. L'input consiste di 25 o 50 nodi e l'output da 2 nodi in quanto sono studiati solo il  $T_1$  e il  $T_2$ . Ogni hidden layer è composto da 300 nodi. Le dimensioni della rete sono state selezionate empiricamente, cercando un buon compromesso tra il tempo di addestramento della rete, lo spazio di archiviazione e l'accuratezza delle ricostruzioni. Il tasso di apprendimento della rete è stato impostato come 0.001 e la funzione di costo è data dall'errore quadratico medio. Il training viene effettuato sfruttando un dizionario con circa 69000 esempi, mentre per la convalida è stato sfruttato il BrainWeb digital brain phantom insieme a simulazioni montecarlo, introducendo rumore gaussiano.

L'accuratezza della rete è comparabile a quella dei metodi tradizionali di MRF e in particolare è stato riscontrato una migliore efficienza in caso di rumore o undersampling.

## 1.12 MRF con reti neurali a valori complessi

Dati i risultati ottenuti con le reti neurali a valori reali (già promettenti) e i miglioramenti ottenuti dalle reti neurali a valori complessi nell'analisi di segnali ondulatori nei confronti delle reti a valori reali, propongo in questo documento l'applicazione di queste ultime al problema MRF.

L'argomento è stato trattato anche in [38], dove si comparano diversi metodi applicabili al fingerprinting. Riportiamo di seguito i metodi comparati ed i risultati ottenuti:

- prodotto scalare del segnale misurato con quello simulato del dizionario;
- rete neurale a 2 canali, che lavorano separatamente su parte reale e parte immaginaria del segnale;
- rete neurale a valore reale analoga alla precedente, ma di dimensione doppia, con 1024 e 512 “feature channels” nei 2 hidden layer;
- rete neurale a valori complessi, con un solo canale che utilizza la “cardioid activation function” definita nel documento citato;
- rete neurale a valori complessi che utilizza come funzione di attivazione una sigmoide applicata separatamente a parte reale e parte immaginaria;
- rete neurale a valori complessi che utilizza la funzione d'attivazione siglog [39];

Come risultato in [38] si ottiene che le reti neurali in generale sono piú efficaci nella predizione dei valori  $T_1$  e  $T_2$ , mentre il prodotto scalare rimane il piú accurato per quanto riguarda  $\Delta B_0$ . Nel confronto tra reti neurali a valori reali e reti a valori complessi, queste ultime sembrano performare meglio. In particolare nel caso di segnale pulito, la rete a valori complessi prevale in accuratezza sia su  $T_1$  che su  $T_2$ , mentre prevale solo nell'accuratezza su  $T_1$  in caso di segnale con rumore.

# Conclusioni

La risonanza magnetica si é dimostrata estremamente utile per la diagnosi, la prognosi e la valutazione terapeutica, ma ha lo svantaggio di essere lenta rispetto ad altri strumenti diagnostici. Inoltre é generalmente qualitativa: il principale mezzo di informazione utilizzato per caratterizzare una patologia é il contrasto tra i tessuti.

Si cerca di soppiare alla mancanza di misure oggettive attraverso il MRF, tecnica di risonanza magnetica quantitativa che presenta una notevole velocizzazione nell'acquisizione del segnale, con il vantaggio di ottenere da questo i valori di piú parametri per volta, ma con lo svantaggio principale di spendere molto tempo nel matching tra segnale e dizionario.

Le reti neurali sembrano essere una soluzione efficace: velocizzano i tempi di calcolo dei parametri da studiare e non necessitano della creazione del dizionario per ogni acquisizione perché capaci di generalizzazione. In particolare quelle a valori complessi sembrano essere maggiormente efficienti se l'input é di natura ondulatoria, proprio come nel caso della risonanza magnetica.

Il loro utilizzo però non é ancora popolare e gli accorgimenti necessari per il loro funzionamento non sono banali, a partire dalla derivazione della funzione di costo. Per questo motivo é difficile garantire il successo di tali modelli che, per l'appunto, non sono ancora utilizzati in ambito clinico. A questo proposito sono state discusse anche tecniche di ottimizzazione e generalizzazione delle reti neurali, da applicare assieme alle reti neurali a valori complessi, come gli ensemble di reti e la funzione d'attivazione stocastica. Tutto ciò nella speranza che questa possa effettivamente essere una risposta concreta al problema del MRF affrontato.

# Bibliografia

- [1] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.
- [2] Donald L Birx and Stephen J Pipenberg. A complex mapping network for phase sensitive classification. *IEEE transactions on Neural Networks*, 4(1):127–135, 1993.
- [3] Hiroshi Sawada, Ryo Mukai, Shoko Araki, and Shoji Makino. Polar coordinate based nonlinear function for frequency-domain blind source separation. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 86(3):590–596, 2003.
- [4] Ryo Yamaki and Akira Hirose. Singular unit restoration in interferograms based on complex-valued markov random field model for phase unwrapping. *IEEE Geoscience and Remote Sensing Letters*, 6(1):18–22, 2008.
- [5] Bernard Widrow. Generalization and information storage in networks of adaline neurons. self organizing systems. *Yovitz, MC, Jacobi, GT, and Goldstein, GD editors*, pages 435–461, 1962.
- [6] Bernard Widrow, John McCool, and Michael Ball. The complex lms algorithm. *Proceedings of the IEEE*, 63(4):719–720, 1975.
- [7] DH Brandwood. A complex gradient operator and its application in adaptive array theory. In *IEE Proceedings H-Microwaves, Optics and Antennas*, volume 130, pages 11–16. IET, 1983.
- [8] Wilhelm Wirtinger. Zur formalen theorie der funktionen von mehr komplexen veränderlichen. *Mathematische Annalen*, 97(1):357–375, 1927.
- [9] Ramaswamy Savitha, Sundaram Suresh, N Sundararajan, and P Saratchandran. A new learning algorithm with logarithmic performance index for complex-valued neural networks. *Neurocomputing*, 72(16-18):3771–3781, 2009.

- [10] Akira Hirose. Complex-valued neural networks: The merits and their origins. In *2009 International joint conference on neural networks*, pages 1237–1244. IEEE, 2009.
- [11] Nitzan Guberman. On complex valued convolutional neural networks. *arXiv preprint arXiv:1602.09046*, 2016.
- [12] Chiheb Trabelsi, Olexa Bilaniuk, Ying Zhang, Dmitriy Serdyuk, Sandeep Subramanian, João Felipe Santos, Soroush Mehri, Negar Rostamzadeh, Yoshua Bengio, and Christopher J Pal. Deep complex networks. *arXiv preprint arXiv:1705.09792*, 2017.
- [13] Nils Mönning and Suresh Manandhar. Evaluation of complex-valued neural networks on real-valued classification tasks. *arXiv preprint arXiv:1811.12351*, 2018.
- [14] Jingkun Gao, Bin Deng, Yuliang Qin, Hongqiang Wang, and Xiang Li. Enhanced radar imaging using a complex-valued convolutional neural network. *IEEE Geoscience and Remote Sensing Letters*, 16(1):35–39, 2018.
- [15] Simone Scardapane, Steven Van Vaerenbergh, Amir Hussain, and Aurelio Uncini. Complex-valued neural networks with nonparametric activation functions. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2018.
- [16] F Fogelman Soulie, P Gallinari, Yann Le Cun, and S Thiria. Evaluation of network architectures on test learning tasks. In *Unknown Host Publication Title*. SOS Printing, 1987.
- [17] Robert T Clemen. Combining forecasts: A review and annotated bibliography. *International journal of forecasting*, 5(4):559–583, 1989.
- [18] David H Wolpert. Stacked generalization. *Neural networks*, 5(2):241–259, 1992.
- [19] Ethem Alpaydin. Multiple networks for function learning. In *IEEE International Conference on Neural Networks*, pages 9–14. IEEE, 1993.
- [20] Harris Drucker, Corinna Cortes, Lawrence D Jackel, Yann LeCun, and Vladimir Vapnik. Boosting and other machine learning algorithms. In *Machine Learning Proceedings 1994*, pages 53–61. Elsevier, 1994.
- [21] Anders Krogh and Jesper Vedelsby. Neural network ensembles, cross validation, and active learning. In *Advances in neural information processing systems*, pages 231–238, 1995.
- [22] Richard Maclin, Jude W Shavlik, et al. Combining the predictions of multiple classifiers: Using competitive learning to initialize neural networks. In *IJCAI*, pages 524–531. Citeseer, 1995.

- [23] Michael P Perrone. A soft-competitive splitting rule for adaptive tree-structured neural networks. In *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, volume 3, pages 689–693. IEEE, 1992.
- [24] William P Lincoln and Josef Skrzypek. Synergy of clustering multiple back propagation networks. In *Advances in neural information processing systems*, pages 650–657, 1990.
- [25] Lars Kai Hansen and Peter Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (10):993–1001, 1990.
- [26] Sherif Hashem, Bruce Schmeiser, and Yuehwern Yih. Optimal linear combinations of neural networks: An overview. In *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, volume 3, pages 1507–1512. IEEE, 1994.
- [27] David W Opitz and Jude W Shavlik. Using genetic search to refine knowledge-based neural networks. In *Machine Learning Proceedings 1994*, pages 208–216. Elsevier, 1994.
- [28] Geoffrey G Towell and Jude W Shavlik. Knowledge-based artificial neural networks. *Artificial intelligence*, 70(1-2):119–165, 1994.
- [29] David William Opitz. *An anytime approach to connectionist theory refinement: refining the topologies of knowledge-based neural networks*. PhD thesis, Citeseer, 1995.
- [30] David W Opitz and Jude W Shavlik. Generating accurate and diverse members of a neural-network ensemble. In *Advances in neural information processing systems*, pages 535–541, 1996.
- [31] Leo Breiman. Bagging predictors (technical report 421). *University of California, Berkeley*, 1994.
- [32] Michael S Lewicki. A review of methods for spike sorting: the detection and classification of neural action potentials. *Network: Computation in Neural Systems*, 9(4):R53–R78, 1998.
- [33] Shaofeng Cai, Jinyang Gao, Meihui Zhang, Wei Wang, Gang Chen, and Beng Chin Ooi. Effective and efficient dropout for deep convolutional neural networks. *arXiv preprint arXiv:1904.03392*, 2019.
- [34] Joonho Lee, Kumar Shridhar, Hideaki Hayashi, Brian Kenji Iwana, Seokjun Kang, and Seiichi Uchida. Probact: A probabilistic activation function for deep neural networks. *arXiv preprint arXiv:1905.10761*, 2019.

- [35] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.
- [36] Ouri Cohen, Bo Zhu, and Matthew S Rosen. Mr fingerprinting deep reconstruction network (drone). *Magnetic resonance in medicine*, 80(3):885–894, 2018.
- [37] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.
- [38] Patrick Virtue, X Yu Stella, and Michael Lustig. Better than real: Complex-valued neural nets for mri fingerprinting. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3953–3957. IEEE, 2017.
- [39] George M Georgiou and Cris Koutsougeras. Complex domain backpropagation. *IEEE transactions on Circuits and systems II: analog and digital signal processing*, 39(5):330–334, 1992.