# 1 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. Introduco prima brevemente il concetto di rete neurale, l'architettura, la dinamica e l'apprendimento relativi ad esse e di seguito i problemi relativi all'introdurre i numeri complessi nel modello di rete neurale. In particolare tratto le diversitá introdotte dalla differente algebra del prodotto tra numeri complessi e la complicazione dovuta alla derivazione di funzioni complesse. 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 oltre alla non linearitá giá introdotta in generale con le funzioni d'attivazione.

# 2 Reti neurali

Il caso piú semplice di rete neurale é il percettrone mostrato in figura 1, introdotto da Rosenblatt nel 1958 [**?**]. Con esso, senza ancora introdurre alcuna funzione di attivazione (di cui spiego il ruolo nel capitolo seguente), si possono giá rappresentare tutte le funzioni che sono linearmente separabili come l'AND, l'OR, il NAND o lo XOR. In figura 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'attivitá 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.

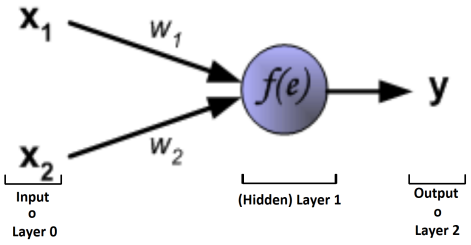


Figure 1: Percettrone con input composto da due neuroni $x_1$ e $x_2$, un unico neurone nel layer centrale e un output $y$. Il valore del neurone del layer 1 é dato da $f(e)$ con $e$ dato dalla somma pesata tra $x_1$ e $x_2$ e i pesi che sono rispettivamente $w_1$ e $w_2$.

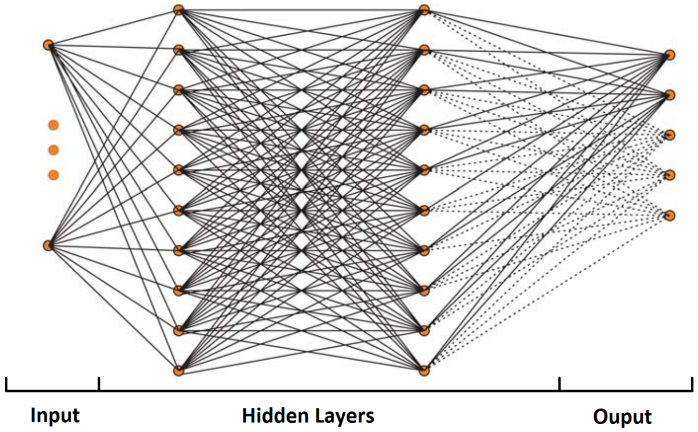


Figure 2: Struttura di una rete neurale. Ogni neurone definito dal cerchio arancione si comporta in modo analogo al percettrone visto in precedenza, con la differenza che l' "output" viene nuovamente combinato, come se fosse uno dei neuroni di input del layer successivo. L'operazione si ripete per il numero di layer, fino a giungere all'output della rete. Continuando l'analogia con la figura 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 cosí 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 é preferito applicare una funzione d'attivazione diversa, per esempio, ai neuroni che costituiscono l'output rispetto a tutti gli altri.

## 2.1 Struttura

Partendo dal caso piú semplice mostrato in figura 1 si visualizzano gli elementi fondamentali che costituiscono una rete neurale. L'input $x$ é composto da un determinato numero di "neuroni" ed é da essi che il segnale comincia a propagarsi. Ogni unitá che costituisce l'input é 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) che determina l'influenza del "neurone" precedente nella propagazione del segnale e una eventuale funzione di attivazione ($f(e)$ in figura 1) viene applicata al segnale finale per introdurre la non linearitá nel sistema, definendo quindi il valore finale del percettrone del layer centrale. Nel caso del percettrone tale valore puó essere direttamente l'output a meno di eventuali fattori moltiplicativi, altrimenti, nelle reti come in figura 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 6 pare essere una delle piú utilizzate. Mostro comunque altre tra le funzioni di attivazione piú incontrate:

- Threshold Logic Units, cioé funzioni a gradino:

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

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

- Funzioni lineari:
$$f(x) = \alpha x + \beta; \tag{2}$$
con $\alpha$ e $\beta$ che si aggiungono ai parametri della rete;

- Sigmoidi:
$$f(x) = \frac{1}{1 + e^{-x}}; \tag{3}$$

- tangenti iperboliche
$$f(x) = tanh(x) \tag{4}$$

- gaussiane
$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2} \tag{5}$$
con $\sigma$ e $\mu$ che si aggiungono ai parametri della rete;

- Rectifier Linear Units (ReLU)
$$f(x) = ReLU(x) = \max(0; x); \tag{6}$$

- Leaky ReLU
$$f(x) = LeakyReLU(x) = \begin{cases} \lambda x & \text{se} \quad x < 0 \\ x & \text{se} \quad x \geq 0 \end{cases}; \tag{7}$$
con $\lambda$ solitamente ¡¡1 che si aggiunge ai parametri della rete;

Passando all'approssimare funzioni piú complesse si introduce la rete neurale in figura 2, in cui ogni elemento di ogni layer si comporta in modo analogo al percettrone.

## 2.2 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 dovrá apprendere).

$$a_j = \sum_{i=1}^{n} w_i o_i \tag{8}$$

con $j$ che indica che stiamo considerando il $j$-esimo neurone del layer successivo e $o_i$ (che in questo caso é 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 é 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.

## 2.3 Apprendimento

La definizione di una rete neurale si completa introducendo quello che é l'aspetto principale di questo algoritmo: l'apprendimento. La rete neurale apprende dagli esempi contenuti nel set di training $\tau = \{z_1, \cdots, z_n\}$. Il training set differisce a seconda che l'apprendimento sia supervisionato: $\tau = \{(x_1, \widehat{y}_1), \cdots, (x_n, \widehat{y}_n)\}$ o non supervisionato: $\tau = \{x_1, \cdots, x_n\}$.

L'apprendimento é un processo di modifiche progressive effettuate ai parametri della rete (principalmente i pesi, ma, come visto pocanzi, 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:

$$C(\tau, w) = C(W) = \frac{1}{2}\sum_{i=1}^{n}(\widehat{y}_i - y_i)^2. \tag{9}$$

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

$$w' = w + \Delta w \tag{10}$$

$$\Delta w = -\eta\frac{\partial C}{\partial w} \tag{11}$$

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 $L_1, \cdots, L_{l-1}$ 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:

$$\frac{\partial C}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}}\left\{\frac{1}{2}\sum_{i=1}^{m}(\widehat{y}_i - y_i)^2\right\} \tag{12}$$

$$= \frac{\partial}{\partial o_j}\left\{\frac{1}{2}\sum_{i=1}^{m}(\widehat{y}_i - y_i)^2\right\}\frac{\partial o_j}{\partial w_{jk}} \tag{13}$$

$$= \left\{\frac{1}{2}\sum_{i=1}^{m}\frac{\partial}{\partial o_j}(\widehat{y}_i - y_i)^2\right\}\frac{\partial o_j}{\partial w_{jk}} \tag{14}$$

$$= \left\{-\sum_{i=1}^{m}(\widehat{y}_i - y_i)\frac{\partial y_i}{\partial o_j}\right\}\frac{\partial o_j}{\partial w_{jk}} \tag{15}$$

$$\frac{\partial y_i}{\partial o_j} = \frac{\partial f_i(a_i)}{\partial a_i}\frac{\partial a_i}{\partial o_j} \tag{16}$$

$$= \dot{f}_i(a_i)\frac{\partial}{\partial o_j}\sum_j w_{ij}o_{ij} \tag{17}$$

$$= \dot{f}_i(a_i)w_{ij} \tag{18}$$

$$\Rightarrow \frac{\partial C}{\partial w_{jk}} = \left\{ -\sum_{i=1}^{m} (\widehat{y}_i - y_i) \dot{f}_i(a_i) w_{ij} \right\} \frac{\partial o_j}{\partial w_{jk}} \tag{19}$$

$$= -\left( \sum_{i=1}^{m} w_{ij} \delta_i \right) \frac{\partial f_j(a_j)}{\partial a_j} \frac{\partial a_j}{\partial w_{jk}} \tag{20}$$

$$= -\left( \sum_{i=1}^{m} w_{ij} \delta_i \right) \dot{f}_j(a_j) o_k \tag{21}$$

$$\tag{22}$$

$$\Rightarrow \Delta w_{jk} = -\eta \frac{\partial C}{\partial w_{jk}} = \eta \left( \sum_{i=1}^{m} w_{ij} \delta_i \right) \dot{f}_j(a_j) o_k \tag{23}$$

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

Definendo:

$$\delta_j = \begin{cases} (\widehat{y}_j - y_j)\dot{f}_j(a_j) & \text{se } \; j \in L_l \\ \left( \sum_{i \in L_{k+1}} w_i j \delta_i \right) \dot{f}_j(a_j) & \text{se } \; j \in L_k \;\; \text{dove } \; k = l-1, \cdots, 0 \end{cases} \tag{24}$$

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

$$\Delta w_{jk} = \eta \delta_j o_k \tag{25}$$

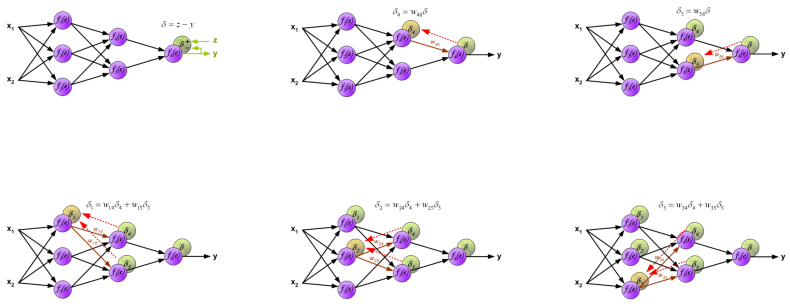

Figure 3: Utilizzando la nomenclatura introdotta in questo capitolo si visualizzano i valori di $\delta_j$ dovuti alla back propagation, rispettando l'effettivo ordine in cui vengono calcolati.

Volendo il piú delle volte definire la percentuale di successo della rete e avendo solitamente un set finito $\{(x_1, \widehat{y}_1), \cdots, (x_N, \widehat{y}_N)\}$, si sceglie il set di training con $n < N$ per poter utilizzare gli esempi rimanenti come set di convalida $\zeta = \{(x_{n+1}, \widehat{y}_{n+1}), \cdots, (x_N, \widehat{y}_N)\}$. Applicando la rete su questi nuovi input si riesce quindi a stimare l'efficacia della rete neurale.

# 3 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 [**?**], nella "voice processing" [**?**] e nell'analisi dei segnali sonar per la stima di informazioni su un determinato paesaggio [**?**].

## 3.1 Storia

Uno dei primi esempi di rete neurale fu il percettrone di Rosenbatt (1958) [**?**]. Di seguito fu creata ADELINE (Widrow & Hoff, 1960) [**?**], 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:

$$y = x \cdot w. \tag{26}$$

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

$$e = \widehat{y} - y \tag{27}$$
$$L = e^2 \tag{28}$$
$$w \leftarrow arg\,min(L). \tag{29}$$

L'algoritmo $LMS$ ottimizza $L$ usando il gradiente discendente:

$$\nabla L_w := -2ex \tag{30}$$
$$w \leftarrow w + \eta ex \tag{31}$$

dove $\eta$ é il tasso di apprendimento.

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

Aggiorniamo quindi il problema nel dominio complesso:

$$L = e\overline{e} \tag{32}$$
$$\nabla L_w := -2e\overline{x} \tag{33}$$
$$w \leftarrow w + \eta e\overline{x}. \tag{34}$$

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.

## 3.2 Struttura

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

$$h = f\left(W^{(h)}x + b^{(h)}\right) \tag{35}$$

$$y = W^{(o)}x + b^{(o)} \tag{36}$$

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

$$\begin{array}{ccc} W^{(h)} \in R^{M\times M} & or & W^{(o)} \in C^{M\times M} \\ b^{(h)} \in R^{M} & or & b^{(o)} \in C^{M} \\ W^{(o)} \in R^{N\times M} & or & W^{(o)} \in C^{N\times M} \\ b^{(o)} \in R^{N} & or & b^{(o)} \in C^{N} \end{array} \tag{37}$$

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

## 3.3 Funzione d'attivazione

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

**Identitá** Permette una modellizzazione lineare

$$f^{(-)}(x) := x \tag{38}$$

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

$$f^{(\sigma)} := tanh(x) \tag{39}$$

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

$$f^{(ri)}(x) := tanh(Re\ x) + i\ tanh(Im\ x) \tag{40}$$

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

$$f^{(ap)}(x) := tanh(|x|)\, e^{i\, arg\, x} \tag{41}$$

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

$$ModReLU(z) = ReLU(|z| + b)\, e^{i\theta_z} = \begin{cases} (|z| + b)\, \frac{z}{|z|} & \text{se } |z| + b \geq 0 \\ 0 & \text{altrimenti} \end{cases} \tag{42}$$

dove $\theta_z$ é 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 é inattivo. Tale funzione non soddisfa le equazioni di Cauchy-Riemann e quindi non é olomorfa.

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

$$CReLU\,(z) = ReLU\,(Re\,(z)) + i\,ReLU\,(Im\,(z))\,. \tag{43}$$

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 $\theta_z \in \left(0, \frac{\pi}{2}\right)$ oppure $\theta_z \in \left(\pi, \frac{3\pi}{2}\right)$.

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

$$zReLU\,(z) = \begin{cases} z & \text{se}\theta_z \in \left[0, \frac{\pi}{2}\right] \\ 0 & \text{altrimenti} \end{cases} \tag{44}$$

## 3.4 Back propagation

### 3.4.1 Loss function

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

$$e := \widehat{y} - y \tag{45}$$

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

$$L(e) = \sum_{i=0}^{N-1} |e_i|^2 \tag{46}$$

$$= \sum_{i=0}^{N-1} e_i \overline{e_i}. \tag{47}$$

La 46 é una funzione a valori reali scalari non negativi, che tende a zero insieme al modulo dell'errore. Savitha, Suresh e Sundararajan [**?**] proposero di sostituire l'errore 45 con:

$$e := log(\widehat{y}) - log(y). \tag{48}$$

La *loss function* diventerebbe quindi:

$$L(e) = (log\,|\widehat{y_i}| - log\,|y_i|)^2 + (arg(\widehat{y_i}) - arg(y_i))^2 \tag{49}$$

L'equazione 49 ha la proprietá di rappresentare esplicitamente l'ampiezza e la fase. Le equazioni 45 e 48 possono essere *error function* appropriate per reti neurali complesse.

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

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

Definiamo:

$$f(z) := f(z, \overline{z}) \tag{50}$$
$$= g(x, y) \tag{51}$$
$$= u(x, y) + iv(x, y) \tag{52}$$

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

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

$$\left.\frac{\partial f}{\partial z}\right|_{\overline{z}\ costante} \tag{53}$$
$$\left.\frac{\partial f}{\partial \overline{z}}\right|_{z\ costante} \tag{54}$$

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

$$\frac{\partial f}{\partial z} = \frac{1}{2}\left(\frac{\partial f}{\partial x} - i\frac{\partial f}{\partial y}\right) \tag{55}$$
$$\frac{\partial f}{\partial \overline{z}} = \frac{1}{2}\left(\frac{\partial f}{\partial x} + i\frac{\partial f}{\partial y}\right). \tag{56}$$

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

$$\frac{\partial u}{\partial x} = \frac{\partial v}{\partial y} \tag{57}$$
$$\frac{\partial v}{\partial x} = -\frac{\partial u}{\partial y}. \tag{58}$$

Se applichiamo le condizioni di Cauchy-Riemann alla derivata di $f$ in $\overline{z}$ 56 notiamo che effettivamente essa si annulla. Le funzioni olomorfe quindi non dipendono esplicitamente da $\overline{z}$. Brandwood dimostró che l'annullarsi di 55 o 56 per una generica $f : C \to R$ é condizione sufficiente e necessaria affinché $f$ abbia un punto stazionario. Per estensione se $f : C^N \to R$ é una funzione a valori reali di un certo vettore $z = [z_0,\ z_1,\ \cdots,\ z_{N-1}]^T \in C^N$ e definiamo il cogradiente e il gradiente coniugato come:

$$\frac{\partial}{\partial z} := \left[\frac{\partial}{\partial z_0}, \frac{\partial}{\partial z_1}, \cdots, \frac{\partial}{\partial z_{N-1}}\right] \tag{59}$$
$$\frac{\partial}{\partial \overline{z}} := \left[\frac{\partial}{\partial \overline{z_0}}, \frac{\partial}{\partial \overline{z_1}}, \cdots, \frac{\partial}{\partial \overline{z_{N-1}}}\right] \tag{60}$$

allora $\frac{\partial f}{\partial z} = 0$ o $\frac{\partial f}{\partial \overline{z}} = 0$ 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:

$$df = \frac{\partial f}{\partial z}dz + \frac{\partial f}{\partial \overline{z}}d\overline{z}. \tag{61}$$

Se $f$ é reale allora avremmo:

$$df = 2Re\left\{\frac{\partial f}{\partial z}dz\right\}. \tag{62}$$

Definendo ora l'operatore gradiente come:

$$\nabla_z := \left(\frac{\partial}{\partial \overline{z}}\right)^T \tag{63}$$

$$= \left(\frac{\partial}{\partial z}\right)^* \tag{64}$$

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

$$J_{f\circ g} = J_f J_g + J_f^{(c)}\overline{\left(J_g^{(c)}\right)} \tag{65}$$

$$J_{f\circ g}^{(c)} = J_f J_g^{(c)} + J_f^{(c)}\overline{(J_g)}. \tag{66}$$

Supponiamo di avere una funzione composta $(f \circ g \circ h)(z, \overline{z})$, 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 ( $\frac{\partial h}{\partial \overline{z}=0}$ ), applichiamo la *chain rule*:

$$J_f = \frac{\partial f}{\partial g} \tag{67}$$

$$J_f^{(c)} = \frac{\partial f}{\partial \overline{g}} \tag{68}$$

$$J_{f\circ g} = J_f \frac{\partial g}{\partial h} + J_f^{(c)}\overline{\left(\frac{\partial g}{\partial \overline{h}}\right)} \tag{69}$$

$$J_{f\circ g\circ h} = J_{f\circ g}\frac{\partial h}{\partial z} \tag{70}$$

$$\nabla_z f = (J_{f\circ g\circ h})^* \tag{71}$$

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.

## 3.5 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 difficoltá nella formazione e nelle prestazioni, infatti, come riprenderó piú avanti, le reti neurali complesse non sempre convengono e

se comparate alle reti neurali a valori reali, soprattutto in problemi naturamente nel dominio reale e non complesso. In tal caso infatti introducendo i numeri complessi si ottengono risultati non molto distanti, ma con una complessitá di calcolo maggiore. Il numero di parametri reali é una metrica per quantificare la capacitá di una rete nella sua capacitá di approssimare funzioni strutturalmente complesse. Con troppi parametri il modello tende a overfittare i dati mentre con troppi pochi tende a underfittare. Spesso la letteratura peró ignora il numero di parametri, finendo col confrontare reti di dimensioni notevolmente diverse. Bisogna garantire che queste architetture siano comparabili per dimensioni e capacitá del modello. Per definire la dimensione della rete si puó 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 $a + \mathbf{i}b$ (utilizzando quindi due numeri reali) é dato dal fatto che, a paritá 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 capacitá il numero di parametri a valore reale per layer deve essere uguale o almeno il piú simile possibile. In questo modo le differenze di prestazione sono causate dall'introduzione di numeri complessi come parametri e non da una differenza di capacitá. Si consideri il numero di parametri in uno strato 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 é:

$$p_R = (d \times m) + m, \quad p_C = 2(d \times m) + 2m \tag{72}$$

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

$$p_R = d \times m + k(m \times m) + m \times c, \tag{73}$$

$$p_C = 2\left[d \times m + k(m \times m) + m \times c\right] \tag{74}$$

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

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

$$(1 \times d)\ \overbrace{(d \times m_1)}^{Input\ layer} \rightarrow (1 \times m_1)\ \overbrace{(m_1 \times m_2)}^{Hidden\ layer} \tag{75}$$

$$\rightarrow (1 \times m_2)\ \overbrace{(m_2 \times m_3)}^{Hidden\ layer} \rightarrow (1 \times m_3)\ \overbrace{(m_3 \times m_4)}^{Hidden\ layer} \tag{76}$$

$$\rightarrow (1 \times m_4)\ \overbrace{(m_4 \times m_5)}^{Hidden\ layer} \rightarrow (1 \times m_5)\ \overbrace{(m_5 \times c)}^{Hidden\ layer} \tag{77}$$

$$\rightarrow\ \overbrace{(1 \times c)}^{Model\ ouput} \tag{78}$$

Mentre per il caso complesso:

$$(1 \times n)(n \times m_1/2) \rightarrow (1 \times m_1/2)(m_1/2 \times m_2) \tag{79}$$

$$\rightarrow (1 \times m_2)(m_2 \times m_3/2) \rightarrow (1 \times m_3/2)(m_3/2 \times m_4) \tag{80}$$

$$\rightarrow (1 \times m_4)(m_4 \times m_5/2) \rightarrow (1 \times m_5/2)(m_5/2 \times c) \tag{81}$$

$$\rightarrow (1 \times c) \tag{82}$$

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 $c$.

$$m_R = \begin{cases} -\frac{n+c}{2k} + \sqrt{\frac{n+c}{2k}^2 + \frac{p_R}{k}}, & \text{se } k > 0 \\ \frac{p_R}{n+c} & \text{altrimenti} \end{cases} \tag{83}$$

$$m_C = \begin{cases} -\frac{n+c}{2k} + \sqrt{\frac{n+c}{2k}^2 + \frac{p_R}{k}}, & \text{se } k > 0 \\ \frac{2(p_R)}{n+c} & \text{altrimenti} \end{cases} \tag{84}$$

Confrontando reti neurali reali e complesse di capacitá 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 dal segnale in cui i dati di input hanno una interpretazione fisica nel dominio complesso[**?**] [**?**] [**?**] [**?**] [**?**] [**?**]. Il pregio principale delle reti neurali a valore complesso é quello di riuscire a gestire meglio il rumore sul piano complesso. Per confrontare reti neurali a valori reali con quelli a valori complessi, Nils Mönning e Suresh Manandhar in $Evaluation\ of\ Complex{-}Valued\ on\ Real{-}Valued\ Classification\ Tasks$ [**?**] 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 83 e 84 e viene arrotondato per eccesso.

É stata utilizzata l'inizializzazione dei pesi discussa da Trabelsi et al. [**?**] per tutti gli esperimenti. Per ridurre l'impatto dell'inizializzazione, ogni modello é stato addestrato 10 volte, con training composte da 100 epochs. Come funzione d'attivazione per l'ultimo layer é stata usata sempre la $sigmoid(|z|^2)$ o la $softmax(|z|^2)$.

Le tabelle 1,2,3,4 mostrano i risultati per reti neurali con dimensione variabile e budget illimitato per i parametri (esperimento 1). Le tabelle 5,6,7,8 mostrano invece i risultati dell'esperimento 2.

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

Table 1: 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 | *identity* | 0.8116 | 0.7936 |
| | | *tanh* | 0.8117 | 0.7912 |
| | | *ReLU* | 0.8081 | 0.7934 |
| | | $\|z\|^2$ | 0.8050 | 0.7885 |
| | | $\|z\|$ | 0.8068 | 0.7992 |
| k=2 | 651136 | *identity* | 0.8005 | 0.7836 |
| | | *tanh* | 0.7978 | 0.7320 |
| | | *ReLU* | 0.7921 | 0.7854 |
| | | $\|z\|^2$ | 0.7725 | 0.6874 |
| | | $\|z\|$ | 0.7996 | 0.7823 |
| k=4 | 659328 | *identity* | 0.7925 | 0.7787 |
| | | *tanh* | 0.7814 | 0.4199 |
| | | *ReLU* | 0.7734 | 0.7671 |
| | | $\|z\|^2$ | 0.5895 | 0.0650 |
| | | $\|z\|$ | 0.7863 | 0.7694 |
| k=8 | 675712 | *identity* | 0.7929 | 0.7796 |
| | | *tanh* | 0.7542 | 0.1861 |
| | | *ReLU* | 0.7555 | 0.7676 |
| | | $\|z\|^2$ | 0.0053 | 0.0053 |
| | | $\|z\|$ | 0.7671 | 0.7524 |

Table 2: 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 | *identity* | 0.4044 | 0.1063 |
| | | *tanh* | 0.4885 | 0.1431 |
| | | *ReLU* | 0.4902 | 0.4408 |
| | | $\|z\|^2$ | 0.5206 | 0.1000 |
| | | $\|z\|$ | 0.5256 | 0.1720 |
| k=2 | 427264 | *identity* | 0.4039 | 0.1000 |
| | | *tanh* | 0.5049 | 0.1672 |
| | | *ReLU* | 0.5188 | 0.4960 |
| | | $\|z\|^2$ | 0.1451 | 0.1361 |
| | | $\|z\|$ | 0.5294 | 0.1000 |
| k=4 | 460032 | *identity* | 0.4049 | 0.1000 |
| | | *tanh* | 0.4983 | 0.1549 |
| | | *ReLU* | 0.8445 | 0.6810 |
| | | $\|z\|^2$ | 0.1000 | 0.1000 |
| | | $\|z\|$ | 0.5273 | 0.1000 |
| k=8 | 525568 | *identity* | 0.4005 | 0.1027 |
| | | *tanh* | 0.4943 | 0.1365 |
| | | *ReLU* | 0.5072 | 0.4939 |
| | | $\|z\|^2$ | 0.1000 | 0.1000 |
| | | $\|z\|$ | 0.5276 | 0.1000 |

Table 3: 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 | *identity* | 0.1758 | 0.0182 |
| | | *tanh* | 0.2174 | 0.0142 |
| | | *ReLU* | 0.1973 | 0.1793 |
| | | $\|z\|^2$ | 0.2314 | 0.0158 |
| | | $\|z\|$ | 0.2423 | 0.0235 |
| k=2 | 438784 | *identity* | 0.1720 | 0.0100 |
| | | *tanh* | 0.2314 | 0.0146 |
| | | *ReLU* | 0.2400 | 0.2123 |
| | | $\|z\|^2$ | 0.0143 | 0.0123 |
| | | $\|z\|$ | 0.2411 | 0.0100 |
| k=4 | 471552 | *identity* | 0.1685 | 0.0100 |
| | | *tanh* | 0.2178 | 0.0157 |
| | | *ReLU* | 0.2283 | 0.2059 |
| | | $\|z\|^2$ | 0.0109 | 0.0100 |
| | | $\|z\|$ | 0.2313 | 0.0100 |
| k=8 | 537088 | *identity* | 0.1677 | 0.0100 |
| | | *tanh* | 0.2000 | 0.0130 |
| | | *ReLU* | 0.2111 | 0.1956 |
| | | $\|z\|^2$ | 0.0100 | 0.0100 |
| | | $\|z\|$ | 0.2223 | 0.0100 |

Table 4: 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 | |
|---|---|---|---|---|---|---|
| | $m_R$ | | $m_C$ | | R | C |
| k=0 | 162 | | 81 | *identity* | 0.4335 | 0.1006 |
| | | | | *tanh* | 0.5032 | 0.1676 |
| | | | | *ReLU* | 0.5007 | 0.4554 |
| | | | | $\|z\|^2$ | 0.5179 | 0.1006 |
| | | | | $\|z\|$ | 0.5263 | 0.2381 |
| k=2 | 148 | | 77 | *identity* | 0.4069 | 0.1000 |
| | | | | *tanh* | 0.5205 | 0.1673 |
| | | | | *ReLU* | 0.5269 | 0.4963 |
| | | | | $\|z\|^2$ | 0.1395 | 0.1273 |
| | | | | $\|z\|$ | 0.5315 | 0.1000 |
| k=4 | 138 | | 74 | *identity* | 0.4052 | 0.1000 |
| | | | | *tanh* | 0.5218 | 0.1475 |
| | | | | *ReLU* | 0.5203 | 0.4975 |
| | | | | $\|z\|^2$ | 0.1065 | 0.1010 |
| | | | | $\|z\|$ | 0.5234 | 0.1000 |
| k=8 | 123 | | 69 | *identity* | 0.4050 | 0.1003 |
| | | | | *tanh* | 0.5162 | 0.1396 |
| | | | | *ReLU* | 0.5088 | 0.4926 |
| | | | | $\|z\|^2$ | 0.1000 | 0.1000 |
| | | | | $\|z\|$ | 0.5194 | 0.1000 |

Table 5: Test di accuratezza di reti con $k + 2$ layer, con un budget di 500000 parametri reali. L'output é composto da $c = 10$ neuroni. Esperimento 2.

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

Table 6: Test di accuratezza di reti con $k + 2$ layer, con un budget di 500000 parametri reali. L'output é composto da $c = 46$ neuroni. Esperimento 2.

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

Table 7: Test di accuratezza di reti con $k + 2$ layer, con un budget di 50000 parametri reali. L'output é composto da $c = 10$ neuroni. Esperimento 2.

| Hidden layers $k$ | Units | | $p_R$ | Activation function $\phi$ | CIFAR-100 | |
|---|---|---|---|---|---|---|
| | $m_R$ | | $m_C$ | | R | C |
| k=0 | 158 | | 79 | *identity* | 0.2807 | 0.0314 |
| | | | | *tanh* | 0.2308 | 0.0193 |
| | | | | *ReLU* | 0.2153 | 0.1935 |
| | | | | $\|z\|^2$ | 0.2364 | 0.0124 |
| | | | | $\|z\|$ | 0.2439 | 0.0279 |
| k=2 | 144 | | 75 | *identity* | 0.1723 | 0.0100 |
| | | | | *tanh* | 0.2440 | 0.0203 |
| | | | | *ReLU* | 0.2481 | 0.2224 |
| | | | | $\|z\|^2$ | 0.0155 | 0.0151 |
| | | | | $\|z\|$ | 0.2453 | 0.0100 |
| k=4 | 135 | | 72 | *identity* | 0.1727 | 0.0100 |
| | | | | *tanh* | 0.2397 | 0.0150 |
| | | | | *ReLU* | 0.2381 | 0.2147 |
| | | | | $\|z\|^2$ | 0.0122 | 0.0100 |
| | | | | $\|z\|$ | 0.2390 | 0.0100 |
| k=8 | 121 | | 67 | *identity* | 0.1706 | 0.0100 |
| | | | | *tanh* | 0.2209 | 0.0164 |
| | | | | *ReLU* | 0.2167 | 0.2027 |
| | | | | $\|z\|^2$ | 0.0100 | 0.0100 |
| | | | | $\|z\|$ | 0.2191 | 0.0100 |

Table 8: Test di accuratezza di reti con $k + 2$ layer, con un budget di 50000 parametri reali. L'output é composto da $c = 100$ neuroni. Esperimento 2.

Confrontando i diversi risultati ottenuti con le funzioni di attivazione si nota come la funzione identitá, ma 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. Per dimostrare come invece siano effettivamente migliori nel caso di analisi di segnali esprimibili nel dominio complesso, si fatica a trovare confronti diretti come quelli mostrati in precedenza, ma si trovano in letteratura molti casi di applicazioni di successo. Per citarne alcuni: in [**?**] vengono utilizzate le reti neurali a valori complessi per rimuovere il rumore sonoro da registrazioni audio lasciando la parte "parlata"; in [**?**] vengono utilizzate le reti neurali complesse per rilevare difetti nei metalli analizzando le risposte dei materiali agli ultrasuoni; [**?**] le utilizza per la "voice processing" e [**?**] per stimare informazioni su un determinato paesaggio ottenute tramite radar satellitari. 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 giá anticipato in precedenza, qua si propone di applicare le reti neurali a valori complessi per il MRI fingerprinting, che verrá trattato nella sezione 5 piú approfonditamente. Di seguito vengono trattati dei metodi per poter migliorare la generalizzazione e la resistenza all'overfitting della rete neurale.

# 4 Ottimizzazione

## 4.1 ensemble di reti neurali

Una rete di dimensioni finite raramente apprende completamente un particolare mapping e puó generalizzare male. Aumentare la dimensione o il numero di hidden layer peró, il piú delle volte, non porta a nessun miglioramento [**?**] anzi, aumentando la dimensionalitá del problema aumenta la possibilitá che la rete si stabilizzi lontana dalla soluzione desiderata. Molti ricercatori hanno dimostrato che la semplice combinazione degli output di molti classificatori puó generare previsioni piú accurate di quelle di qualsiasi classificatore [**?**] [**?**]. In particolare, la combinazione di reti neurali addestrate separatamente (comunemente indicato come ensemble di reti neurali) si é dimostrata particolarmente efficace [**?**] [**?**] [**?**] [**?**] [**?**]. Figura 4 illustra la struttura generale di un ensemble di reti neurali. Ogni rete nell'ensemble é per prima cosa addestrata e poi, per ogni esempio del training, l'output predetto di ognuna delle reti é combinato dal judge per produrre l'output dell'ensemble.

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 [**?**]. 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 figura **??** e **??** si mostra l'errore di una singola rete rispetto alla dimensione

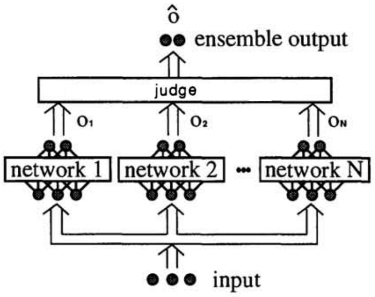


Figure 4: da inserire

dell'hidden layer riscontrato da [**?**] per diversi step di training. D'altra parte la miglior capacitá 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 [**?**] vengono confrontate le prestazioni di ogni singola rete con le prestazioni di un enseble di 5 reti neurali, con stessa struttura della singola.

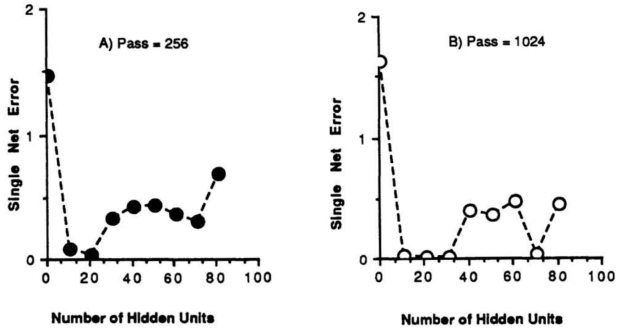


Figure 5: Errore della singola rete in funzione del numero di hidden layer, in dimostrazione del fatto che la relazione non é lineare.

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.

### 4.1.1 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 finale $Y$. Esistono diversi metodi per calcolare $Y$, ma per

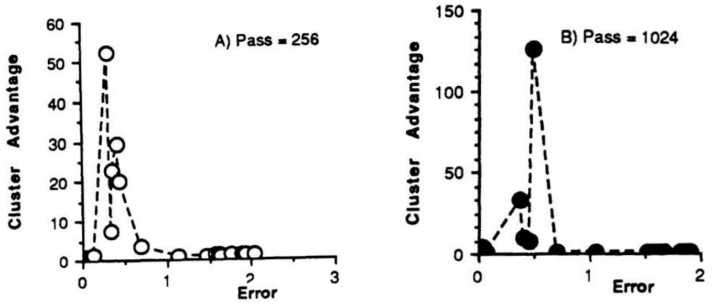


Figure 6: Vantaggio del cluster in funzione dell'errore. A) dopo 256 step del training e B) dopo 1024 step del training.

semplicitá consideriamo i due casi piú semplici:

$$Y = \sum_{k=1}^{N} \frac{1}{N} Y_k \tag{85}$$

$$Y = \sum_{k=1}^{N} W_k Y_k \tag{86}$$

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

$$W_k = W_k \cdot G \cdot \frac{e}{e_k} \tag{87}$$

$$e = \frac{1}{N} \sum k = 1^N e_k \tag{88}$$

$$e_k = |Y - Y_k| \tag{89}$$

Con $e$ che indica il tasso di correzione e $e_k$ é 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 piú soggetta ad ulteriore training. Tuttavia, come giá anticipato, si puó 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 migliori la resistenza al rumore e l'auto-organizzazione (*dacapirecosasia*).

### 4.1.2 ottimizzazione dell'ensemble

Precedenti lavori sia teorici [**?**] [**?**] che empirici [**?**] [**?**] 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 $f : R^N \to R$ per la quale abbiamo $p$ coppie input-output per il training con $Y_k = f(x_k)$ e

$k = 1, \cdots, p$. L'ensenble 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:

$$\overline{V}(x) = \sum_{\alpha} w_{\alpha} V^{\alpha}(x). \tag{90}$$

Consideriamo i pesi $w_{\alpha}$ come l'affidabilitá della rete $\alpha$ e di conseguenza li vincoliamo ad essere positivi e normalizzati, cioé $\sum_{\alpha} w_{\alpha} = 1$. L'*ambuiguitá* su un input $x$ di un singolo membro dell'ensemble é definito come $a^{\alpha}(x) = \left(V^{\alpha}(x) - \overline{V}(x)\right)^2$. La *ambiguitá dell'ensemble* su un input $x$ é data da:

$$\overline{a}(x) = \sum_{\alpha} w_{\alpha} a^{\alpha}(x) = \sum_{\alpha} w_{\alpha} \left(V^{\alpha}(x) - \overline{V}(x)\right)^2. \tag{91}$$

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

$$e^{\alpha}(x) = \left(f(x) - V^{\alpha}(x)\right)^2 \tag{92}$$

$$e(x) = \left(f(x) - \overline{V}(x)\right)^2 \tag{93}$$

Aggiungendo e sottraendo $f(x)$ alla 91 si ottiene:

$$\overline{a}(x) = \sum_{\alpha} w_{\alpha} e^{\alpha}(x) - e(x). \tag{94}$$

Chiamando la media pesata degli errori delle singole reti $\overline{e}(x) = \sum_{\alpha} w_{\alpha} e^{\alpha}(x)$ allora la 93 diventa:

$$e(x) = \overline{e}(x) - \overline{a}(x). \tag{95}$$

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

$$E^{\alpha} = \int dx\, p(x)\; e^{\alpha}(x) \tag{96}$$

$$A^{\alpha} = \int dx\, p(x)\; a^{\alpha}(x) \tag{97}$$

$$E = \int dx\, p(x)\; e(x). \tag{98}$$

Le prime due sono rispettivamente l'errore generalizzato e l'ambiguitá della rete $\alpha$ e $E$ é l'errore generalizzato dell'ensemble. Dalla 95 otteniamo:

$$E = \overline{E} - \overline{A} \tag{99}$$

con $\overline{E} = \sum_{\alpha} w_{\alpha} E^{\alpha}$ la media pesata degli errori generalizzati delle singole reti e $\overline{A} = \sum_{\alpha} w_{\alpha} A^{\alpha}$ la media pesata delle ambiguitá delle singole reti. Questa equazione separa l'errore generalizzato in un termine che dipende dagli errori generalizzati 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 generalizzato 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 generalizzato dell'ensemble é sempre minore della media pesata degli errori degli ensemble: $E < \overline{E}$. In particolare, per pesi uniformi:

$$E \leq \frac{1}{N} \sum_{\alpha} E^{\alpha} \tag{100}$$

Dalla 99 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 generalizzato. Come aumentare l'ambiguitá dell'ensemble? Un metodo puó essere quello di utilizare diverse tipologie di reti neurali, oppure di utilizzare set di training diversi. Inoltre, per essere in grado di stimare il primo termine in 99, sarebbe auspicabile una cross-validation. In [**?**] viene testato questo metodo per approssimare con un ensemble di reti neurali un'onda quadra in una variabile, mostrata in figura 7. Sono state utilizzate 5 reti con un hidden layer di 20 "neuroni", addestrate indipendenemente le une dalle altre mediande 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 8 viene mostrato l'errore

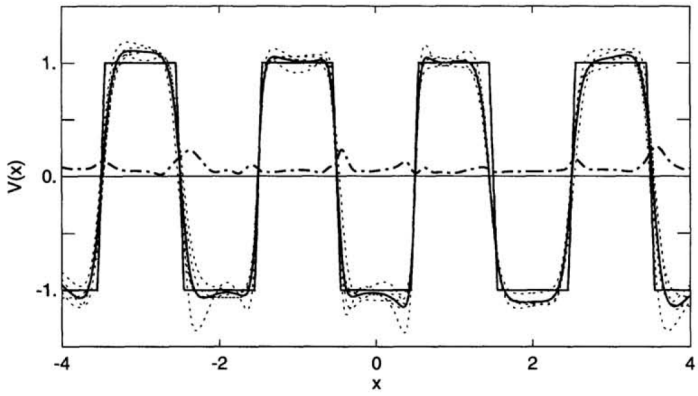


Figure 7: La linea spessa e continua indica l'output dell'ensemble, mentre quelle tratteggiate gli output delle singole reti. Viene mostrata anche l'ambiguitá (dash-dot line).

generalizzato in funzione della dimensione $K$ dei set per la cross-validation. Innanzitutto notiamo come l'errore generalizzato 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

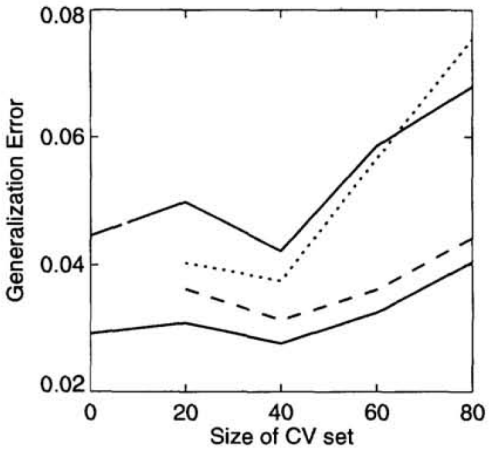


Figure 8: La linea continua mostra l'errore generalizzato per pesi uniformi in funzione di $K$, dove $K$ é la dimensione dei set di convalida. La linea tratteggiata é l'errore stimato dall'equazione 99. La linea tratteggiat é per i pesi ottimali stimati dall'uso degli errori di generalizzazione per le singole reti, stimate dai set di crossvalidation. La linea continua inferiore é l'errore generalizzato che si otterrebbe se i singoli errori di generalizzazione fossero conosciuti esattamente.

modo in cui generare un buon 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 accurato 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 piú performanti e al contempo piú 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:

$$Fitness_i = Accuracy_i + \lambda Diversity_i = (1 - E_i) + \lambda D_i \tag{101}$$

con la diversitá definita come:

$$D_i = \sum \left[ V_i(x) - \overline{V}(x) \right]^2 . \tag{102}$$

Definiamo il termine di accuratezza del set di convalida della rete come $A_i = 1 - E_i$ e utilizziamo l'equazione 102 per calcolare il termine di diversitá $D_i$. 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 $\widehat{E}$ diminuisce mentre consideriamo nuove reti. Cambia invece se si verifica una

delle seguenti: (1) se l'errore della popolazione $\overline{E}$ non sta aumentando e la diversitá $D$ sta diminuendo, aumentiamo $\lambda$; (2) se $\overline{E}$ sta aumentando e $\overline{D}$ non decresce, caliamo $\lambda$.

Tabella **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 103.
  - Misurare la diversitá di ogni rete rispetto alla popolazione corrente (equazione 102).
  - Normalizzare i termini di accuratezza e diversitá delle singole reti.
  - Cacolare la funzione di fitness in equazione 101.
  - Aggiornare la popolazione utilizzando le $N$ reti con valore maggiore per la 101.
  - Corregere $\lambda$ dell'equazione 101
  - Utilizzare la popolazione attuale di reti come ensemble e combinare gli output delle reti seguendo l'equazione 87

Ribadiamo 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 preoccupiamo 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:

$$Cost = \sum_{k \in T} \left| \frac{t(k) - \widehat{o}(k)}{\widehat{E}} \right|^{\frac{\lambda}{\lambda+1}} [t(k) - a(k)]^2 \tag{103}$$

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 nostra rete non é ancora un membro dell'ensemble, $\widehat{o}(k)$ e $\widehat{E}$ non dipendono dalla nostra rete; il nuovo termine é perció una costante quando calcoliamo le derivate nella back propagation. Normalizziamo $t(k) - \widehat{o}(k)$ dividendo per l'errore dell'ensemble $\widehat{E}$ in modo che il valore medio del nostro nuovo termine sia circa 1 indipendentemente dalla correttezza dell'insieme. Ció é particolarmente importante con popolazioni particolarmente accurate, poiché $t(k) - \widehat{o}(k)$ sará vicino a 0 per la maggior parte degli esempi e la rete verrebbe addestrata solo su un numero esiguo di esempi. L'esponenete $\frac{\lambda}{\lambda+1}$ 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. Combiniamo le previsioni delle reti prendendo una somma ponderata dell'output di ciascuna rete, dove ogni peso é definito come in precedenza. Riportiamo di seguito i risultati ottenuti da David W. Opitz e Jude W. Shavlik [**?**]. L'algoritmo genetico che viene utilizzato per generare nuove topologie di rete é l'algoritmo REGENT [**?**]. 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 [**?**], 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 opolazione 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 ció che attualmente é noto su tale attivitá [**?**] [**?**]. Gli esperimenti misurano l'errore del set di test dell'algoritmo genetico su queste attivitá. Ogni ensemble é composto da 20 reti e gli algoritmi REGENT e ADDEMUP (cosí é stato chiamato l'algoritmo genetico utilizzato in [**?**]) hanno considerato 250 reti durante la loro ricerca genetica. La tabella 9 presenta i risultati del caso in cui si crei casualmente la topologia delle reti. La prima riga della tabella 9, la migliore rete, deriva da una rete neurale a singolo strato in cui, per ogni fold abbiamo addestrato 20 reti contenenti tra 0 e 100 hidden nodes 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 [**?**] 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", cioé 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. Breiman (1994) [**?**] ha dimostrato che il bagging é efficace su algoritmi di apprendimento "instabili", come le reti neurali, in cui piccoli cambiamenti nel set di addestramento comportano grandi cambiamenti nelle previsioni. La riga inferiore della tabella 9, 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 domini la combinazione dell'output di piú reti addestrate di generalizza meglio rispetto al tentativo di scegliere la rete singola rete migliore. Mentre la tabella 9 mostra la potenza degli insiemi di reti neurali, la tabella 10 mostra la capacitá di ADDEMUP di utilizzare le conoscenze a priori del problema. La prima riga della tabella 10 contiene i risultati di generalizzazione dell'algoritmo KBANN, mentre la riga successica, 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 9). La riga successiva, REGENT-combined, contiene i risultati della semplice combinazione, usando l'equazione 87, degli output delle reti nella popolazione finale di REGENT. ADDEMUP, l'ultima riga della tabella

10, differisce principalmente da REGENT-combined in due modi: (1) la sua funzione di fitness (equazione 101 ) tiene conto della diversitá 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 diversitá di ADDEMUP, anche se i risultati aggiuntivi riportati in Opitz [**?**] 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 nella tabella 10 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.

Table 9: 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% |

Table 10: 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% |

## 4.2 Funzione d'attivazione sticastica

La funzione d'attivazione gioca un ruolo fondamentale nel training delle rete 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 sará 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 giá 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 condominio 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 comoditá la riprendiamo di seguito:

$$ReLU\,(x) = max\,(x)\,. \tag{104}$$

La sua derivata é data da:

$$ReLU'(x) = \begin{cases} 1 & \text{se} \quad x < 0 \\ O & \text{altrimenti} \end{cases} \tag{105}$$

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 ReLU7. L'idea di utilizzare una funzione di attivazione stocastica viene dal comportamento degli impulsi nervosi, i cui picchi sono disturbati da effetti bio-meccanici [**?**]. 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 9 visualizziamo i principali effetti dovuti all'applicazione di questa nuova funzione d'attivazione.

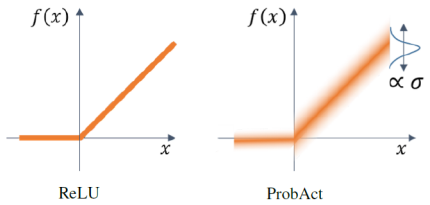


Figure 9: Confronto tra ReLU e funzione d'attivazione stocastica. L'effetto sfumato é stato introdotto solo per visualizzare come la funzione d'attivazione stocastica abbia come valore medio proprio $ReLU(x)$.

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

$$Y = f\left(w^T x\right), \tag{106}$$

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 9 introduciamo una variabile randomica nella ReLU per creare una funzione d' attivazione stocastica. In particolare la si puó definire come segue:

$$f(x) = \mu(x) + z \tag{107}$$

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

$$z = \sigma\epsilon. \tag{108}$$

Il parametro perturbato $\sigma$ puó essere fisso o addestrabile e definisce il range della perturbazione stocastica ed $\epsilon$ é il valore ottenuto dalla distribuzione normale. Studiamo i casi in cui $\sigma$ sia fissa o addestrata. Nel caso sia fissa, ci sono diversi modi per definirne il valore piú adatto. Scegliere una $\sigma$ costante per

tutti i valori é una opzione valida poiché $\epsilon$ é ottenuta randomicamente da una distribuzione normale e $\sigma$ funge da fattore di scala. 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 vantaggio principale di fissare $\sigma$ é quello di non aumentare il numero di parametri da dover addestrare nel training. 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:

$$\frac{\partial F}{\partial \sigma_{l,i}} = \frac{\partial F}{\partial y_{l,i}} \frac{\partial y_{l,i}}{\partial \sigma_{l,i}} \tag{109}$$

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:

$$\frac{\partial y_{l,i}}{\partial \sigma_{l,i}} = \epsilon \tag{110}$$

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 $\sigma = \alpha sigmoid(\beta k)$ per avere $\sigma$ confinata tra 0 e $\alpha$. Con $k$ il parametro da imparare e $\alpha$ e $\beta$ parametri che posso essere tenuti fissi. Negli esperimenti descritti di seguito avremo $\alpha = 2$ e $\beta = 5$ (valori scelti dopo un test apposito). Figura 9 illustra gli effetti della perturbazione stocastica. 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 aai pesi del primo e secondo layer come segue:

$$y_2 = \mu\left[w_2\mu\left(w_1 x\right) + w_2\sigma_1\epsilon\right] + \sigma_2\epsilon \tag{111}$$

$$= \begin{cases} \mu\left[w_2\mu\left(w_1 x\right)\right] + w_2\sigma_1\epsilon + \sigma_2\epsilon & \text{se } \ w_2\mu\left(w_1 x\right) + w_2\sigma_1\epsilon > 0; \\ \sigma_2\epsilon & \text{altrimenti} \end{cases} \tag{112}$$

$$\sim \begin{cases} N\left(\mu\left[w_2\mu\left(w_1 x\right)\right], \left(w_2\sigma_1\right)^2 + \sigma_2^2\right) \\ N\left(0, \sigma_2^2\right). \end{cases} \tag{113}$$

Come mostrato in figura 9, tenderá ad essere appresa una piccola varianza del rumore nello strato finale, per fare in modo che l'output della rete sia stabile. 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:

$$h = g(d, z) \tag{114}$$

Se utilizziamo l'equazione 114 per altri metodi di aggiunta di rumore come il dropout [**?**], 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.

Valutiamo ora in alcuni esperimenti la funzione d'attivazione stocastica descritta in [**?**], applicata alla classificazione di immagini. Mostriamo 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.

Table 11: Confronto dell'accuratezza ottenuta dalla rete utilizzando diverse funzioni di attivazione. Il valore ottenuto é dato dalla media su tre set di test.

| Activation function | CIFAR-10 | CIFAR-100 | STL-10 | Train time | Test time |
|---|---|---|---|---|---|
| ReLU | 86.67% | 52.94% | 60.80% | **1.00 X ReLU** | **1.00 X ReLU** |
| 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 | **1.00 X ReLU** |
| Swish | 86.55% | 54.01% | 63.50% | 1.20 X ReLU | 1.13 X ReLU |
| 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% | **58.45**% | 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 Trainable $\sigma$ (unbound) | 86.40% | 53.10% | 61.70% | 1.25 X ReLU | 1.31 X ReLU |
| Element-wise Trainable $\sigma$ (bound) | **88.92**% | 55.83% | **64.17**% | 1.26 X ReLU | 1.33 X ReLU |

Per valutare le performance del metodo proposto si confronta la funzione d' attivazione *ProbAct* con: ReLU, LeakyReLU, PReLU, e Swosh[**?**]. Si utilizza una rete a 16 layer Visual Geometry Group (VGG). Non vengono utilizzati metodi di regolarizzazione ulteriori o per-training o data augmentation. Gli

input vengono normalizzati. Le immagini tratte da STL-10 sono state ridotte di soluzione 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 10 é mostrato l' andamento del training di $\sigma$.

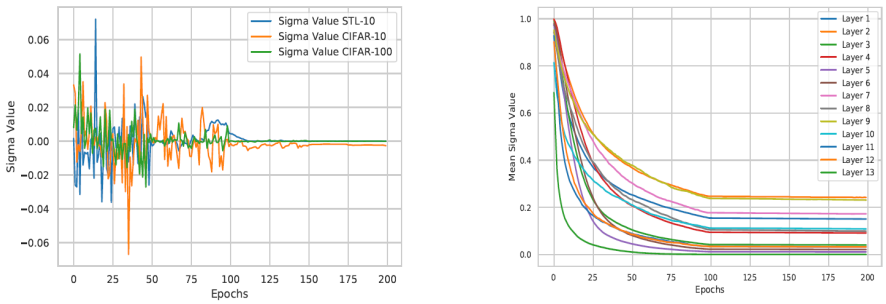


Figure 10: A sinistra la transizione del singolo sigma addestrabile nelll'architettura VGG-16 sui tre set di dati. A destra il valore medio dei $\sigma$ addestrati sul set di dati CIFAR10, evidenziando i diversi valori per ogni layer.

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 11 mostra la distribuzione in frequenza dell' elemento $k$ dopo 400 epochs. Osserviamo due picchi per ogni distribuzione attraverso i 3 dataset.

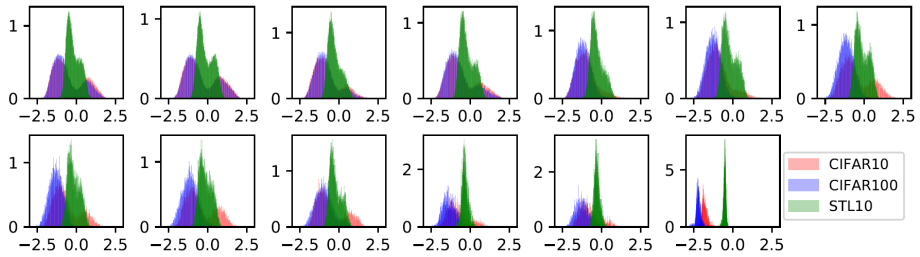


Figure 11: Distribuzione in frequenza dell' elemento $k$ dopo l'allenamento dell'architettura VGG-16 per i set di dati CIFAR-10, CIFAR-100 e STL-10 per ogni layer. L'asse X indica il valore $k$ dopo l'allenamento e l'asse Y indica la sua frequenza. Ogni sottofigura rappresenta un layerin cui viene applicata 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:

$$\gamma = TrainAccuracy(\%) - TestAccuracy(\%). \tag{115}$$

Se $\gamma$ é piccolo, la rete in seguito al training si é generalizzata bene anche al set di test. Se invece é grande, la rete ha imparato molto bene a classificare gli esempi del training, ma non riesce a classificare casi diversi da quelli giá visti. In figura 12 si confrontano le abilitá di generalizzazione con e senza l'utlizzo di dropout per il dataset CIFAR-100.

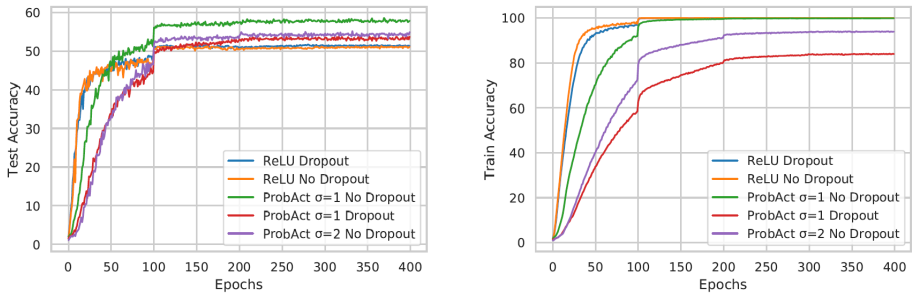


Figure 12: Confronto tra ReLU e funzione d'attivazione stocastica, sia con che senza dropout, utilizzando il dataset CIFAR-100. In particolare a sinistra il confronto é effettuato nel set di test e a destra nel set di training.

Il dropout viene applicato con una probabilitá 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 ció sono stati confrontati i risultati di reti identiche, una con funzione d'attivazione ReLU e l'altra con funzione d'attivazione stocastica (ProbAct [**?**]), utilizzando solo una frazione dei dataset CIFAR-10 e CIFAR-100. I risultati ottenuti sono mostrati in tabella 12.

Table 12: 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 | **84.73%** | **46.11%** | **79.02%** | **31.67%** |

# 5 Applicazione delle reti neurali complesse al 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. 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.

## 5.1 introduzione al MRF

La risonanza magnetica ha lo svantaggio di essere lenta rispetto ad altri strumenti diagnostici ed é generalmente qualitativa: il principale mezzo di informazione utilizzato per caratterizzare una patologia é 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 variabilitá nell'interpretazione e potenzialmente limita l'utilitá 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 proprietá dei tessuti come i tempi di rilassamento T1 e T2. La quantificazione delle proprietá dei tessuti consente ai medici di distinguere meglio tra tessuto sano e patologico in senso assoluto, rende piú semplice il confronto oggettivo di esami diversi e potrebbe essere piú rappresentativo dei cambiamenti rispetto all'imaging ponderato standard. Si ottiene quindi il vantaggio di una minore soggettivitá dei risultati, che puó aiutare nella diagnosi, in particolare nella caratterizzazione dei tessuti.

Queste tecniche si basano sull'acquisizione di piú 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 necessitá di mantenere costanti tutti i parametri della sequenza tranne uno e la necessitá 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 piú rapide, tra le quali quella che decriveró a breve.

Come appena accennato, la maggior parte degli approcci MR attuali consente la misurazione delle proprietá dei tessuti, ma sono relativamente lenti e generalmente forniscono una sola proprietá alla volta. Inoltre, la caratterizzazione delle patologie dipende spesso non solo da una singola proprietá, ma da una combinazione di proprietá 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 T1 e T2. La tecnica che approfondisco é il Magnetic Resonance Fingerprinting (MRF). Il MRF é un potente strumento diagnostico, prognostico e di valutazione della terapia che, grazie alla sua natura versatile rispetto ad altre modalitá di imaging, consente di sondare diversi parametri contemporaneamente. Tuttavia permangono importanti ostacoli all'adozione clinica, in particolare una necessitá di una quantificaizione al contempo rapida e accurata. Il MRF cambia completamente il modo in cui viene eseguita la risonanza magnetica quantitativa con un approccio completamente divesto da quello delle tecniche convenzionali. Questa tecnica mira a fornire misurazioni simultanee di piú parametri come T1, T2 densitá di spin relativa, disomogeneitá $B_0$ ecc., utilizzando un'unica acquisizione, efficiente in termini di tempo.

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. É possibile simulare le 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 T1, T2, 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, di conseguenza 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 T1, T2 (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 peró 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 T1 e T2 come parametri di interesse. Ció comporta 18838 voci di dizionario che possono essere calcolate in circa 8 minuti su un computer standard e genera un dizonario 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 presenta di una qualitá limitata. 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 volte per un dizionario TrueFISP e fino a 4.8 volte per un dizionario FISP. La compressione del dizionario basato su SVD ha una riduzione della precisione dei parametri stimati inferiore al 2%. 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 ruduce 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 algoritmmo 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 riduce la velocitá di calcolo della corrispondenza di un ordine di grandezza rispetto alla compressione SVD e due oridini di grandezza rispetto alla corrispondenza diretta senza perdita significativa nella qualitá della corrispondenza. La tecnica del MRF viene riassunta nell'immagine 13.

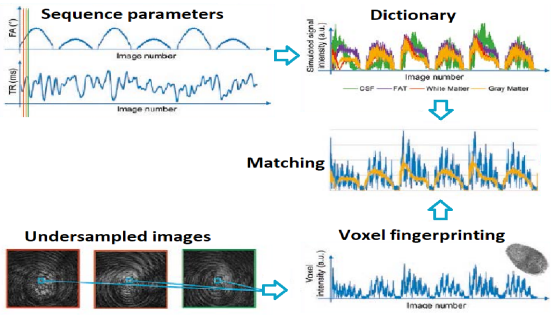


Figure 13: In alto é mostrata la generazione del dizionario, mentre in basso si ha la misura effettuata. Viene poi effettuato il matching per determinare quale segnale del dizionario si avvicina a quello effettivamente misurato.

## 5.2 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 T1 e T2. 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 dal 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 peró 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 [**?**]. 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 piú 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 [**?**] viene definita una rete neurale con 4 layer (uno di input, 2 hidden layer e un layer di output) utilizzando il TensorFlow [**?**]. L'input consiste di 25 o 50 nodi e l'output da 2 nodi in quanto sono studiati solo il T1 e il T2. Ogni hidden layer é 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 é stato impostato come 0.001 e la funzione di costo é data dall'errore quadratico medio. Il training viene effettuato sfruttando un dizionario con circa 69000 esempi, mentre per la convalida é stato sfruttato il BrainWeb digital brain phantom insieme a simulazioni montecarlo, introducendo rumore gaussiano. L'accuratezza della rete é comparabile a quella dei metodi tradizionali di MRF e in particolare é stato riscontrato una migliore efficienza in caso di rumore o undersampling.

## 5.3 MRF con reti neurali a valori complessi

Dati i risultati ottenuti con le reti neurali a valori reali (giá 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 é stato trattato anche in [**?**], 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 [**?**];

Come risultato in [**?**] si ottiene che le reti neurali in generale sono piú efficaci nella predizione dei valori T1 e T2, 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 T1 che su T2, mentre prevale solo nell'accuratezza su T1 in caso di segnale con rumore.

Questa non é che una proposta di applicazione e in generale di studio di reti neurali a valori complessi nel campo del MRF. Le reti neurali presentano ancora margini di miglioramento ed in particolare quelle a valori complessi che fino a non molto tempo fa suscitavano scarso interesse per via delle difficoltá di calcolo per la back-propagation. Nel caso in cui effettivamente possano risultare un'alternativa concreta alle tecniche attuali, non solo si riuscirebbero a misurare piú parametri MR contemporaneamente, ma si potrebbe avere un riscontro quasi immediato e di tipo quantitativo.