Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea Magistrale in Matematica

Reti Neurali e Apprendimento Automatico per dati NMR

Tesi di Laurea in Analisi Numerica

Relatrice:
Chiar.ma Prof.ssa
Fabiana Zama

Presentata da:
Federico Vallese

Correlatori:
Dr. Davide Evangelista
Dr. Giovanni Vito Spinelli

Anno Accademico 2023-2024

Indice

Introduzione

La Risonanza Magnetica Nucleare (RMN) rappresenta una metodologia fondamentale per analizzare e definire diverse proprietà chimico-fisiche di un'ampia varietà di materiali. Le peculiarità della RMN, in qualità di metodo analitico, risiedono nella sua natura non invasiva e nella sua capacità di distinguere sottili variazioni nella composizione chimica, nell'aggregazione e nella mobilità molecolare.

Attualmente, la RMN si articola in tre principali settori: rilassometria, spettroscopia e imaging. La rilassometria indaga l'evoluzione temporale della magnetizzazione nucleare e come questa sia influenzata dalla dinamica molecolare all'interno di un campione. La spettroscopia si concentra sull'analisi dello spettro radio ad alta risoluzione, rivelando le caratteristiche chimiche del campione, come struttura e composizione molecolare. L'imaging, invece, si dedica all'acquisizione di immagini dettagliate delle strutture interne di un campione.

Il focus di questo lavoro riguarda una specifica tecnica di rilassometria chiamata *Fast Field Cycling Nuclear Magnetic Resonance* (FFC-NMR). La FFC-NMR è un approccio analitico distintivo e non invasivo che offre informazioni immediate sulle dinamiche fisiche e le interazioni molecolari presenti in un materiale.

La tecnica FFC-NMR sta guadagnando crescente riconoscimento come strumento potente per esplorare le proprietà fisico-chimiche di vari sistemi in svariati ambiti scientifici. Ad esempio, trova applicazione nell'analisi di problematiche ambientali, nella discriminazione tra differenti tipi di alimenti e nella valutazione di nuovi materiali.

Una delle peculiarità della FFC-NMR è la sua capacità di determinare come la velocità R_1 (o tempo di rilassamento longitudinale T_1, dove $T_1 = 1/R_1$) di un campione varia modificando l'intensità di un campo magnetico applicato, dando origine ai profili NMRD. Questa tecnica consente di rilevare dinamiche su un'ampia gamma di scale temporali in un unico esperimento [1]. Oltre a ciò, l'analisi di rilassamento in funzione della frequenza può svelare i meccanismi sottostanti alla dinamica molecolare.

Tuttavia, l'adozione della FFC-NMR su larga scala è limitata dalla complessità degli strumenti analitici e dalla necessità di una profonda competenza in RMN e nella fisica dei materiali. In risposta a questa sfida, questo lavoro propone l'adozione dell'apprendimento automatico attraverso reti neurali specificatamente progettate per l'analisi FFC-NMR.

Alcuni tentativi sono stati già presentati in letteratura riguardo alla spettroscopia [2] e alla rilassometria a basso campo [3]. Tuttavia, specificatamente per la FFC-NMR, la ricerca è ancora agli albori.

In questa tesi, si intraprende un'indagine preliminare sull'impiego di tecniche di apprendimento automatico per modellare la rilassometria FFC-NMR, concentrandosi in particolare sull'apprendimento dei parametri che caratterizzano il *Quadrupolar Relaxation Enhancement* (QRE)[1]. Questo rappresenta un problema inverso non lineare, notoriamente difficile da trattare con tecniche tradizionali, richiedendo metodi avanzati di ottimizzazione e regolarizzazione. Perciò, si è ricorso all'uso di percettroni multistrato (dall'inglese *multilayer perceptrons* o MLP) ovvero dei modelli di rete neurale artificiale che mappano degli insiemi di dati in input in degli insiemi di dati opportuni in output. Nello specifico, singolarmente ad ogni campione analizzato, l'obiettivo è quello di creare una rete ad hoc che, una volta addestrata, sia capace di associare ad ogni curva di dispersione in ingresso una previsione affidabile dei parametri quadrupolari.

Le caratteristiche migliori dell'MLP sono state dedotte al seguito di numerosi esperimenti, e, in virtù del mal condizionamento del problema, è stato posto il focus sulla *loss function* migliore da utilizzare durante la fase di addestramento. Nello specifico, è stata ideata un'opportuna combinazione convessa tra la media degli errori assoluti commessi sulla previsione dei parametri e sulla ricostruzione del grafico del profilo NMRD. Facendo ciò, la rete impara a prevedere i parametri tenendo contemporaneamente anche conto della forma della curva di dispersione corrispondente.

Per quanto concerne la struttura dell'elaborato in questione, in primo luogo sono state trattate le premesse teoriche fondamentali alla comprensione delle caratteristiche del modello scelto 1 e l'applicazione fisica del problema 2. Successivamente, si è analizzato il metodo di risoluzione del problema mediante ottimizzazione, e si è effettuato un confronto con l'approccio innovativo sopracitato 3. Infine, variando le caratteristiche del modello, sono stati eseguiti diversi esperimenti aventi come fine il miglioramento della performance 4.

[1] In un esperimento FFC-NMR possono verificarsi interazioni dinamiche nucleari complesse come il QRE, il quale è dovuto all'accoppiamento magnetico dipolare intramolecolare tra nuclei quadrupolari con spin $S \geq 1$. Una conseguenza di questo effetto si manifesta nella presenza di massimi locali nel profilo NMRD dovuti a fenomeni di risonanza. La posizione di questi picchi dipende fortemente da parametri chimico-fisici quadrupolari che si vuole stimare.

Capitolo 1

Fondamenti dell'Apprendimento Automatico

Poiché il *Deep Learning* (DL) è un'applicazione specifica di *Machine Learning* (ML), è necessario approfondirne le basi al fine di poter comprendere al meglio le caratteristiche di tale metodologia di apprendimento.

Il *Machine Learning*, o apprendimento automatico, è un sottoinsieme dell'intelligenza artificiale (AI), ovvero della disciplina che studia se, e in che modo, si possano realizzare sistemi informatici intelligenti in grado di simulare la capacità e il comportamento del pensiero umano.

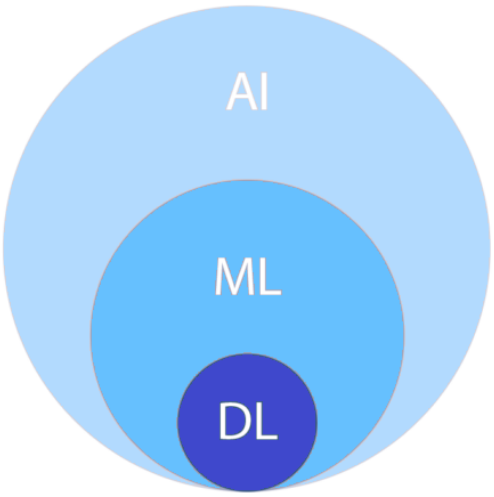


Figura 1.1: Diagramma di Venn che mostra il legame tra *Artificial intelligence*, *Machine Learning* e *Deep Learning*.

Il termine *Machine Learning* è stato definito da Arthur Lee Samuel nel 1959 come il campo di studi che fornisce ai computer l'abilità di apprendere senza essere programmati esplicitamente.
Un'ulteriore definizione particolarmente esaustiva è quella data nel 1997 dal computer scientist americano Tom Michael Mitchell [4]:
"Diciamo che un programma al computer apprende dall'esperienza E rispetto ad alcune classi di compiti T e rispetto alla misura della performance P, se la sua performance nei compiti T, come misurato da P, migliora con l'esperienza E.".

Compiti T

Nei sistemi di ML, i compiti dipendono da come dovrebbe essere processato un dato esempio. Gli esempi sono delle collezioni di valori che sono state opportunamente misurate da un oggetto o evento d'interesse. Esempi tipici, solitamente rappresentati sotto forma di vettore, sono i tensori che definiscono le immagini RGB o i campionamenti di una data funzione.

Due dei compiti maggiormente noti sono:

- Classificazione

  Chiediamo al programma di specificare a quali delle k categorie appartenga l'input. Solitamente l'obiettivo è quello di produrre una funzione della forma

  $$f : \mathbb{R}^n \to \{1, 2, \ldots, k\},$$

  in questo caso l'input viene direttamente etichettato con un valore intero tra 1 e k, estremi compresi.

  Una variante tipica al tema è quella di associare ad ogni input un codice numerico, come ad esempio una distribuzione di probabilità finita su $\{1, 2, \ldots, k\}$.

  In tal caso

  $$f : \mathbb{R}^n \to [0, 1]^k,$$

  ovvero mappiamo ogni input in un vettore lungo k, il cui i-esimo elemento indica la probabilità che l'input abbia etichetta i.

  L'esempio di classificazione più famoso è il riconoscimento di oggetti, come facce, espressioni, bevande ecc.. Tali modelli risolutivi possono essere molto complessi e, per questo motivo, spesso richiedono l'uso di reti neurali profonde tipiche del *Deep Learning*.

- Regressione

  In questa tipologia di compito, al programma è chiesto di predire un valore numerico dato l'input. Per risolvere tale problema, tipicamente, chiediamo all'algoritmo di produrre in output una funzione

  $$f : \mathbb{R}^n \to \mathbb{R}.$$

  Questo compito è simile al precedente, eccetto per la forma continua dell'output.
  Esempi di regressione sono la predizione della richiesta dell'importo atteso che farà una persona assicurata, o la previsione dei prezzi futuri dei titoli.

Performance P

Al fine di valutare l'abilità predittiva di un algoritmo di ML, è necessario definire una misura della sua performance P, che è spesso fortemente dipendente dal compito che deve svolgere il sistema. Ad esempio, nel caso della classificazione, è spesso uile calcolare la *accuracy*, o precisone, del modello, ovvero il rapporto tra il numero

di esempi predetti correttamente e quello totale. Un'altra possibile performance in questo contesto è l'*error rate* (anche nota come *0-1 loss*), che, al contrario della precedente, calcola la percentuale di esempi non opportunamente etichettati.

Per quanto riguarda invece i modelli di regressione, non è possibile calcolare l'*accuracy*, infatti, in questo caso, l'obiettivo delle nostre previsioni non è quello di predire i valore target in maniera esatta, ma vogliamo sapere quanto le previsioni risultano vicine ai valori che ci aspettiamo. In questo contesto la performance viene misurata da metriche che prevedono il calcolo dei residui, come ad esempio l'Errore Quadratico Medio o *Mean Squared Error* (MSE). Approfondiremo tale metrica nella sezione successiva, quando tratteremo il tema dell'addestramento e della funzione di costo di una rete neurale.

Poiché è di grande interesse sapere quanto l'algoritmo di ML performa su dati non utilizzati nell'addestramento, valutiamo la misura di tale performance usando un insieme *test* di dati distinti da quello usato durante la fase di addestramento del modello.

Esperienza E

In un modello di ML, l'esperienza E fa riferimento ai dati usati durante la fase di addestramento. In letteratura si è soliti suddividere gli algoritmi di ML in diverse categorie [6]. Le tre maggiormente note sono:

- Apprendimento supervisionato (o *Supervised learning*)

  Il modello viene addestrato su un *dataset* provvisto di etichette (o *label*), cioè su un insieme di esempi di cui è già noto l'output desiderato.

  Per questo motivo, gli algoritmi che risolvono problemi di classificazione e regressione appartengono tradizionalmente a questa tipologia.

- Apprendimento non supervisionato (o *Unsupervised learning*)

  Il *dataset* usato durante la fase di addestramento non è etichettato, perciò l'algoritmo deve essere capace di individuare autonomamente dei *pattern* all'interno dell'insieme di esempi al fine di classificarli.

  Il problema classico che si avvale di algoritmi dall'apprendimento non supervisionato è il *clustering* che consiste nella divisione del *dataset* in sottogruppi contenenti esempi simili.

- Apprendimento per rinforzo (o *Reinforcement learning*)

  In questo caso c'è un ciclo di feedback tra il sistema di apprendimento e le sue esperienze. Gli algoritmi di ML in questione non fanno esperienza su un *dataset* fissato.

Uno stesso compito può essere risolto da strategie differenti basate su tipologie di apprendimento diverse.

Gli algoritmi di ML più semplici funzionano bene su una grande varietà di problemi importanti, ma non sono in grado di risolvere tutti i problemi centrali nell'ambito dell'intelligenza artificiale. Per venire in contro a questa esigenza sono stati costruiti modelli in cui il processo di apprendimento avviene su più livelli, questi prendono il nome di reti neurali artificiali o *deep feedforward networks* e sono alla base del *Deep Learning* (DL).

1.1 Introduzione alle Reti Neurali Artificiali

Deep feedforward networks, chiamati anche *feedforward neural networks*, o *multilayer perceptrons* (MLPs), sono i modelli di Deep Learning per eccellenza [4][5].
L'obiettivo di una *feedforward neural networks* è quello di approssimare una qualche funzione f^* che mappa $\mathbf{X} \in \mathbb{R}^{n \times d}$, ovvero un minibatch di n esempi dove ogni esempio ha d inputs (features), nell'output $\mathbf{Y} \in \mathbb{R}^{n \times q}$.

Otteniamo dunque

$$\mathbf{Y} = f^*(\mathbf{X}).$$

Una rete *feedforward* definisce un mapping della forma

$$\mathbf{O} = f(\mathbf{X}; \boldsymbol{\theta}) \in \mathbb{R}^{n \times q}$$

e impara il valore dei pesi e dei bias $\boldsymbol{\theta}$ affinché la funzione f approssimi al meglio f^*.

Questi modelli sono chiamati *feedforward* poiché i dati in input si propagano in avanti, senza cicli, attraverso la rete, passando per uno o più strati di neuroni, fino a raggiungere l'output.
Il termine *network*, invece, si riferisce al fatto che questi siano tipicamente rappresentati dalla composizione di differenti funzioni. Il modello, infatti, può essere rappresentato da un grafo aciclico che descrive come le funzioni interagiscono tra loro. Per esempio, supponiamo di avere le funzioni f_1 e f_2, interconnesse tra loro a catena nel seguente modo: $f(\mathbf{X}) = f_2(f_1(\mathbf{X}))$. Queste strutture a catena sono le più comunemente utilizzate nelle reti neurali, e, in questo caso, f_1 prende il nome di primo strato (o *layer*) della rete, mentre f_2 è il secondo. In generale, la lunghezza della catena fornisce la profondità del modello. Il nome *Deep Learning* nasce da questa terminologia. Infine, queste reti sono chiamate *neurali* poiché l'origine di queste risiede nella neuroscienza. Ogni nodo della rete ricorda un neurone, nel senso che riceve input da molte altre unità e calcola il suo valore di attivazione.

1.1.1 Struttura di una Rete Neurale Artificiale

Un MLP ha tre gradi di libertà:

- L *layer*, di cui l'ultimo costituisce l'output, mentre gli altri sono interni (*hidden layers*). L prende il nome di profondità della rete,
- un vettore di lunghezza L, in cui ciascun elemento indica il numero di nodi nel *layer* corrispondente. Si è soliti indicare tale vettore con il termine *width*,
- le funzioni di attivazione tra i neuroni.

Il numero di nodi in input dipende dai dati del problema.

Ad esempio, consideriamo il seguente *multilayer perceptron*

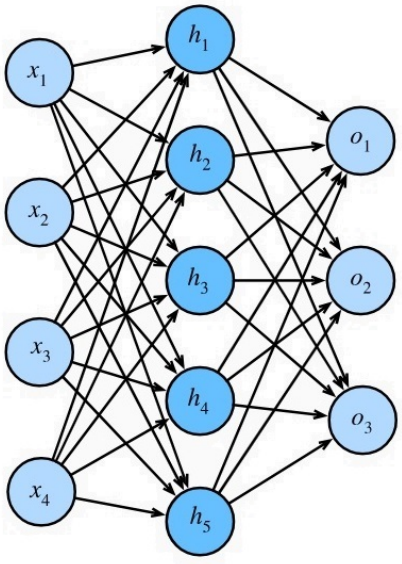


Figura 1.2: MLP con 4 neuroni nel *layer* in input, 5 nel nascosto e 3 nell'output. Fonte: [5].

Tale rete consiste di 4 input, 3 output e un singolo *hidden layer* contenente 5 unità. Produrre degli output usando questa architettura richiede implementare calcoli sia per ogni *layer* ad esclusione del primo; perciò, il numero di *layer* nell'MLP in esempio è 2.
Osserviamo come tutti i *layer* siano completamente connessi, ovvero per ogni coppia di nodi appartenenti a *layer* consecutivi, esiste un arco che li connette. Ogni input influenza, quindi, ogni neurone nei *layer* nascosti, e, di conseguenza, anche ogni neurone presente nell'ultimo strato.

1.1.2 *Forward Propagation*

L'informazione all'interno della rete fluisce a partire dal *layer* in input e attraversa con ordine tutti i successivi fino a raggiungere quello in output.
In generale l'input viene trasformato in maniera non lineare, ma procediamo con ordine e momentaneamente introduciamo un modello semplificato.

Come sopra, denotiamo con $\mathbf{X} \in \mathbb{R}^{n \times d}$, un *minibatch* di n esempi dove ogni esempio ha d inputs (*features*). Per un MLP avente un singolo *layer* nascosto composto di h unità interne, denotiamo con $\mathbf{H} \in \mathbb{R}^{n \times h}$, l'output del *layer* nascosto. Dal momento che il *layer* nascosto e quello in output sono completamente connessi, abbiamo la matrice dei pesi riferita al *layer* nascosto $\mathbf{W}^{(1)} \in \mathbb{R}^{d \times h}$ e vettore *bias* $\mathbf{b}^{(1)} \in \mathbb{R}^{1 \times h}$ e la matrice dei pesi riferita al *layer* in output $\mathbf{W}^{(2)} \in \mathbb{R}^{h \times q}$ e vettore *bias* $\mathbf{b}^{(1)} \in \mathbb{R}^{1 \times q}$. Ciò ci permette di calcolare l'output $\mathbf{O} \in \mathbb{R}^{n \times q}$ della rete a due *layer* complessivi, nel seguente modo:

$$\mathbf{H} = \mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}$$
$$\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$$

In questo stiamo concatenando due trasformazioni lineari, e formalmente possiamo condensarle in un'unica nel seguente modo

$$\mathbf{O} = (\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})\mathbf{W}^{(2)} + \mathbf{b}^{(2)} = \mathbf{X}\mathbf{W}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)} = \mathbf{X}\mathbf{W} + \mathbf{b}$$

dove $\mathbf{W} = \mathbf{W}^{(1)}\mathbf{W}^{(2)}$ e $\mathbf{b} = \mathbf{b}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}$.

Possiamo generalizzare tale trasformazione dell'input in maniera non lineare introducendo un ulteriore ingediente fondamentale, ovvero una funzione di attivazione non lineare da applicare ad ogni unità nascosta e, quindi, ad ogni trasformazione affine. La funzione di attivazione più comunemente utilizzata è la ReLU (*Rectified Linear Unit*), ed è una funzione lineare a tratti, che associa ad un elemento il massimo tra questo e lo 0.

Applicando le funzioni di attivazione ai nodi del grafo, in generale, non è più possibile ricondurre l'MLP in un modello lineare:

$$\mathbf{H} = \sigma(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)}),$$
$$\mathbf{O} = \mathbf{H}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}.$$

Al fine di costruire MLP più generali, possiamo aumentare il numero di *layer* nascosti, ed ottenere ad esempio:

$$\mathbf{H}^{(1)} = \sigma_1(\mathbf{X}\mathbf{W}^{(1)} + \mathbf{b}^{(1)})$$

e

$$\mathbf{H}^{(2)} = \sigma_2(\mathbf{H}^{(1)}\mathbf{W}^{(2)} + \mathbf{b}^{(2)}).$$

Il calcolo e la memorizzazione delle variabili intermedie (output compreso) che avviene in una rete neurale, in ordine dal *layer* di input a quello di output, prende il nome di *forward propagation* o propagazione in avanti.

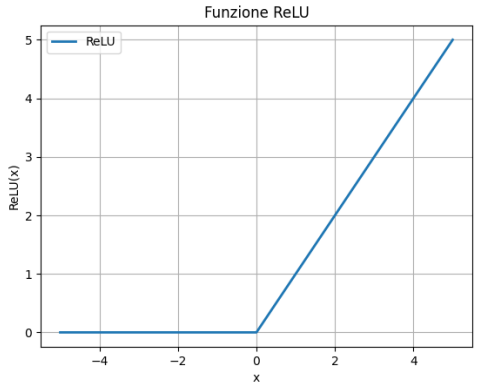
Rectified Linear Unit (ReLU)

Le funzioni di attivazione ReLU sono la scelta consigliata nella stragrande maggioranza delle *Deep feedforward networks*.

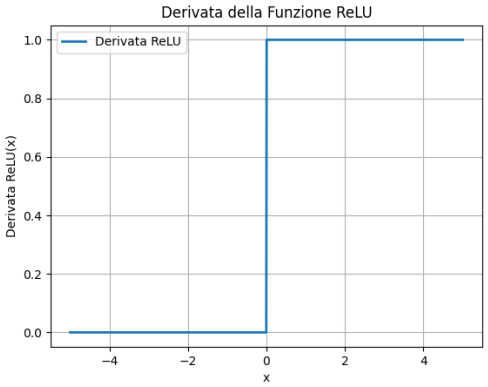
Queste hanno la seguente forma

$$\sigma(x) = \max(0, x)$$

dove l'operazione sull'argomento è da intendere un elemento per volta. In maniera informale le funzioni ReLU ritornano in output solo elementi positivi o nulli, e scartano tutti i negativi fissando l'attivazione corrispondente a 0.



(a) Grafico della funzione *Rectified Linear Unit*.



(b) Grafico della derivata della funzione *Rectified Linear Unit*.

Applicando tale funzione all'output di una trasformazione affine, si rende questa non lineare. Ciononostante il risultato sarà vicino all'essere lineare in virtù della natura della funzione ReLU, che è lineare a tratti. Poiché le ReLU sono vicine all'essere lineari, queste preservano molte delle proprietà che rendono i modelli lineari facili da ottimizzare usando metodi che si basano sul calcolo dei gradienti.

Osserviamo che la funzione ReLU non è differenziabile nello 0, per evitare problemi possiamo fissare manualmente a 0 la derivata della funzione in quel punto. Tale passaggio potrebbe non essere necessario dal momento che il singoletto $\{0\}$ ha misura nulla rispetto al dominio della funzione, quindi questa risulta essere differenziabile quasi ovunque.

Sigmoide

La funzione sigmoide è una mappa della seguente forma:

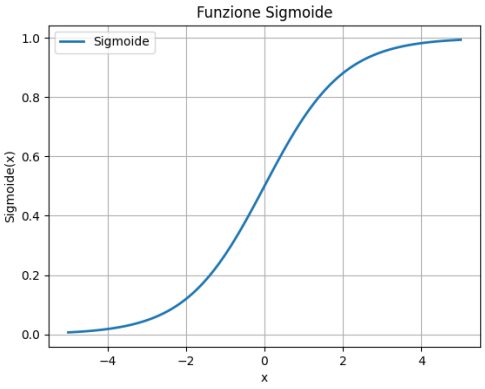
$$sigmoid : \mathbb{R} \to (0, 1)$$

che agisce sugli elementi del dominio come:

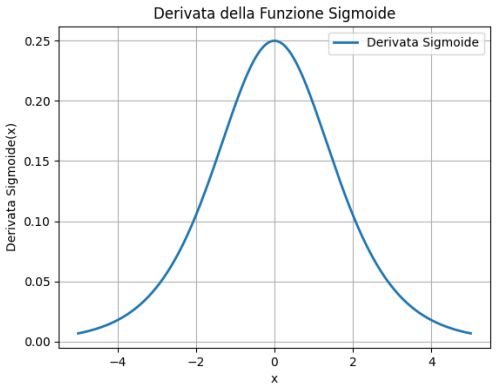
$$sigmoid(x) = \frac{1}{e^{-x} + 1}$$

Osserviamo come, in prossimità dello 0, la funzione si comporta come una trasformazione lineare.

Tale applicazione trasforma l'input reale in un valore positivo miniore di 1. Per via di questa proprietà, secondo cui ogni numero nel range $(-\infty, \infty)$ ha come immagine un valore che giace nell'intervallo $(0, 1)$, si è soliti riferirsi a tale funzione chiamandola *squashing function*.



(a) Grafico della funzione sigmoide.



(b) Grafico della derivata della funzione sigmoide.

Le sigmoidi sono ancora ampiamente utilizzate come funzioni di attivazioni sulle unità di output, infatti queste ci permettono di interpretarlo come una probabilità per quanto riguarda problemi di classificazione binaria. Per il resto, queste sono state ampliamente sostituite dalle più semplici e facilmente addestrabili ReLU relativamente ai *layer* nascosti. Il motivo alla base di questo è che la derivata della sigmoide si annulla a tendere del modulo dell'argomento a $+\infty$, e la presenza di questi *plateau* non limitati può risultare un grande problema durante la fase di ottimizzazione.

L'equazione della derivata della sigmoide è la seguente

$$\frac{d}{dx} sigmoid(x) = \frac{e^{-x}}{(1 + e^{-x})^2} = sigmoid(x)(1 - sigmoid(x))$$

il massimo di tale funzione è 0.25, ed è raggiunto quando l'input è 0. Inoltre, come riportato in precedenza, la funzione va a 0 quando l'input diverge in modulo.

Tangente iperbolica

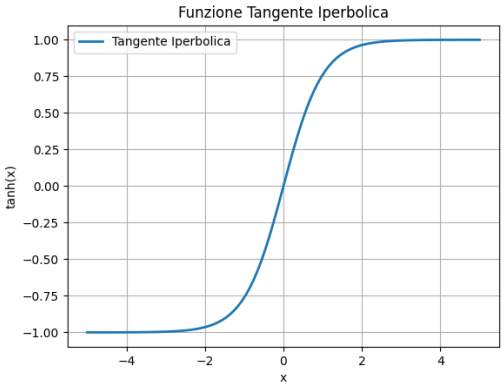
La tangente iperbolica ha un comportamento molto simile alla sigmoide, infatti

$$tanh : \mathbb{R} \to (-1, 1)$$

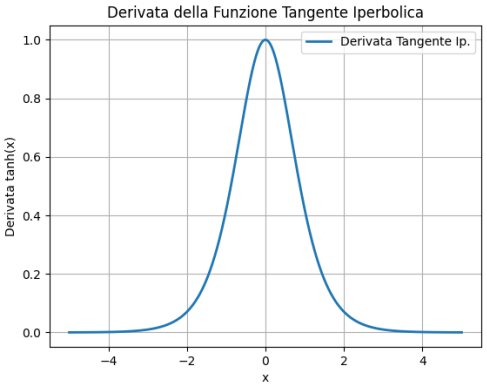
Nello specifico, l'azione sull'input è la seguente:

$$tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

All'avvicinarsi dell'input a 0, anche questa funzione si comporta come un'applicazione lineare. Inoltre l'origine del sistema di assi cartesiani rappresenta un punto di simmetria per la funzione.



(a) Grafico della funzione tangente iperbolica.



(b) Grafico della derivata della funzione tangente iperbolica.

Calcoliamone la derivata:

$$\frac{d}{dx}tanh(x) = \frac{2e^{-2x}(1 + e^{-2x}) + 2e^{-2x}(1 - e^{-2x})}{(1 + e^{-2x})^2} = \frac{4e^{-2x}}{(1 + e^{-2x})^2}$$

Sommando al numeratore $(e^{-2x})^2 - (e^{-2x})^2 + 1 - 1$ e raccogliendo i due quadrati, otteniamo

$$\frac{d}{dx}tanh(x) = 1 - tanh^2(x)$$

il massimo di tale funzione è 1, ed è raggiunto quando l'input è 0. Inoltrela funzione va a 0 quando il modulo dell'input diverge a $+\infty$.

Osserviamo, inoltre, che è possibile scrivere la sigmoide in funzione della tangente iperbolica

$$-2sigmoid(2x) + 1 = tanh(x).$$

Infatti,

$$-2sigmoid(2x) + 1 = -2\frac{1}{e^{-2x} + 1} - 1 = \frac{e^{-2x} - 1}{e^{-2x} + 1} = tanh(x).$$

1.1.3 Addestramento e funzione di costo

Al fine di stabilire la relazione che lega correttamente i valori in input $\mathbf{X} = \{\mathbf{X}_1, .., \mathbf{X}_n\}$ a quelli in output $\mathbf{O} = \{f(\mathbf{X}_1), .., f(\mathbf{X}_n)\}$ è fondamentale trovare i valori ottimali dei pesi e bias $\boldsymbol{\theta}$. Ciò avviene nella cosidetta fase di addestramento, che consiste in un problema di minimo di una funzione di costo o perdita (in inglese *loss function*), che quantifica la lontananza dell'output dal risultato ricercato.

Due delle funzioni di costo più comunemente utilizzate sono la *Mean Squared Error* (MSE) e la *Mean Absolut Error* (MAE) [7].

- ***Mean Squared Error*** **(MSE)**
  Per quanto concerne l'errore quadratico medio, dato l'evento i-esimo, denotiamo con $f(\mathbf{X}_i)$ la sua previsione e con $\mathbf{Y}_i$ la sua immagine per mezzo della funzione f^*. L'errore quadratico è dato da:

  $$\ell^{(i)}(\boldsymbol{\theta}) = \frac{1}{2}\|f(\mathbf{X}_i) - \mathbf{Y}_i\|_2^2,$$

  questo dipende dai pesi e bias $\boldsymbol{\theta}$ del modello e la costante $\frac{1}{2}$ non è utile ai fini della valutazione ma è una notazione opportuna poiché si cancella in seguito al calcolo della derivata della *loss*. Al fine di misurare la qualità del modello su un intero dataset composto da n esempi, semplicemente calcoliamo la media degli errori quadratici, ottenendo dunque l'errore quadratico medio

  $$\mathcal{L}(\boldsymbol{\theta}) = MSE(\boldsymbol{\theta}) := \frac{1}{n}\sum_{i=1}^{n} \ell^{(i)}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \frac{1}{2}\|f(\mathbf{X}_i) - \mathbf{Y}_i\|_2^2.$$

- ***Mean Absolut Error*** **(MAE)**
  Nelle ipotesi precedenti, definiamo errore assoluto

  $$\ell^{(i)}(\boldsymbol{\theta}) = \|f(\mathbf{X}_i) - \mathbf{Y}_i\|_1.$$

  Quindi, l'errore assoluto medio è

  $$\mathcal{L}(\boldsymbol{\theta}) = MAE(\boldsymbol{\theta}) := \frac{1}{n}\sum_{i=1}^{n} \ell^{(i)}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \|f(\mathbf{X}_i) - \mathbf{Y}_i\|_1.$$

Quindi, per quanto riguarda un generico MLP composto da L *layer*, per l=1, 2, ..,L, denotiamo con:

- n_l il numero di neuroni presenti nell'l-esimo strato della rete
- $\mathbf{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ la matrice dei pesi riferita all'l-esimo strato
- $b^{(l)}$ è il vettore bias per l'l-esimo strato
- σ_l è la funzione di attivazione l-esima
- $\mathbf{H}^{(l)} = \sigma_l(\mathbf{H}^{(l-1)}\mathbf{W}^{(l)} + \mathbf{b}^{(l)})$ è l'output dello strato l-esimo

Quindi, osserviamo come $\mathbf{X} = \mathbf{H}^{(0)}$ e $f(\mathbf{X}) = \mathbf{H}^{(L)}$, inoltre

$$\boldsymbol{\theta} = \left(\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, .., \mathbf{W}^{(L)}, \mathbf{b}^{(L)}\right).$$

Vogliamo trovare un insieme di matrici di pesi e vettori *bias* che rendono la *loss function* più piccola possibile. Possiamo fare ciò usando l'algoritmo di discesa del gradiente e sue varianti più efficienti.

1.1.4 Algoritmi di ottimizzazione

Consideriamo la soluzione numerica di un problema di ottimizzazione non vincolato

$$\min_{\mathbf{x}} \phi(\mathbf{x})$$

dove assumiamo che $\phi : \mathbb{R}^m \to \mathbb{R}$ sia differenziabile e limitata dal basso.

Tutti i metodi pratici che risolvono tali problemi di minimo sono di natura iterativa e, partendo da $\mathbf{x}_0$ generano una sequenza $\mathbf{x}_k$ di iterazioni che convergono ad un'approssimazione di un minimo locale di ϕ, [8].

Diciamo che $\phi \in C^1(\mathbb{R}^m)$ è convessa se e solo se $\forall \mathbf{x}, \mathbf{y} \in \mathbb{R}^m$ vale che

$$\phi(\mathbf{y}) > \phi(\mathbf{x}) + \nabla\phi(\mathbf{x})^T(\mathbf{y} - \mathbf{x}).$$

Se ϕ è convessa, allora, ogni minimizzatore locale $\mathbf{x}^*$ è anche globale. Purtroppo tale condizione è molto restrittiva e, nel pratico, non viene mai soddisfatta dalla *loss function* che si vuole minimizzare, per questo motivo, ci limitiamo a calcolarne un suo minimo locale.

Nei cosidetti metodi *line-search*, dato $\mathbf{x}_k$, il nuovo iterato $\mathbf{x}_{k+1}$ è definito come

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \alpha_k \mathbf{p}_k$$

dove

- $\alpha_k \in \mathbb{R}^+$ è l'ampizza del passo (in inglese *step-size*) o tasso di apprendimento,
- $\mathbf{p}_k \in \mathbb{R}^n$ è la direzione di ricerca.

Una direzione di ricerca $\mathbf{p}$ è di discesa per ϕ in $\mathbf{x}$ se questa forma con $-\nabla\phi(\mathbf{x})$ un angolo di ampiezza strettamente inferiore a $\pi/2$ radianti, ovvero deve valere che

$$\mathbf{p}^T \nabla\phi(\mathbf{x}) < 0$$

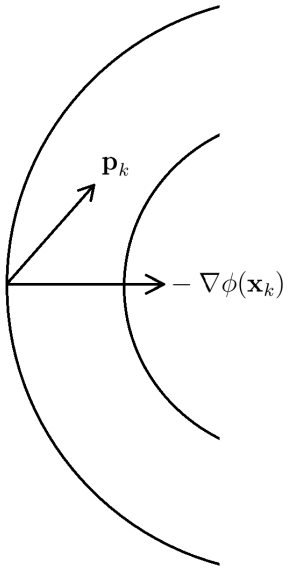


Figura 1.6: Esempio di direzione di discesa.

La direzione di discesa più comunemente utilizzata, $\forall k$, è

$$\mathbf{p}_k = -\nabla\phi(\mathbf{x}_k),$$

e, in tal caso, l'algoritmo iterativo che risolve il problema di minimo prende il nome di *discesa del gradiente* (o *steepest descent*).

Nella strategia risolutiva *line-search*, dopo aver scelto la direzione di discesa corrente, partendo da $\mathbf{x}_k$, si sceglie l'apiezza opportuna del passo al fine di procedere nella ricerca del valore di minimo della funzione.

Possiamo calcolare α_k seguendo diversi criteri:

- Ampiezza del passo costante

  $\alpha_k = \bar{\alpha}$, per qualche scalare non negativo $\bar{\alpha} > 0$.

- Tasso di apprendimento dinamico

  È possibile scegliere α_k affinché questo oscilli tra un valore massimo fissato e uno minimo al variare di k. Un esempio di questa strategia è il *cosine annealing schedule* che approfondiremo in seguito.

- *Exact line search*

  α_k può essere calcolato risolvendo il seguente problema di minimo unidimensionale nella variabile α

  $$\alpha_k = \underset{\alpha>0}{\operatorname{argmin}}\ \phi(\mathbf{x}_k + \alpha \mathbf{p}_k).$$

  Questa metodo risulta, però, inefficiente in generale.

- *Inexact line search*

  Calcoliamo una successione di possibili valori di α_k fino al soddisfacimento di un criterio di arresto, come ad esempio l'*Armijo sufficient decrease condition*

  $$\phi(\mathbf{x}_k + \alpha \mathbf{p}_k) \leq \phi(\mathbf{x}_k) + c\,\alpha \nabla \phi(\mathbf{x}_k)^T \mathbf{p}_k$$

  dove $c \in (0, 1)$ e $\mathbf{p}_k$ è una direzione di discesa.

Mostriamo, quindi, lo pseudocodice del metodo di discesa del gradiente con ampiezza del passo costante.

Algorithm 1 Discesa del gradiente - passo fisso

```
1:  function GD
2:      Input: x_0, φ, k_max, ε_tol, α
3:      for k = 0, 1, ··· , k_max do
4:          p_k = −∇φ_k(x_k)
5:          x_{k+1} = x_k + α p_k
6:          if ||∇φ(x_{k+1})|| ≤ ε_tol then
7:              return x_{k+1} (stop)
8:          end if
9:      end for
10: end function
```

Discesa del gradiente *batch*

La discesa del gradiene *batch*, al fine di computare un singolo passo, usa tutto il dataset di addestramento. Infatti viene calcolata la media dei gradienti di tutti gli esempi di addestramento e poi, per mezzo di tale gradiente medio, aggiorna i parametri.

In questo caso, quindi, al fine di aggiornare i pesi e i *bias* $\boldsymbol{\theta}$ della rete neurale, si considera come funzione oggetto ϕ la *loss function*

$$\mathcal{L}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{i=1}^{n} \ell^{(i)}(\boldsymbol{\theta}).$$

La direzione di discesa al tempo k è della forma

$$\mathbf{p}_k = -\nabla\mathcal{L}(\boldsymbol{\theta}_k) = -\frac{1}{n}\sum_{i=1}^{n} \nabla\ell^{(i)}(\boldsymbol{\theta}_k).$$

Tale strategia può essere computazionalmente molto costosa, specialmente se il dataset è molto esteso, inoltre è necessario che la superficie dell'errore che vogliamo minimizzare sia relativamente liscia e convessa.

Discesa del gradiente stocastico (SGD)

Diversamente dall'approccio precedentemente esposto, la discesa del gradiente stocastico (SGD) vuole essere efficiente anche nel caso di dataset molto estesi, questo perchè i modelli di *Deep Learning*, tendenzialmente, risultano più affidabili tanto più vengono addastrati su un numero maggiore di esempi.

Nell' SGD, per compiere un singolo passo, si considera solo un esempio per volta. Nello specifico, ad ogni passo, si sceglie un esempio da inserire nella rete neurale, si calcola il gradiente e lo si usa per aggiornare i pesi.

In questo caso, la direzione di discesa al tempo k è della forma

$$\mathbf{p}_k = -\nabla\ell^{(i)}(\boldsymbol{\theta}_k),$$

dove i è estratto dall'insieme $\{1, 2, \ldots, n\}$ distribuito in maniera uniforme.

Osserviamo come, in media, il gradiente stocastico $\nabla\ell^{(i)}(\boldsymbol{\theta})$ sia una buona stima del gradiente, infatti vale che

$$\mathbb{E}_{i\sim u[1,\ldots n]}\nabla\ell^{(i)}(\boldsymbol{\theta}) = \frac{1}{n}\sum_{j=1}^{n} \nabla\ell^{(j)}(\boldsymbol{\theta}) = \nabla\mathcal{L}(\boldsymbol{\theta}).$$

Poiché stiamo considerando solo un esempio per volta, la funzione di costo potrebbe uscillare, con il rischio di rallentare la convergenza, incappare in un minimo locale o, in generale, non approssimando in maniera soddisfaciente il minimo della funzione.

Nonostante ciò, l'uso di singoli esempi di addestramento rende l'aggiornamento dei pesi piuttosto veloce e ciò rende l'SGD adatto nel caso di dataset grandi.

Minibatch stochastic gradient descent (Minibatch SGD)

Il *minibatch stochastic gradient descent* (minibatch SGD) è un compromesso tra la dispendiosa discesa del gradiente batch e l'SGD. Infatti, non usiamo tutto il dataset o un singolo esempio, ma preleviamo piccoli sottoinsiemi randomici, di taglia fissata, che prendono il nome di *minibatch*.

Nello specifico, durante la k-esima iterazione, viene selezionato dall'insieme $\{1, 2, \ldots, n\}$ distribuito in maniera uniforme un *minibatch* $\mathcal{B}_k$ da inserire nella rete neurale, si calcola il gradiente medio di questo e si usa per aggiornare i pesi. Successivamente, al variare delle iterazioni, si ripete tale procedura anche con tutti gli altri *minibatch* selezionati.

La direzione di discesa al tempo k è

$$\mathbf{p}_k = -\frac{1}{|\mathcal{B}_k|} \sum_{i \in \mathcal{B}_k} \nabla \ell^{(i)}(\boldsymbol{\theta}_k),$$

Esattamente come nell'SGD, con l'avanzare delle iterazioni, la funzione di costo nel *minibatch stochastic gradient descent* potrebbe fluttuare, soprattuto se il numero di esempi che consideriamo volta per volta è molto piccolo. Questo approccio, rispetto ai precedenti, richiede la configurazione di un iperparametro aggiuntivo, ovvero la taglia dei sottoinsiemi casuali che vogliamo utilizzare [9].

Ottimizzatore Adam

Infine introduciamo un algoritmo di ottimizzazione molto efficiente, ottenuto modificando il *minibatch* SGD affinché questo abbia memoria a lungo termine delle direzioni dei gradienti passati [5].

Per fare ciò definiamo i momenti iniziali $\mathbf{m}_0 = \mathbf{0}$ e $\mathbf{v}_0 = \mathbf{0}$. Ad ogni iterazione aggiorniamo i momenti calcolando delle combinazioni convesse dipendenti dalla direzione di discesa. Per $k = 0, 1, \cdots, k_{max}$, otteniamo

$$\mathbf{m}_{k+1} = \beta_1 \mathbf{m}_k + (1 - \beta_1)\mathbf{p}_k,$$

$$\mathbf{v}_{k+1} = \beta_2 \mathbf{v}_k + (1 - \beta_2)\mathbf{p}_k^2,$$

che intervengono nel calcolo dell'iterato $\mathbf{x}_{k+1}$.

Mostriamo lo pseudocodice dell'algoritmo [10]

Algorithm 2 Ottimizzatore Adam

function ADAM
  Input: $\mathbf{x}_0, \phi, k_{max}, \alpha = 0.001, \beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 1e-08,$
  $\mathbf{m}_0 = \mathbf{0}$, $\mathbf{v}_0 = \mathbf{0}$
  for $k = 0, 1, \cdots, k_{max}$ **do**
    $\mathbf{p}_k = \nabla\phi_k(\mathbf{x}_k)$
    $\mathbf{m}_{k+1} = \beta_1\mathbf{m}_k + (1-\beta_1)\mathbf{p}_k$
    $\mathbf{v}_{k+1} = \beta_2\mathbf{v}_k + (1-\beta_2)\mathbf{p}_k^2$
    $\hat{\boldsymbol{m}}_{k+1} = \mathbf{m}_{k+1}/(1-\beta_1^{k+1})$
    $\hat{\boldsymbol{v}}_{k+1} = \mathbf{v}_{k+1}/(1-\beta_2^{k+1})$
    $\mathbf{x}_{k+1} = \mathbf{x}_k - \alpha\hat{\boldsymbol{m}}_{k+1}/(\sqrt{\hat{\boldsymbol{v}}_{k+1}} + \epsilon)$
  end for
  return $\mathbf{x}_{k+1}$
end function

Dove le operazioni di elevamento a potenza e di rapporto sono da interpretare come puntuali, ovvero un elemento per volta.

Nel metodo Adam, la combinazione del primo e del secondo momento favorisce l'accelerazione della convergenza dell'ottimizzazione e constribuisce ad una gestione adattiva del tasso di apprendimento. Queste caratteristiche rendono tale metodo uno degli ottimizzatori più popolari ed efficaci nell'ambito delle reti neurali.

Tasso di apprendimento dinamico

In un algoritmo di ottimizzazione, la scelta del tasso di apprendimento è fondamentale ai fini della convergenza del metodo alla soluzione.

Dalla letteratura [13], è noto che, sotto la restrittiva ipotesi che la funzione ϕ sia fortemente convessa, ovvero, suppondo che $\exists\, c > 0$ tale che

$$\phi(\mathbf{x}) \geq \phi(\mathbf{y}) + \nabla\phi(\mathbf{y})^T(\mathbf{x}-\mathbf{y}) + \frac{c}{2}\|\mathbf{x}-\mathbf{y}\|_2^2, \quad \forall\, \mathbf{x}, \mathbf{y} \in \mathbb{R}^m,$$

nel caso in cui il tasso di apprendimento sia costante, non è possibile dismostrare la convergenza alla soluzione, ma solamente ad un intorno del valore ottimale. Per questo motivo è opportuno usare un'ampiezza del passo variabile.

Durante la fase di addestramento di una rete neurale è possibile regolare in maniera dinamica il tasso di apprendimento affinché il modello converga più lentamente all'inizio (ovvero quando i pesi sono ancora lontani dalla soluzione ottimale), e poi acceleri l'ottimizzazione quando i pesi si avvicinano a una soluzione soddisfacente [11].

Un esempio che adotta tale strategia è il *cosine annealing schedule* [12]. In questo caso, per $k = 1, \ldots, k_{max}$

$$\alpha_k = \alpha_{min} + \frac{1}{2}(\alpha_{max} - \alpha_{min})\left(1 + \cos\left(\pi \frac{k\%T_{max}}{T_{max}}\right)\right),$$

dove α_{max} e α_{min} sono rispettivamente il tasso di apprendimento massimo e minimo che vogliamo fissare, T_{max} è il numero massimo di epoche in cui vogliamo continuare a ridurre il tasso di apprendimento e il simbolo $\%$ sta ad indicare l'operatore modulo.

Ad esempio mostriamo il grafico dell'andamento del tasso di apprendimento nel caso in cui $\alpha_{max} = 1e-4$, $\alpha_{min} = 0$ e $T_{max} = \frac{1}{3}k_{max}$

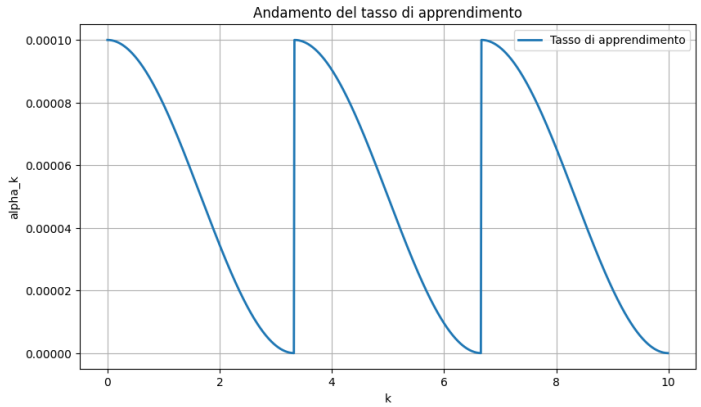


Figura 1.7: Evoluzione del tasso di apprendimento durante le epoche.

Fissando $\alpha_{min} = 0$ e $T_{max} = k_{max}$, otteniamo una successione di $\{\alpha_k\}_{k=0,1,\ldots,k_{max}}$ i cui elementi decrescono fino ad annullarsi, questo motodo può aiutare ad evitare oscillazioni indesiderate e ad accelerare la convergenza.

1.1.5 *Backward Propagation*

La Retropropagazione (in inglese *Backward Propagation* o, più semplicemente, *Backpropagation*) è la fase di calcolo del gradiente della funzione di costo rispetto ai pesi e bias della rete neurale. Tale metodologia, partendo dal *layer* di output, attraversa la rete all'indietro e calcola le derivate parziali della *loss function* $\mathcal{L}$ usando la cosidetta *chain rule*, ovvero la regola di derivazione delle funzioni composte. I gradienti calcolati in questa fase sono necessari all'addestramento della rete. Ciò avviene per mezzo della risoluzione del problema di minimo della funzione di costo grazie ad un algoritmo di ottimizzazione.

Ricordiamo che l'insieme dei pesi e *bias* del modello è definito con

$$\boldsymbol{\theta} = \left(\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, .., \mathbf{W}^{(l)}, \mathbf{b}^{(l)}, .., \mathbf{W}^{(L)}, \mathbf{b}^{(L)}\right),$$

dove le componenti di $\mathbf{W}^{(l)}$ e $\mathbf{b}^{(l)}$ sono rispettivamente $W^{(l)}_{j,k}$ e $b^{(l)}_j$ $\forall l, j, k$.

L'obiettivo è quello di calcolare le derivate parziali della *loss function* rispetto alle variabili $W_{j,k}^{(l)}$ e $b_j^{(l)}$.

Sotto l'ipotesi che $\mathbf{H}^{(l)} \in \mathbb{R}^{n_l}$, definiamo con $\mathbf{z}^{(l)} = \mathbf{H}^{(l-1)}\mathbf{W}^{(l)} + \mathbf{b}^{(l)} \in \mathbb{R}^{n_l}$, dove $l = 2, .., L$, ed osseriviamo come l'output dello strato l-esimo sia, dunque

$$\mathbf{H}^{(l)} = \sigma_l\big(\mathbf{H}^{(l-1)}\mathbf{W}^{(l)} + \mathbf{b}^{(l)}\big) = \sigma_l\big(\mathbf{z}^{(l)}\big).$$

Sia $\boldsymbol{\delta}^{(l)} \in \mathbb{R}^{n_l}$ definita da

$$\delta_j^{(l)} = \frac{\partial \mathcal{L}}{\partial z_j^{(l)}} \tag{1.1}$$

dove $1 \leq j \leq n_l$ e $2 \leq l \leq L$. $\delta_j^{(l)}$ misura il grado di sensibilità della funzione di perdita rispetto all'input pesato per il neurone j-esimo nel livello l-esimo.

Introduciamo ora un'utile operazione

Definizione 1.1.1 (Prodotto di Hadamard)**.** *Dati due vettori $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, il loro prodotto di Hadamard è il vettore in cui ogni elemento è il prodotto tra le componenti dei vettori corrispondenti, otteniamo dunque:*

$$\boldsymbol{x} \circ \boldsymbol{y} = \big(x_i y_i\big)_{i=1,..n} \in \mathbb{R}^n$$

Otteniamo il seguente teorema [14].

Teorema 1.1.1. *Abbiamo che*

$$\boldsymbol{\delta}^{(L)} = \sigma_l'\big(\boldsymbol{z}^{(L)}\big) \circ \frac{\partial \mathcal{L}}{\partial \boldsymbol{z}^{(L)}} \tag{1.2}$$

$$\boldsymbol{\delta}^{(l)} = \sigma_l'\big(\boldsymbol{z}^{(l)}\big) \circ \big(\boldsymbol{W}^{(l+1)}\big)^T \boldsymbol{\delta}^{(l+1)} \quad per \quad l = 2, ..., L-1 \tag{1.3}$$

$$\frac{\partial \mathcal{L}}{\partial b_j^{(l)}} = \delta_j^{(l)} \quad per \quad l = 2, ..., L-1 \tag{1.4}$$

$$\frac{\partial \mathcal{L}}{\partial W_{k,j}^{(l)}} = \delta_k^{(l)} H_j^{(l-1)} \quad per \quad l = 2, ..., L-1 \tag{1.5}$$

Dimostrazione. Poniamo inizialmente l'attenzione su 1.2. Ricordiamo che stiamo supponendo che

$$\mathbf{H}^{(l)} = \sigma_l\big(\mathbf{H}^{(l-1)}\mathbf{W}^{(l)} + \mathbf{b}^{(l)}\big) = \sigma_l\big(\mathbf{z}^{(l)}\big) \in \mathbb{R}^{n_l} \ \ \forall l.$$

e, osserviamo come

$$\mathbf{H}^{(0)} = \mathbf{X} \in \mathbb{R}^{n_0};$$
$$\mathbf{H}^{(L)} = f(\mathbf{X}), \mathbf{Y} \in \mathbb{R}^{n_0}.$$

Nello specifico, vale la relazione $\mathbf{H}^{(L)} = \sigma_l\big(\mathbf{z}^{(L)}\big)$ e quindi

$$\frac{\partial H_j^{(L)}}{\partial z_j^{(L)}} = \sigma_l'\big(z_j^{(L)}\big)$$

Inoltre, coerentemente con quanto supposto sopra,

$$\frac{\partial \mathcal{L}}{\partial H_j^{(L)}} = \frac{\partial}{\partial H_j^{(L)}} \frac{1}{2} \sum_{k=1}^{n_L} (Y_k - H_k^{(L)})^2 = -(Y_j - H_j^{(L)})$$

Usando la regola della catena, otteniamo

$$\delta_j^{(L)} = \frac{\partial \mathcal{L}}{\partial z_j^{(L)}} = \frac{\partial \mathcal{L}}{\partial H_j^{(L)}} \frac{\partial H_j^{(L)}}{\partial z_j^{(L)}} = (H_j^{(L)} - Y_j)\sigma_l'\big(z_j^{(L)}\big), \ \forall j$$

da cui 1.2.

Per dimostrare 1.3 usiamo la definizione 1.1 e applichiamo la regola della catena

$$\delta_j^{(l)} = \frac{\partial \mathcal{L}}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \frac{\partial \mathcal{L}}{\partial z_k^{(l+1)}} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = \sum_{k=1}^{n_{l+1}} \delta_k^{(l+1)} \frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} \tag{1.6}$$

Ora, dalla definizione di $\mathbf{z}^{(l)}$, sappiamo che

$$z_k^{(l+1)} = \sum_{s=1}^{n_l} W_{s,k}^{(l+1)} \sigma_l\big(z_s^{(l)}\big) + b_k^{(l+1)}$$

Perciò

$$\frac{\partial z_k^{(l+1)}}{\partial z_j^{(l)}} = W_{j,k}^{(l+1)} \sigma_l'\big(z_j^{(l)}\big).$$

Sostituendo ciò in 1.6, otteniamo

$$\delta_j^{(l)} = \sum_{k=1}^{n_{l+1}} \delta_k^{(l+1)} W_{j,k}^{(l+1)} \sigma_l'\big(z_j^{(l)}\big),$$

che può essere riscritto come

$$\delta_j^{(l)} = \Big(\big(\mathbf{W}^{(l+1)}\big)^T \boldsymbol{\delta}^{(l+1)}\Big) \sigma_l'\big(z_j^{(l)}\big), \ \ \forall j$$

da cui segue 1.3

Per dimostrare 1.4, osserviamo la relazione che lega $z_j^{(l)}$ e $b_j^{(l)}$

$$z_j^{(l)} = \Big(\sigma_{l-1}\big(\mathbf{z}^{(l-1)}\big)\mathbf{W}^{(l)}\Big)_j + b_j^{(l)}.$$

Poiché $\mathbf{z}^{(l-1)}$ non dipende da $b_j^{(l)}$, vale che

$$\frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = 1.$$

Quindi, dalla regola della catena,

$$\frac{\partial \mathcal{L}}{\partial b_j^{(l)}} = \frac{\partial \mathcal{L}}{\partial z_j^{(l)}} \frac{\partial z_j^{(l)}}{\partial b_j^{(l)}} = \frac{\partial \mathcal{L}}{\partial z_j^{(l)}} := \delta_j^{(l)}$$

ovvero 1.4.

Infine, per ottenere 1.5, analizziamo la forma di un generico elemento di $\mathbf{z}^{(l)}$

$$z_j^{(l)} = \sum_{k=1}^{n_{l-1}} H_k^{(l-1)} W_{k,j}^{(l)} + b_j^{(l)}$$

da cui

$$\frac{\partial z_j^{(l)}}{\partial W_{k,j}^{(l)}} = H_k^{(l-1)}, \quad \forall j$$

e

$$\frac{\partial z_s^{(l)}}{\partial W_{k,j}^{(l)}} = 0, \quad \text{per} \quad s \neq j$$

Da queste due ultime equazioni e dalla chain rule

$$\frac{\partial \mathcal{L}}{\partial W_{k,j}^{(l)}} = \sum_{s=1}^{n_l} \frac{\partial \mathcal{L}}{\partial z_s^{(l)}} \frac{\partial z_s^{(l)}}{\partial W_{k,j}^{(l)}} = \frac{\partial \mathcal{L}}{\partial z_k^{(l)}} \frac{\partial z_k^{(l)}}{\partial W_{k,j}^{(l)}} = \frac{\partial \mathcal{L}}{\partial z_k^{(l)}} H_j^{(l-1)} = \delta_k^{(l)} H_j^{(l-1)}.$$

Questo completa la dimostrazione. □

Quindi, partendo da l=L e proseguendo a ritroso fino a l=2, possiamo aggiornare i pesi e i bias nel seguente modo

$$\mathbf{W}^{(l)} = \mathbf{W}^{(l)} - \alpha \boldsymbol{\delta}^{(l)} \mathbf{H}^{(l-1)^T}$$

$$\mathbf{b}^{(l)} = \mathbf{b}^{(l)} - \alpha \boldsymbol{\delta}^{(l)}$$

dove α prende il nome di *learning rate* o tasso di apprendimento, e corrisponde all'ampiezza del passo usata nell'algoritmo di discesa del gradiente.

In sintesi, la fasi di *Forward Propagation* consiste nell'elaborazione dei dati in input attraverso la rete al fine di ottenere l'output, invece, la *Backward Propagation* si occupa del calcolo dei gradienti della *loss function* $\mathcal{L}$ rispetto ai pesi e *bias* che caratterizzano la rete neurale. Tali gradienti sono utilizzati in un algoritmo di ottimizzazione per regolare i parametri e individuare un'approssimazione soddisfacente di un minimo locale della funzione di costo. Il susseguirsi di tali metodologie permette alla rete di apprendere autonomamente dai dati, migliorando, con l'avanzare delle epoche, le sue prestazioni durante la fase di addestramento del modello.

Capitolo 2

Introduzione alla Risonanza Magnetica Nucleare

La risonanza magnetica nucleare (NMR) è un fenomeno fisico in cui i nuclei, sottoposti all'azione di un campo magnetico, vengono perturbati da un impulso di radiofrequenza e rispondono producendo un segnale elettromagnetico con una frequenza caratteristica [16].
Questo processo coinvolge nuclei che subiscono transizioni energetiche tra livelli energetici di momento angolare. Le energie in gioco talmente basse che il segnale viene rivelato in competizione con il rumore termico [17].

Per far verificare tale fenomeno fisico è necessario che i nuclei atomici abbiano un momento magnetico, affinché questi possano interagire con un campo magnetico $\boldsymbol{B_0}$. Questa interazione porta al raggiungimento di un ordine tra i livelli energetici, favorito dall'azione del campo magnetico, e contrastato dall'agitazione termica (k_BT, con $k_B = 1.38 \cdot 10^{-23} J/K$ *costante di Boltzmann* e T è la temperatura).

Il momento magnetico nucleare $\boldsymbol{\mu}$ di un nucleo atomico che interagisce con un campo magnetico costante $\boldsymbol{B_0}$, tende ad orientarsi nella direzione del campo magnetico. Perciò, il campo magnetico esterno esercita una torsione sul momento magnetico:

$$\boldsymbol{\tau} = \boldsymbol{\mu} \times \boldsymbol{B_0}$$

Questo causa la precessione di $\boldsymbol{\mu}$ attorno alle linee di forza del campo magnetico $\boldsymbol{B_0}$, con una precisa frequenza angolare ν_0, che prende il nome di frequenza di Larmor. Ciò si traduce nella formazione di una magnetizzazione nucleare di equilibrio $\boldsymbol{M}$ che può essere rilevata al fine di ottenere informazioni sul sistema.
Per rivelare $\boldsymbol{M}$ occorre alterarla fornendo energia in modo da soddisfare la condizione di risonanza e predisporre un sistema hardware utile per seguirne l'evoluzione durante il ritorno all'equilibrio.
La quantità osservabile negli esperimenti NMR è, quindi, la magnetizzazione nucleare $\boldsymbol{M}(t)$. Questa dipende dal tempo e risulta essere, in ogni istante temporale, un vettore nello spazio tridimensionale. Ciò che si misura, in unità arbitrarie, è il segnale NMR rivelato da una bobina, come segnale elettrico indotto nella bobina dalla variazione di flusso magnetico prodotto dal moto di $\boldsymbol{M}(t)$ in un sistema di riferimento opportuno. La sua intensità ha le stesse unità di misura del momento magnetico nucleare (J/T), che è proporzionale allo spin $\boldsymbol{I}$ [1] della specie nucleare n

[1] Lo spin è una proprietà fondamentale associata alle particelle. Questa può essere positiva o negativa e si presenta in multipli di 1/2. Se un nucleo ha un momento di spin nullo, allora il fenomeno di risonanza magnetica non si verifica [15].

avente momento magnetico nucleare:

$$\boldsymbol{\mu} = \gamma_n \frac{h}{2\pi} \boldsymbol{I}$$

dove γ_n è il rapporto giromagnetico [2] della specie nucleare n e si misura in MHz/T. Quindi, una condizione necessaria affinché si verifichi il fenomeno di risonanza magnetica nucleare è che i nuclei soddisfino $\boldsymbol{I} \neq \mathbf{0}$.

Considerando un sistema con un singolo spin, l'operatore quantomeccanico corrispondente all'energia è l'operatore Hamiltoniano

$$\boldsymbol{H}_z = -\gamma \hbar \boldsymbol{I}_z B_0$$

L'interazione dei momenti magnetici con il campo magnetico, detta interazione Zeeman, provoca lo splitting dei livelli energetici, ove i livelli di energia (livelli Zeeman) sono definiti dagli autovalori dell'operatore Hamiltoniano:

$$E_m = -\gamma \hbar m B_0$$

dove $\hbar = \frac{h}{2\pi} \approx 10^{-34} J \cdot s$ è la costante di Plank ridotta (o costante di Dirac) e $m \in [-I, I]$.

I nuclei si distribuiscono su un totale di 2I+1 livelli energetici secondo la distribuzione di Boltzmann, dove i livelli con energia inferiore risultano essere quelli più popolati

$$\frac{n_{m-1}}{n_m} = exp\left(- \frac{\hbar \omega_0}{k_B T} \right),$$

dove $\omega_0 = \gamma B_0 = 2\pi\nu_0$ è il modulo della velocità angolare del momento magnetico nucleare $\boldsymbol{\mu}$ attorno alle linee di forza del campo magnetico $\mathbf{B}_0$.

La differenza tra popolazioni di livelli energetici risulta essere proporzionale al modulo del campo magnetico $\mathbf{B}_0$. Questo risultato prende il nome di Legge di Curie. Coerentemente con quanto descritto in precedenza, definiamo la magnetizzazione di equilibrio per unità di volume come un vettore avente direzione e verso di $\mathbf{B}_0$ e modulo

$$M_0 = N \frac{\gamma^2 \hbar^2 I(I+1)}{3 k_B T} B_0$$

Osserviamo con un esempio che, in generale, M_0 non è proporzionale a B_0. Supponiamo che dei nuclei abbiano spin I=1/2. Sono possibili due stati:

- stato up $m = \frac{1}{2}$,
- stato down $m = -\frac{1}{2}$,

[2] rapporto tra il momento magnetico e il momento angolare

la cui differenza di energia è $\Delta E = \gamma\hbar B_0$. Indicando con N_+ e N_- la numerosità di nuclei per unità di volume degli stati, e con $N = N_+ + N_-$ il numero totale di nuclei per unità di volume, otteniamo

$$\frac{N_+}{N_-} = exp\left(\frac{\Delta E}{k_B T}\right) = exp\left(\frac{\hbar\gamma B_0}{k_B T}\right),$$

da cui ricaviamo

$$N_+ = \frac{N}{1 + exp\left(-\frac{\hbar\gamma B_0}{k_B T}\right)}, \qquad N_- = \frac{N}{1 + exp\left(\frac{\hbar\gamma B_0}{k_B T}\right)}$$

Vale che M_0, magnetizzazione nucleare di equilibrio per unità di volume, è proporzionale alla differenza di numerosità di nuclei per unità di volume degli stati

$$M_0 = \mu(N_+ - N_-)$$

quindi

$$M_0 = N\mu \frac{1 - exp\left(-\frac{\hbar\gamma B_0}{k_B T}\right)}{1 + exp\left(-\frac{\hbar\gamma B_0}{k_B T}\right)}$$

equivalentemente,

$$M_0 = N\mu \tanh\left(\frac{\hbar\gamma B_0}{2k_B T}\right)$$

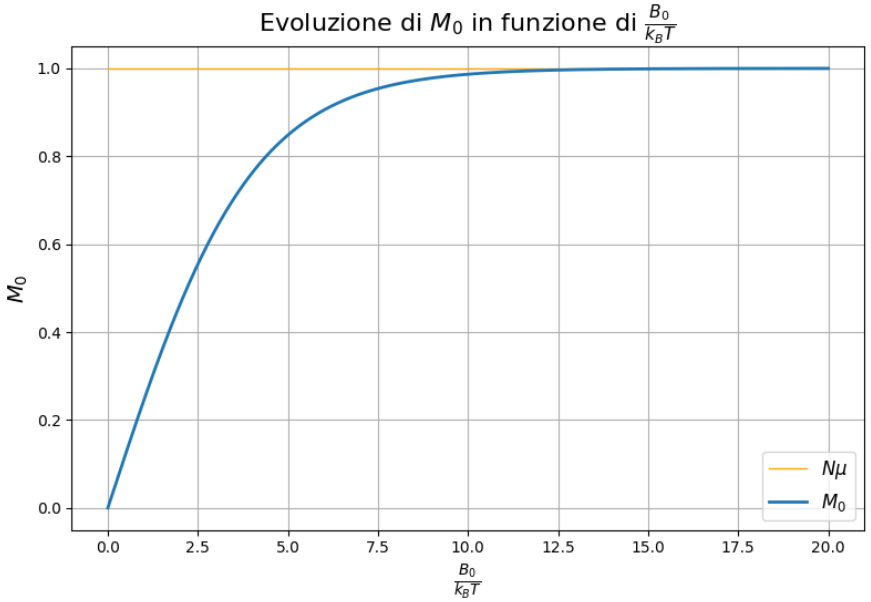


Figura 2.1: Evoluzione di M_0.

2.1 Moto di precessione della Magnetizzazione Nucleare e curva di dispersione

Si osserva sperimentalmente che è possibile descrivere il moto dell'insieme degli spin in termini dell'evoluzione della magnetizzazione nucleare nel tempo. Tale equazione del moto fa riferimento alla precessione di $\mathbf{M}$ attorno alle linee di forza del campo magnetico $\boldsymbol{B_0}$ alla velocità angolare $\omega_0 = \gamma B_0$ ed è della forma

$$\frac{d\mathbf{M}}{dt} = \gamma \mathbf{M} \times \boldsymbol{B_0}$$

Al fine di ottenere un moto di precessione non banale, è necessario che $\mathbf{M}$ non sia allineato alle linee di forza del campo magnetico $\boldsymbol{B_0}$. Per allontanare la magnetizzazione nucleare dalla direzione del campo magnetico polarizzante è necessario perturbare il sistema mediante l'applicazione di un campo magnetico perpendicolare a $\boldsymbol{B_0}$, oscillante alla frequenza di Larmor $\omega = \omega_0$. Tale campo magnetico rotante, che indichiamo con $\boldsymbol{B_1}$, varia secondo la legge

$$\boldsymbol{B_1}(t) = B_1 \cos(\omega_0 t)\boldsymbol{i} - B_1 \sin(\omega_0 t)\boldsymbol{j},$$

dove $\boldsymbol{i}$ e $\boldsymbol{j}$ sono i versori nel piano, [18].

In questo caso, il campo magnetico totale sarà della forma $\boldsymbol{B}(t) = \boldsymbol{B_0} + \boldsymbol{B_1}(t)$. L'equazione del moto di $\mathbf{M}$ cambia in

$$\frac{d\mathbf{M}}{dt} = \gamma \mathbf{M} \times \boldsymbol{B} = \begin{cases} \frac{dM_x}{dt} = \gamma(M_y B_0 + M_z B_1 \sin(\omega_0 t)) \\ \frac{dM_y}{dt} = \gamma(M_z B_1 \cos(\omega_0 t) - M_x B_0) \\ \frac{dM_z}{dt} = \gamma(-M_x B_1 \sin(\omega_0 t) - M_y B_1 \cos(\omega_0 t)) \end{cases} \tag{2.1}$$

Chiamando $\boldsymbol{k}$ il versore avente direzione e verso di $\boldsymbol{B_0}$, e imponendo, al tempo $t = 0$, la seguente condizione iniziale

$$\mathbf{M}(t)\big|_{t=0} = M_0 \boldsymbol{k}$$

otteniamo

$$M_x(t) = M_0 \sin(\omega_1 t) \sin(\omega_0 t)$$
$$M_y(t) = M_0 \sin(\omega_1 t) \cos(\omega_0 t)$$
$$M_z(t) = M_0 \cos(\omega_1 t)$$

dove $\omega_1 = \gamma B_1$.

Quindi, applicando un campo magnetico $\boldsymbol{B_1}$ ortogonale a $\boldsymbol{B_0}$ e che precede attorno alle linee di forza di questo alla stessa velocità angolare di $\boldsymbol{M}$,si può allontanare la magnetizzazione nucleare dalla direzione del campo magnetico polarizzante di un angolo che dipende dal tempo di applicazione dell'impulso a radiofrequenza

$$\alpha = \gamma B_1 t = \omega_1 t,$$

che prende il nome di angolo di nutazione (*flip angle*).

Cessata la radiofrequenza, il sistema di spin torna alla distribuzione di equilibrio di Boltzmann, quindi

$$\lim_{t\to\infty} M_z(t) = M_0$$

$$\lim_{t\to\infty} M_{xy}(t) = 0$$

I processi fisici di ritorno all'equilibrio per la componente della magnetizzazione lungo l'asse longitudinale e per quella nel piano trasversale risultano differenti, infatti il primo è di tipo energetico, mentre il secondo di tipo entropico. Al termine dell'applicazione dell'impulso a radiofrequenza, la velocità istantanea con la quale la componente longitudinale della magnetizzazione torna all'equilibrio dipende dalla velocità con cui gli spin si redristibuiscono sui due livelli energetici secondo la distribuzione di Boltzman, risultando quindi proporzionale alla differenza fra la magnetizzazione longitudinale stessa e quella di equilibrio. Vale che

$$\frac{dM_z(t)}{dt} = -\frac{(M_z(t) - M_0)}{T_1} \tag{2.2}$$

dove T_1 è il tempo di rislassamento longitudinale.

La stessa relazione di proporzionalità è valida anche per la componente trasversale, quindi, ricordando che il valore di equilibrio è nullo, otteniamo

$$\frac{dM_{xy}(t)}{dt} = -\frac{(M_{xy}(t))}{T_2}. \tag{2.3}$$

T_2 prende il nome di tempo di rilassamento trasversale, e, a differenza della controparte longitudinale, non è legato ad uno scambio di energia. Per questo motivo vale sempre che

$$T_2 \leq T_1.$$

Le equazioni 2.2 e 2.3 prendono il nome di equazioni di Bloch, ed ammettono rispettivamente soluzione

$$M_z(t) = M_z(0)e^{-\frac{t}{T_1}} + M_0\Big(1 - e^{-\frac{t}{T_1}}\Big)$$

e

$$M_{xy}(t) = M_{xy}(0)e^{-\frac{t}{T_2}}.$$

Osservandone i grafici, possiamo avere una chiara interpretazione dell'evoluzione della magnetizzazione nel tempo.

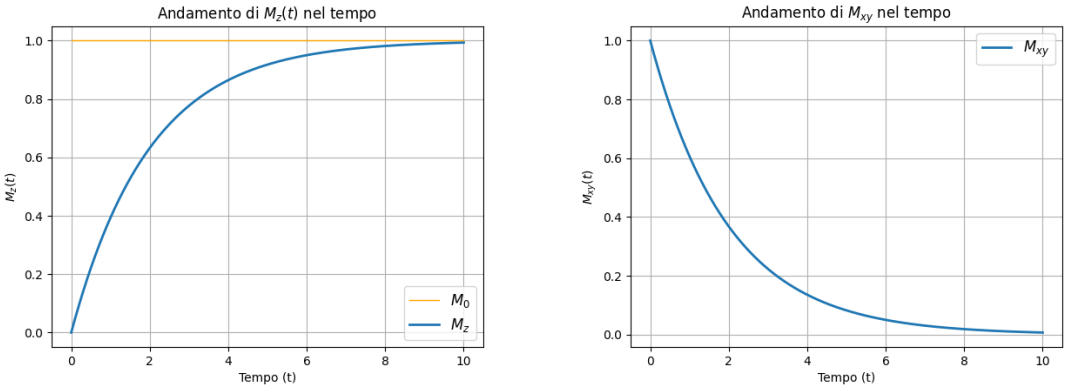


Figura 2.2: Grafici delle evoluzioni dei componenti del vettore magnetizzazione.

2.2 Risonanza magnetica nucleare a ciclo di campo rapido

La Risonanza magnetica nucleare a ciclo di campo rapido o *Fast Field Cycling Nuclear Magnetic Resonance* (FFC-NMR) è una tecnica di risonanza magnetica non distruttiva e condotta a basso campo.
A differenza delle classiche tecniche NMR a campo fisso, nella FFC-NMR l'intensità del campo magnetico $\boldsymbol{B_0}$ applicato al campione viene variata rapidamente, ciò porta all'alterazione della frequenza di risonanza del protone e alla misura del tasso di rilassamento longitudinale R_1 come una funzione della frequenza angolare di Larmor, $\omega = 2\pi f$.
La variazione rapida del campo consente di ottenere informazioni su specifiche dinamiche molecolari che altrimenti sarebbero difficili da rilevare con le tecniche NMR a campo fisso.

Questa tecnica rappresenta un importante strumento per analizzare la struttura di un mezzo poroso, e il campo applicativo è, di conseguenza, molto ampio e varia dai sistemi biologici alle rocce sedimentarie contenenti idrocarburi.

La FFC-NMR è una tecnica che valuta quanto il tasso di rilassamento longitudinale R_1 (o il relativo tempo di rilassamento ($T_1 = 1/R_1$)) di uno spin nucleare cambia al variare della forza del campo magnetico applicato.
La variazione del tasso di rilassamento longitudinale R_1, in funzione della frequenza angolare di Larmor, forma il profilo di dispersione della risonanza magnetica nucleare o *Nuclear Magnetic Resonance Dispersion* (NMRD).

Le interazioni dinamiche tra gli spin possono essere molto complesse, un esempio è il *Quadrupole Relaxation Enhancement* (QRE), che è causato dall'accoppiamento tra il dipolo magnetico e i nuclei quadrupolari di spin arbitrari $S \geq 1$. Studiando il QRE è possibile ottenere informazioni fondamentali sull'organizzazione molecolare di un dato sistema. Ciò ha un'ampia gamma di applicazioni che spaziano dalla scienza ambientale [24], allo studio di liquidi ionici, proteine [23] e alimenti [25][26]. Ad esempio, nei sistemi contenenti azoto, la presenza del QRE, che è dovuta all' interazione $^1H{-}^{14}N$, è rappresentata da massimili locali (o picchi) che caratterizzano la curva R_1. La posizione dei picchi dipende dai parametri quadrupolari che sono determinati dal tensore del gradiente di campo elettrico nella posizione dell'azoto-14. Per questo motivo, anche piccoli cambiamenti nella struttura elettronica intorno all'azoto-14 modificano la posizione e la forma dei picchi quadrupolari.

2.2.1 Il modello continuo del profilo NMRD

Secondo l'approccio *model-free* (proposto da [21] e [22], e approfondito da [20]), possiamo interpretare la variazione del tasso di rilassamento longitudinale R_1, ovvero il profilo NMRD, come somma di tre contributi:

$$R_1(\omega) = R_0 + R^{HH}(\omega) + R^{NH}(\omega)$$

Analizziamo con ordine i tre termini:

(i) R_0 è un valore non negativo che influenza la quota del profilo e tiene conto dei rapidi movimenti delle molecole.

(ii) $R^{HH}(\omega)$ dipende dalla funzione di distribuzione $f(\tau)$ nel seguente modo:

$$R^{HH}(\omega) = \int_0^\infty \left(\frac{\tau}{(1 + (\omega\tau)^2)} + \frac{4\tau}{(1 + 4(\omega\tau)^2)} \right) f(\tau) \tag{2.4}$$

dove τ è il tempo di correlazione, cioè il tempo medio richiesto da una molecola per ruotare di un radiante o, equivalentemente, di muoversi per una distanza tanto grande quanto il suo raggio d'inerzia.

(iii) $R^{NH}(\omega)$ descrive l'insorgenza dei picchi quadrupolari

$$R^{NH}(\omega) = C^{NH} \left(\frac{1}{3} + \sin^2(\Theta)\cos^2(\Phi), \frac{1}{3} + \sin^2(\Theta)\sin^2(\Phi), \frac{1}{3} + \cos^2(\Theta) \right) \cdot$$

$$\cdot \begin{pmatrix} \frac{\tau_Q}{1+(\omega-\omega_-)^2\tau_Q^2} + \frac{\tau_Q}{1+(\omega+\omega_-)^2\tau_Q^2} \\ \frac{\tau_Q}{1+(\omega-\omega_+)^2\tau_Q^2} + \frac{\tau_Q}{1+(\omega+\omega_+)^2\tau_Q^2} \\ \frac{\tau_Q}{1+(\omega-\omega_++\omega_-)^2\tau_Q^2} + \frac{\tau_Q}{1+(\omega+\omega_+-\omega_-)^2\tau_Q^2} \end{pmatrix} \tag{2.5}$$

dove

- C^{NH} si riferisce al rapporto giromagnetico e la distanza media di interazione tra i nuclei;
- Θ e Φ sono due angoli che fanno riferimento all'orientazione degli assi dipolo-dipolo $^1H -^{14} N$ rispetto al sistema di assi principali del gradiente di campo elettrico alla posizione di ^{14}N;
- τ_Q è il tempo di correlazione per l'interazione quadrupolare di $^1H -^{14} N$;
- ω_+ e ω_- sono le posizioni delle velocità angolari caratteristiche alle quali si verificano i picchi nel profilo NMRD.

2.2.2 Il modello discreto del profilo NMRD

Sia $\boldsymbol{\omega} \in \mathbb{R}^m$ il vettore degli m valori delle frequenze angolari di Larmor ($\omega = 2\pi\nu$, dove ν si misura in Mhz) in cui R_1 è calcolato, e sia $\boldsymbol{y}$ il corrispondente vettore delle osservazioni, ovvero $y_i = R_1(\omega_i)$ i = 1, . . ., m. Sia $\boldsymbol{f} \in \mathbb{R}^n$ il vettore ottenuto campionando il dominio di $f(\tau)$ in $\tau_1, \ldots, \tau_n$, ovvero n valori equispaziati in modo logaritmico. Infine, sia $\boldsymbol{\Psi} \in \mathbb{R}^6$ tale per cui $\Psi_1 = C^{NH}$, $\Psi_2 = \sin^2(\Theta)$, $\Psi_3 = \sin^2(\Phi)$, $\Psi_4 = \tau_Q$, $\Psi_5 = \omega_-$, $\Psi_6 = \omega_+$.
Discretizzando le equazioni 2.4 e 2.5, otteniamo il modello

$$\boldsymbol{y} = \mathcal{F}(\boldsymbol{f}, \boldsymbol{\Psi}, R_0) := \mathcal{F}_1(\boldsymbol{f}) + \mathcal{F}_2(\boldsymbol{\Psi}) + R_0, \tag{2.6}$$

dove $\mathcal{F} : \mathbb{R}^{n+6+1} \to \mathbb{R}^m$ è scomposto in tre termini. Il primo di questi dipende solo da $\boldsymbol{f}$, cioé è della forma

$$\mathcal{F}_1 : \mathbb{R}^n \to \mathbb{R}^m,$$

ed è inoltre una funzione lineare di $\boldsymbol{f}$ derivante dalla discretizzazione dell'integrale 2.4, quindi può essere scritto come

$$\mathcal{F}_1(\boldsymbol{f}) = \boldsymbol{K}\boldsymbol{f},$$

dove $\boldsymbol{K} \in \mathbb{R}^{m \times n}$ e $\boldsymbol{f} \in \mathbb{R}^n$. Costruiamo $\boldsymbol{K}$ nel seguente modo:

$$K_{i,j} = \frac{\tau_j}{(1 + (\omega_i \tau_j)2)} + \frac{4\tau_j}{(1 + 4(\omega_i \tau_j)2)},$$

solitamente, negli esperimenti di FFC-NMR vale che $m \ll n$.

Il secondo termine di 2.6 è

$$\mathcal{F}_2 : \mathbb{R}^6 \to \mathbb{R}^m,$$

dipende solo dai parametri Ψ_j, dove $j = 1, 2, \dots 6$, e rappresenta la componente quadrupolare R^{NH} 2.5

$$\big(\mathcal{F}_2(\boldsymbol{\Psi})\big)_i = \Psi_1 \left(\frac{1}{3} + \Psi_2(1 - \Psi_3), \frac{1}{3} + \Psi_2\Psi_3, \frac{1}{3} + (1 - \Psi_2) \right) \cdot$$

$$\cdot \begin{pmatrix} \frac{\Psi_4}{1+(\omega_i-\Psi_5)^2\Psi_4^2} + \frac{\Psi_4}{1+(\omega_i+\Psi_5)^2\Psi_4^2} \\ \frac{\Psi_4}{1+(\omega_i-\Psi_6)^2\Psi_4^2} + \frac{\Psi_4}{1+(\omega_i+\Psi_6)^2\Psi_4^2} \\ \frac{\Psi_4}{1+(\omega_i-\Psi_6+\Psi_5)^2\Psi_4^2} + \frac{\Psi_4}{1+(\omega_i+\Psi_6-\Psi_5)^2\Psi_4^2} \end{pmatrix}$$

per $i = 1, 2, \dots, m$.

Infine, l'ultimo termine di 2.6 è il parametro costante $R_0 \geq 0$ che rappresenta la variazione di quota nella curva NMRD.

Siamo interessati ad individuare i parametri $\boldsymbol{f}$, $\boldsymbol{\Psi}$ e R_0 che caratterizzano il vettore $\boldsymbol{y}$. Tale problema inverso è non lineare e mal condizionato.
L'approccio per stabilizzare la procedura di identificazione dei parametri, suggerito da [19], è un metodo di regolarizzazione. Nello specifico, è usata una regolarizzazione L_1 al fine di indurre la sparsità della $\boldsymbol{f}$, infatti, è noto a priori che la distribuzione $f(\tau)$ è una funzione sparsa con solo pochi termini non nulli.

Una volta fissato il box di appartenenza dei parametri $\boldsymbol{\Psi}$

$$\mathcal{B}_\Psi = \Big\{ \boldsymbol{\Psi} : \Psi_1 \in \big[0, \bar{C}\big]; \Psi_2, \Psi_3 \in \big[0, 1\big]; \Psi_4 \in \big[0, \bar{\tau}\big]; \Psi_5, \Psi_6 \in \big[\omega_l, \omega_u\big] \Big\},$$

il problema di identificazione dei parametri è equivalente al seguente problema di ottimizzazione

$$\min_{\boldsymbol{f}, \boldsymbol{\Psi}, R_0} \Big(\|\boldsymbol{y} - (\mathcal{F}_1(\boldsymbol{f}) + \mathcal{F}_2(\boldsymbol{\Psi}) + R_0)\|_2^2 + \lambda \|\boldsymbol{f}\|_1 \Big), \quad t.c. \quad \boldsymbol{f} \geq \boldsymbol{0}, \boldsymbol{\Psi} \in \mathcal{B}_\Psi, R_0 \geq 0. \quad (2.7)$$

Dal parametro di regolarizzazione λ dipende il peso del termine di regolarizzazione L_1, per questo motivo la scelta di λ è fondamentale per identificare i parametri $(\boldsymbol{f}, \Psi, R_0)$ ottenuti risolvendo 2.7.

Capitolo 3

Metodologie risolutive

In questo capitolo confrontiamo il metodo di risoluzione del problema mediante ottimizzazione con l'approccio innovativo che consiste nella previsione dei parametri che caratterizzano una curva di dispersione NMR mediante l'uso di reti neurali.

3.1 Stima dei Parametri mediante Ottimizzazione

Supponiamo fissato il valore del parametro λ, descriviamo il metodo di Gauss Seidel a due blocchi usato per ottenere la soluzione del problema 2.7.

Partizioniamo le incognite di 2.7 in due blocchi, al fine di suddividere tali parametri in base al fatto che la funzione da minimizzare dipenda da questi in maniera lineare oppure no. Otteniamo

$$\boldsymbol{x}_1 \equiv (\boldsymbol{f}, R_0),\ \boldsymbol{x}_2 \equiv \boldsymbol{\Psi}.$$

Fissiamo $\boldsymbol{K}_e = \begin{bmatrix}\boldsymbol{K} & \mathbf{1}\end{bmatrix} \in \mathbb{R}^{m\times(n+1)}$, e riformuliamo il problema nel seguente modo

$$\min_{\boldsymbol{x}_1,\boldsymbol{x}_2} g(\boldsymbol{x}_1,\boldsymbol{x}_2) = \min_{\boldsymbol{x}_1,\boldsymbol{x}_2} \Big(\|\boldsymbol{y} - \boldsymbol{K}_e\boldsymbol{x}_1 - \mathcal{F}_2(\boldsymbol{x}_2)\|_2^2 + \lambda\|\boldsymbol{x}_1\|_1 + \eta\|\boldsymbol{x}_1\|_2^2\Big), \tag{3.1}$$

$$t.c.\quad \boldsymbol{x}_1 \in \{\boldsymbol{x}_1 \geq \mathbf{0}\}, \boldsymbol{x}_2 \in \mathcal{B}_\Psi$$

Osserviamo come $\eta\|\boldsymbol{x}_1\|_2^2$, termine di regolarizzazione L_2 dove η è molto piccolo (ad esempio $\eta = 1e-10$), è stato aggiunto alla funzione oggetto per assicurarsi che $\boldsymbol{K}_e^T\boldsymbol{K}_e + \eta\boldsymbol{I}$ sia una matrice definita positiva.

I sottoinsiemi chiusi $\{\boldsymbol{x}_1 \geq \mathbf{0}\} \subseteq \mathbb{R}^{n+1}$ e $\mathcal{B}_\Psi \subseteq \mathbb{R}^6$ sono entrambi convessi, inoltre, la funzione oggetto $g(\boldsymbol{x}_1,\boldsymbol{x}_2)$ è continua e convessa rispetto a $\boldsymbol{x}_1$ per $\boldsymbol{x}_2$ fissato, ma è non convessa rispetto a $\boldsymbol{x}_2$ per $\boldsymbol{x}_1$ fissato. Nonostante ciò, poiché $\boldsymbol{K}_e^T\boldsymbol{K}_e + \eta\boldsymbol{I}$ è definita positiva e $\mathcal{B}_\Psi$ è limitato, è possibile dimostrare che g è coercitiva su $\{\boldsymbol{x}_1 \geq \mathbf{0}\} \times \mathcal{B}_\Psi$. Dunque, ne ricordiamo la definizione

Definizione 3.1.1. *Una funzione $g : \mathbb{R}^q \to \mathbb{R}$ è detta coercitiva* coercitiva *in X se, per ogni successione $\{\boldsymbol{x}^{(k)}\} \in X$ tale che $\|\boldsymbol{x}^{(k)}\| \to \infty$, abbiamo*

$$\lim_{k\to\infty} g\big(\boldsymbol{x}^{(k)}\big) = +\infty.$$

Continuità e coercività assicurano l'esistenza di almeno un minimizzatore globale di $g(\boldsymbol{x}_1,\boldsymbol{x}_2)$ in $\{\boldsymbol{x}_1 \geq \mathbf{0}\} \times \mathcal{B}_\Psi$.

Nel metodo di Gauss-Seidel vincolato a due blocchi, ad ogni iterazione, la funzione oggetto è minimizzata rispetto entrambe le coordinate che, singolarmente, caratterizzano i blocchi. Mostriamone lo pseudocodice

Algorithm 3 Metodo di Gauss-Seidel non lineare vincolato a due blocchi

function GS
  Input: $\boldsymbol{x}_1^{(0)}, \boldsymbol{x}_2^{(0)}, Tol_{GS}$
  $k = 0$
  repeat
    $k = k + 1$
    $\boldsymbol{x}_1^{(k)} \in \underset{\boldsymbol{z} \geq \boldsymbol{0}}{\operatorname{argmin}}\, g\left(\boldsymbol{z}, \boldsymbol{x}_2^{(k-1)}\right)$
    $\boldsymbol{x}_2^{(k)} \in \underset{\boldsymbol{z} \in \mathcal{B}_\Psi}{\operatorname{argmin}}\, g\left(\boldsymbol{x}_1^{(k)}, \boldsymbol{z}\right)$
  until $\left|g\left(\boldsymbol{x}_1^{(k)}, \boldsymbol{x}_2^{(k)}\right) - g\left(\boldsymbol{x}_1^{(k-1)}, \boldsymbol{x}_2^{(k-1)}\right)\right| \leq Tol_{GS}\left|g\left(\boldsymbol{x}_1^{(k)}, \boldsymbol{x}_2^{(k)}\right)\right|$
  return $\left(\boldsymbol{x}_1^{(k)}, \boldsymbol{x}_2^{(k)}\right)$
end function

Finora supponevamo fissato il valore del parametro λ, la cui scelta è fondamentale al fine di analizzare correttamente il profilo NMRD.

In [27] è proposto un principio di bilanciamento dipendente da un fattore moltiplicativo γ e avente la seguente forma

$$\gamma\lambda\|\boldsymbol{x}_1\|_1 = \|\boldsymbol{y} - \boldsymbol{K}_e\boldsymbol{x}_1 - \mathcal{F}_2(\boldsymbol{x}_2)\|_2^2 + \eta\|\boldsymbol{x}_1\|_2^2.$$

Fissando $\gamma = 1$ otteniamo

$$\lambda = \frac{\|\boldsymbol{y} - \boldsymbol{K}_e\boldsymbol{x}_1 - \mathcal{F}_2(\boldsymbol{x}_2)\|_2^2 + \eta\|\boldsymbol{x}_1\|_2^2}{\|\boldsymbol{x}_1\|_1}.$$

Quindi l'obiettivo è quello di identificare i parametri NMRD e quello di regolarizzazione rispettivamente risolvendo 2.7 con il metodo di Gauss Seidel e usando il principio di bilanciamento ad ogni aggionamento di λ, fino al raggiungimento della seguente condizione di convergenza

$$\left|\lambda^{(k+1)} - \lambda^k\right| \leq Tol_\lambda\left|\lambda^{(k)}\right|,$$

dove $Tol_\lambda > 0$ è la tolleranza fissata.

Finalmente, possiamo presentare lo pseudocodice del metodo iterativo che permette l'identificazione sia di $(\boldsymbol{f}, \boldsymbol{\Psi}, R_0)$, parametri della curva di dispersione NMR, sia del parametro di regolarizzazione λ

Algorithm 4 Metodo iterativo per identificare $(\boldsymbol{f}, \boldsymbol{\Psi}, R_0)$ e λ

function AURORA
  Input: $\boldsymbol{x}_1^{(0)}, \boldsymbol{x}_2^{(0)}, Tol_\lambda$
  $k = 0$
  $\eta = 1e - 10$
  $\lambda^{(0)}$ *scelto opportunamente*
  repeat
    $k = k + 1$
    $(\boldsymbol{x}_1^{(k)}, \boldsymbol{x}_2^{(k)}) \in \underset{\boldsymbol{x}_1 \geq \boldsymbol{0}, \boldsymbol{x}_2 \in \mathcal{B}_\Psi}{\operatorname{argmin}} \Big(\|\boldsymbol{y} - \boldsymbol{K}_e \boldsymbol{x}_1 - \mathcal{F}_2(\boldsymbol{x}_2)\|_2^2 + \eta \|\boldsymbol{x}_1\|_2^2 + \lambda^{(k)} \|\boldsymbol{x}_1\|_1 \Big)$
    $\lambda^{(k+1)} = \dfrac{\left\|\boldsymbol{y} - \boldsymbol{K}_e \boldsymbol{x}_1^{(k)} - \mathcal{F}_2\left(\boldsymbol{x}_2^{(k)}\right)\right\|_2^2 + \eta \left\|\boldsymbol{x}_1^{(k)}\right\|_2^2}{\left\|\boldsymbol{x}_1^{(k)}\right\|_1}$
  until $\left|\lambda^{(k+1)} - \lambda^k\right| \leq Tol_\lambda \left|\lambda^{(k)}\right|$
  return $(\boldsymbol{f}, R_0) = \boldsymbol{x}_1^{(k)}, \boldsymbol{\Psi} = \boldsymbol{x}_2^{(k)}$
end function

Questo metodo prende il nome di AURORA (AUtomatic L1-Regularized mOdel fRee Analysis) e può essere interpretato come un'iterazione di punto fisso per il problema

$$(\boldsymbol{x}_1^*, \boldsymbol{x}_2^*) \in \underset{\boldsymbol{x}_1 \geq \boldsymbol{0}, \boldsymbol{x}_2 \in \mathcal{B}_\Psi}{\operatorname{argmin}} \Big(\|\boldsymbol{y} - \boldsymbol{K}_e \boldsymbol{x}_1 - \mathcal{F}_2(\boldsymbol{x}_2)\|_2^2 + \eta \|\boldsymbol{x}_1\|_2^2 + \lambda^* \|\boldsymbol{x}_1\|_1 \Big),$$

$$\lambda^* = \frac{\left\|\boldsymbol{y} - \boldsymbol{K}_e \boldsymbol{x}_1^* - \mathcal{F}_2\left(\boldsymbol{x}_2^*\right)\right\|_2^2 + \eta \left\|\boldsymbol{x}_1^*\right\|_2^2}{\left\|\boldsymbol{x}_1^{(k)}\right\|_1}.$$

3.2 Soluzione mediante Reti Neurali

Presentiamo ora i risultati numerici ottenuti grazie all'uso di reti neurali capaci di prevedere in maniera soddisfacente i parametri che caratterizzano una curva di dispersione NMR, dove supponiamo di conoscere a priori il tipo di materiale in analisi.

Il linguaggio di programmazione scelto è Python e si è fatto ampio uso delle librerie:

- `torch` [28]
  è il modulo principale di `PyTorch` (framework per il Deep Learning), e fornisce le funzionalità basilari per il calcolo dei tensori e del gradiente. Contiene sottomoduli fondamentali come `torch.nn` e `torch.optim`. Il primo offre gli strumenti per definire e addestrare le reti neurali, il secondo contiene gli ottimizzatori come Adam e SGD .

- `numpy` [29]
  è una libreria fondamentali per il calcolo scientifico, e offre una vasta gamma di funzionalità per lavorare con array multidimensionali e con operazioni matematiche eseguite su questi.

- `pandas` [30]
  è la libreria che permette di caricare i dati da file Excel. Si può fare ciò usando il comando `pd.read_excel()`.

- `matplotlib` [31]
  contiene il modulo `matplotlib.pyplot`, che permette la creazione di grafici e la visualizzazione dei dati.

- `random` [32]
  implementa generatori di numeri pseudo-randomici a partire da differenti distribuzioni come, ad esempio, quella uniforme. Partendo da ciò si possono creare dati sintetici per l'addestramento della rete.

Il cuore del lavoro di tesi svolto consiste di due script Python

- `AddestraModelli`

- `TestaModelli`

Il primo, leggendo opportuni dati di laboratorio da un file Excel, fissa le caratteristiche del problema, crea una rete neurale e la addestra usando esempi sintetici. Al termine dell'addestramento i pesi del modello sono salvati su un file avente estensione `.pth`, che può essere caricato nel secondo script al fine di valutare le performance della rete.

Ci apprestiamo ad analizzare i punti salienti dell'algoritmo.

Dataset Excel

I dataset utilizzati negli esperimenti condotti sono quattro e scandiscono le caratteristiche della curva di dispersione ottenuta a partire da uno specifico materiale analizzato.

I campioni usati in laboratorio sono i seguenti

- Nanospugna idratata
- Nanospugna secca
- Parmigiano
- Formaggio

Per ognuno di questi è noto un campionamento della curva di dispersione R_1, dove ogni coppia di punti è ottenuta eseguendo un singolo esperimento FFC-NMR. Inoltre, sono noti i parametri $\boldsymbol{f}^*$, $\boldsymbol{\Psi}^*$ e R_0^* ottenuti usando l'algoritmo AURORA.

Vogliamo addestrare la rete supponendo di conoscere a priori il tipo di campione, così da considerare il vettore $\boldsymbol{f}^*$ come noto. Quindi, l'obiettivo è quello di prevedere $\boldsymbol{\Psi}^*$ e R_0^* partendo da una generica curva NMRD.

In AddestraModelli, estraiamo dal file con estensione `.xlsx` i dati necessari usando il comando `pd.read_excel` fornito dalla libreria pandas, che è una delle librerie più popolari in Python per la manipolazione e l'analisi dei dati. Dai parametri $\boldsymbol{\Psi}^*$ e R_0^* generiamo *low_b* e *up_b*, liste contenenti rispettivamente i minimi e i massimi degli intervalli del box di appartenenza in cui vogliamo vincolare il valore delle previsioni.

```
data = pd.read_excel('DatiNanoSpugna_secca.xlsx', header=None)

par = data.iloc[:, 0].dropna().to_numpy()
RHH_f = data.iloc[:, 1].dropna().to_numpy()
frequency_points = data.iloc[:, 2].dropna().to_numpy()

d = (par[6] - par[5])/2
low_b = [0,   0.7 * par[1], 0,     0,   0,        2 * np.pi * (par[5]
  - d), 2 * np.pi * (par[5] - d)]
up_b =  [100, 1.3 * par[1], 1,     1,   2*par[4], 2 * np.pi * (par[6]
  + d), 2 * np.pi * (par[6] + d)]
```

Osserviamo come, il box di appartenenza dei parametri sia dato dal prodotto cartesiano di sette intervalli reali non negativi, di cui il primo, il terzo e il quarto sono di default $[0, 100]$, $[0, 1]$ e $[0, 1]$.

Al contrario gli altri dipendono dai parametri estratti dal file Excel, nello specifico,

$$\tau_Q \in \left[0.7\tau_Q^*, 1.3\tau_Q^*\right]$$

e

$$\omega_-, \omega_+ \in \left[2\pi\left(\nu_-^* - \frac{\nu_+^* - \nu_-^*}{2}\right), 2\pi\left(\nu_+^* + \frac{\nu_+^* - \nu_-^*}{2}\right)\right].$$

Passiamo alla costruzione del modello

create_model

Vogliamo costruire una rete neurale che, una volta allenata, sia capace di associare i valori dei parametri ad ogni curva NMRD di uno specifico materiale.

La partizione del dominio della curva R_1, ottenuta in laboratorio, dipende dall'esperimento. Nello script, salviamo con il nome `frequency_points` tale vettore, che è ordinato decrescentemente ed è di lunghezza fino a 48. La lunghezza del vettore non è fissa poiché dipende dal numero di punti sperimentali campionati.

Dal momento che l'immagine di `frequency_points` per mezzo della funzione R_1 è un vettore avente stessa lunghezza (ovvero `len(frequency_points)`), l'obiettivo finale è quello di generare una rete neurale che possa prendere in input un vettore lungo `len(frequency_points)` al fine di fornire in output un vettore opportuno lungo 7.

Allora definiamo la funzione `create_model`, che prende in input

- il numero di neuroni nello strato iniziale e negli strati interni
- le funzioni di attivazione associate
- le liste `low_b` e `up_b` che definiscono in maniera univoca il box di appartenenza dei parametri da prevedere.

Tale funzione contiene due classi

- `BoxActivation`
- `CustomModel`

La prima, prendendo in input le liste `low_b` e `up_b`, permette di creare l'oggetto `triangle_wave`, ovvero una funzione di attivazione da applicare al *layer* in output della rete neurale, affinché i parametri predetti appartengano sicuramente al box.

Nello specifico, per $i = 0, \dots, 6$, il parametro i-esimo viene costretto al suo intervallo di appartenenza $\big[low\_b[i],\ up\_b[i]\big]$. Facciamo ciò per mezzo del valore assoluto, traslato a quota $low\_b[i]$, di un'onda triangolare avente ampiezza $up\_b[i] - low\_b[i]$ e periodo $4\big(up\_b[i] - low\_b[i]\big)$

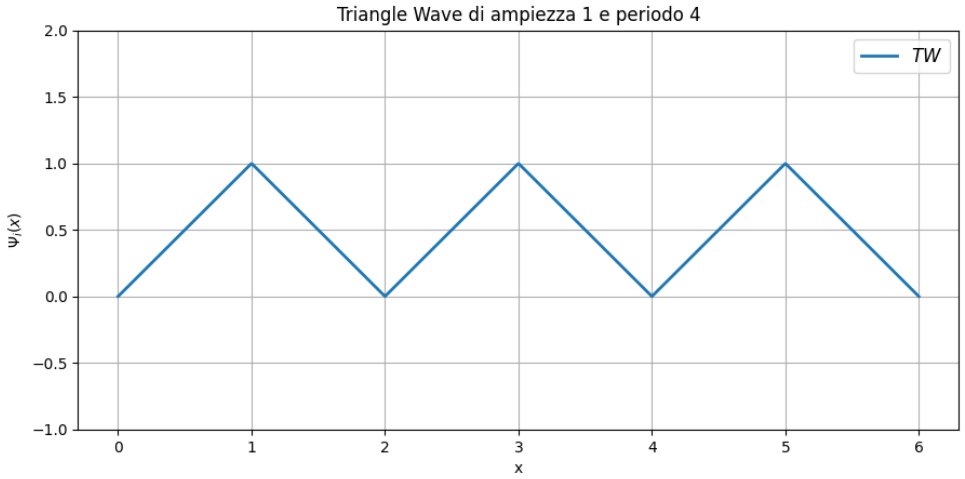


Figura 3.1: Onda triangolare di ampiezza 1 e periodo 4.

La classe `CustomModel`, invece, prende in input il numero di neuroni che vede essere presente in ciascun *layer* e dà in output il modello. Per fare ciò, crea una lista vuota che dovrà contenere tutti gli strati della rete, e appende ciclicamente lo strato lineare corrente seguito dalla funzione di attivazione corrispondente.

Quindi, all'interno di `create_model`, dopo aver definito le due classi, si fissa l'`output_size` a 7, si crea l'oggetto

```
model = CustomModel(input_size,num_neurons_hidden_layer,output_size)
```

e lo si ritorna.

Un possibile uso di `create_model` è il seguente

```
input_size = len(frequency_points)
neurons_per_hidden_layer = [256, 256, 256, 256, 64]
activation_function = nn.ReLU

model = create_model(input_size, neurons_per_hidden_layer,
 activation_function, low_b, up_b)
```

Il modello in questione è una rete neurale con 7 nodi nello strato in input, 5 strati interni di cui l'ultimo presenta 64 nodi e gli altri 256 e, infine, 7 nodi nello strato in output. A quest'ultimo, la funzione di attivazione applicata è un'onda triangolare,

laddove, per quanto concerne gli strati interni e quello in input, abbiamo una classica *Rectified Linear Unit*.

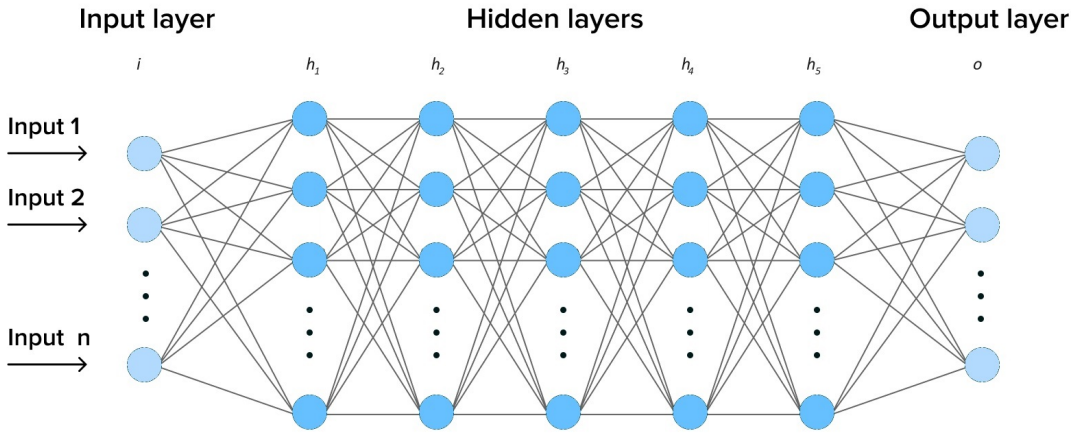


Figura 3.2: MLP con fino a 48 nodi nel *layer* in input, 256 nodi nei primi 4 nascosti, 64 nell'ultimo nascosto e 7 nell'output.

Una volta creato il modello, lo si addestra per mezzo della funzione `train`, che sfrutta, a sua volta, diverse funzioni ausiliarie. Analizziamo con ordine le principali.

genera_parametro

La funzione `genera_parametro` permette di creare una lista contente n esempi di uno stesso parametro. Gli elementi in questione sono scelti a partire da un intervallo distribuito uniformemente di estremi dati in input.
Ad esempio

```
n = 10000
R0_random = genera_parametro(n, low_b[0], up_b[0])
```

`R0_random` è una lista composta da 10000 possibili valori di R_0.
Dopo aver generato le sette liste (una per ogni parametro) aventi stessa lunghezza, le forniamo in input alla funzione `train` dove, tramite `generate_train`, mi permetteranno di ottenere gli esempi, a partire dai quali, potrò addestrare la rete.

generate_train

Tale funzione prende in input le liste associate ai parametri, il vettore $\boldsymbol{f}$ e la partizione del dominio. Oltre al dominio, dà in output un tensore bidimensionale grazie alla funzione `torch.tensor`, presente nel framework di machine learning PyTorch.

```
_, X_train = generate_train(R0, CHN, Theta, Phi, tauQ, omegaMinus,
omegaPlus, RHH_f, W)
```

`X_train`, le cui righe sono esempi di curve di dispersione, può essere inserito all'interno della rete neurale. A quel punto i dati, durante la fase di propagazione in avanti, vengono trasformati non linearmente in un tensore, le cui righe rappresentano la previsione dei parametri dovuta al modello corrente.

```
y_pred = model(X_train)
```

Calcolando la funzione di perdita tra `y_pred` e il tensore contenente i parametri esatti, possiamo aggiustare i pesi del modello durante la retropropagazione.

R1Loss

Dalla scelta della funzione di *loss* dipende la performance del modello addestrato. Per questo motivo sono state calcolate due funzioni costo differenti, al fine di eseguire l'addestramento usando una combinazione convessa di queste. Nello specifico, abbiamo

```
# Funzioni di loss sui 7 parametri
loss_function_psi = nn.L1Loss() #MAE

# Funzione di loss su R1
loss_function_R1 = R1Loss() #MAE
```

`nn.L1Loss()`, fornita da PyTorch, calcola la media degli errori assoluti (MAE) tra due tensori. La usiamo quindi per calcolare la *loss* tra `y_pred` e il tensore contenente i parametri esatti.

Al contrario, `R1Loss()` è una funzione costruita ad hoc che, partendo dai parametri predetti ed esatti, calcola i tensori bidimensionali aventi come righe le curve di dispersione corrispondenti. Su tali tensori calcoliamo la media degli errori assoluti (MAE).

Usando solamente la prima, la rete impara a prevedere i parametri non tenendo conto del grafico della curva di dispersione corrispondente. Al contrario, la seconda cerca di approssimare i parametri prendendo solamente in considerazione la forma della curva. Dal momento che il problema è mal condizionato, i due approcci potrebbero non essere completamente soddisfacenti, per questo motivo è ragionevole calcolare una combinazione convessa di questi, con l'obiettivo di trovare il compromesso migliore.

Fissiamo quindi il valore di un parametro `alpha` $\in [0, 1]$ e calcoliamo l'`elastic net`, combinazione convessa tra le *loss*

```
# Calcoliamo la loss elastic net
loss_R1 = loss_function_R1(y_pred, y_train, RHH_f, W)
loss_psi = loss_function_psi(y_pred, y_train)
loss = alpha * loss_R1 + (1 - alpha) * loss_psi  # elastic net
```

A questo punto, variando il valore di `alpha`, possiamo decidere quale funzione di perdita utilizzare.

Osserviamo che, se `alpha=0`, allora l'`elastic net` coincide con la `loss_psi`, ovvero la *loss function* riferita ai sette parametri. Al contrario, se `alpha=1`, allora la funzione di perdita considerata è equivalente alla `loss_R1`, che tiene unicamente conto della forma del profilo di dispersione.

train

Finalmente possiamo presentare la funzione che si occupa dell'addestramento della rete.
Fornendo in input

- il modello da addestrare
- le liste contenenti i minimi e i massimi degli intervalli del box di appartenenza in cui vogliamo vincolare il valore delle previsioni
- le sette liste dei parametri
- il vettore $\boldsymbol{f}^*$
- la partizione del dominio
- il *learning rate* che vogliamo usare nell'algoritmo di ottimizzazione al fine di minimizzare la loss
- il numero di epoche per il quale vogliamo continuare ad eseguire l'addestramento

la funzione `train` restituisce il modello addestrato.
Per fare ciò, per prima cosa fissa l'ottimizzatore Adam e il tasso di apprendimento dinamico usando la *cosine anneling schedule*, come mostrato nella sezione 1.1.4.
Poi calcola `X_train` e `X_test`, ovvero dei tensori bidimensionali aventi come righe degli esempi di curve di dispersione. Tali tensori sono fondamentali ai fini del calcolo della *loss*, rispettivamente nella fase di addestramento e di test del modello.
Dopo aver fissato `alpha`, esegue un ciclo nel numero massimo di epoche stabilito. Durante ogni iterazione, trova i parametri predetti con la fase di propagazione in avanti, calcola la *loss* `elastic net`, aggiorna i pesi e il tasso di apprendimento durante la fase di retropropagazione e, infine, calcola la *loss* relativa ai dati di test. L'ultimo passaggio assicura che non ci sia *overfit*, ovvero vogliamo evitare lo scenario

in cui la rete interpola perfettamente gli esempi su cui viene addestrata ma fallisce nel tentativo di prevedere i parametri di nuove curve.

Inoltre, durante l'avanzare delle epoche, `train` memorizza la *loss* corrente più piccola, e, ad ogni aggiornamento di questa, salva i pesi del modello. Questo passaggio ci permette di testare agevolemte la performance di una rete neurale, semplicemente caricando i pesi salvati. Al contrario sarebbe poco pratico eseguire ogni volta l'addestramento poiché necessita di tempi molto lunghi.

Quindi, ricapitoliamo il funzionamento dello script `AddestraModelli`. Inizialmente selezioniamo il campione di laboratorio e fissiamo i dati del problema. Poi geriamo una rete neurale che dovrà essere capace di associare ad ogni curva un'approssimazione dei parametri che la caratterizzano. A quel punto, al fine di addestrare la rete, generiamo randomicamente un grande numero (e.g. 10000) di esempi di curve e validiamo tale risultato testando la rete su un minor numero (e.g. 200) di esempi inediti. Al termine di tale procedimento possiamo testare agevolmente le capacità predittive della rete su `TestaModelli`, caricando i pesi salvati su un nuovo modello.

Nel secondo script in Python, generiamo una rete di dimensioni conformi a quella usata durante l'addestramento scelto.

```
model = create_model(input_size, neurons_per_hidden_layer,
  activation_function, low_b, up_b)
```

Carichiamo i pesi del modello precedentemente addestrato usando, come segue, il comando `torch.load()`

```
checkpoint = torch.load("best_model_MAE_NanoSpugna_idrata_zero_cinque
  .pth")
model.load_state_dict(checkpoint)
```

Infine ci apprestiamo a valutare la bontà del risultato.
Quindi, stampiamo a schermo i parametri reali e predetti, eseguiamo il plot dei grafici delle rispettive curve e calcoliamo gli errori relativi commessi sulla ricostruzione dei parametri e la loro media.
In particolare, l'errore relativo commesso su R_0 è dato da

```
err_R0 = np.abs((R0_true[0]-R0_pred)/R0_true[0])
```

Viene anche stampato a schermo quale parametro viene ricostruito meglio e quale peggio, con errori annessi.

3.2.1 Esempio di applicazione

Fissiamo i seguenti dati del problema

```
data = pd.read_excel('DatiNanoSpugna_idrata.xlsx', header=None)
neurons_per_hidden_layer = [256, 256, 256, 256, 64]
alpha = 0.5
```

ovvero, stiamo prendendo in considerazione come campione una nanospugna di ciclodestrina idratata, e generiamo come modello una rete neurale avente cinque *layer* nascosti di cui l'ultimo contiene 64 nodi, mentre gli altri 256.
Addestraimo il modello su 10000 epoche, usando come funzione di perdita la combinazione convessa di parametro `alpha=0.5` tra la media degli errori assoluti commessi sulla riproduzione dei parametri $\mathbf{\Psi}^*$ e quella riferita alla ricostruzione della funzione R_1.

Otteniamo la seguente evoluzione della `loss` e della `test loll`

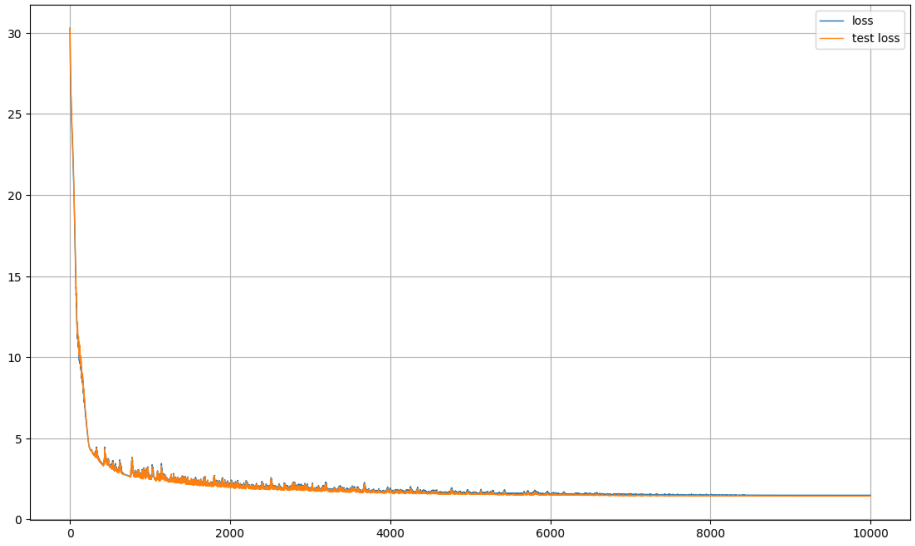


Figura 3.3: Andamento della `Loss` e della `Test Loss` all'aumentare delle epoche.

Salviamo i pesi nel seguente modo

```
torch.save(model.state_dict(), "
 best_model_MAE_NanoSpugna_idrata_zero_cinque.pth")
```

e li carichiamo in `TestaModelli` per verificare quanto il modello costruito riesca a riprodurre fedelmente i parametri $\mathbf{\Psi}^*$ che caratterizzano la curva di dipersione R_1 ottenuta in laboratorio.

Osserviamo come tale funzione R_1 non appartenga all'insieme di dati sintetici usati durante l'addestramento della rete neurale.

Vengono stampati a schermo i parametri veri e predetti, che sono rispettivamente

Parametro	Reale	Predetto
R_0	3.242	3.217
C^{NH}	63.803	67.484
$\sin^2(\Theta)$	0.759	0.720
$\sin^2(\Phi)$	0.585	0.607
τ_Q	0.718	0.720
ω_-	15.865	15.970
ω_+	19.863	19.875

Tabella 3.1: Confronto dei parametri reali e predetti dalla rete.

In questo caso, la media degli errori relativi commessi è

$$\frac{1}{7}\sum_{i=0}^{6}\left(\frac{|\Psi_i^* - \Psi_i|}{|\Psi_i^*|}\right) = 2.36 \times 10^{-2},$$

inoltre, ω_+ è il parametro meglio predetto, con errore relativo 6.3×10^{-4}, al contrario, il parametro predetto peggio è C^{NH}, il cui errore relativo è 5.8×10^{-2}.

Infine, osserviamo il grafico della curva di dispersione reale e la sua previsione,

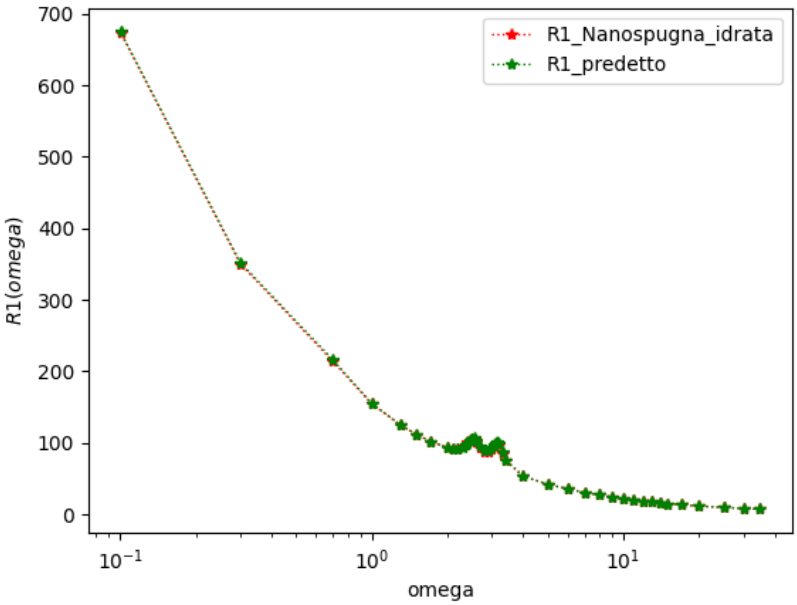


Figura 3.4: Confronto tra la curva di dispersione e la sua previsione.

dove l'asse x è in scala logaritmica.

Capitolo 4

Performance delle Reti Neurali sui Dati NMR

In questo capito confrontiamo i risultati ottenuti variando le caratteristiche del modello, come ad esempio la profondità della rete, la funzione di perdita e il tipo di campione preso in analisi.

4.1 Profondità della rete

Dalla profondità della rete dipende la performance del modello e la durata dell'addestramento. Quindi, vogliamo trovare una lunghezza della rete opportuna, che sia un compromesso accettabile tra le variabili in gioco.

Similmente alla sezione 3.2.1, consideriamo come campione una nanospugna idratata e fissiamo `alpha=0.5`. Al variare della profondità della rete riportiamo la media degli errori relativi commessi nella previsione dei parametri.
Quindi, consideriamo una rete con soli due *layer* nascosti di nodi rispettivamente 256 e 64. Ad ogni esperimento ne incrementiamo la profondità aggiungendo, subito dopo l'input, un ulteriore strato composto di 256 unità. Scegliamo tale numero poiché siamo interessati ad aggiungere ridondanza ai dati, quindi, è opportuno usare un numero sufficientemente più grande di 48 (che è la lunghezza massima della partizione del dominio negli esperimenti svolti).

Osserviamo la media degli errori relativi commessi nella previsione dei parametri usando reti neurali con un numero di *layer* nascosti che va da due a sei.

`neurons_per_hidden_layer`	media errori relativi
`[256, 64]`	1.77×10^{-1}
`[256, 256, 64]`	6.562×10^{-2}
`[256, 256, 256, 64]`	3.244×10^{-2}
`[256, 256, 256, 256, 64]`	$\mathbf{2.357 \times 10^{-2}}$
`[256, 256, 256, 256, 256, 64]`	3.815×10^{-2}

Tabella 4.1: Confronto dei parametri reali e predetti dalla rete.

La rete più performante è quella avente 5 strati nascosti, per questo motivo, in tutti gli esperimenti successivi fissiamo

```
neurons_per_hidden_layer = [256, 256, 256, 256, 64]
```

Osserviamo anche graficamente l'evoluzione della media degli errori relativi al crescere della profondità della rete.

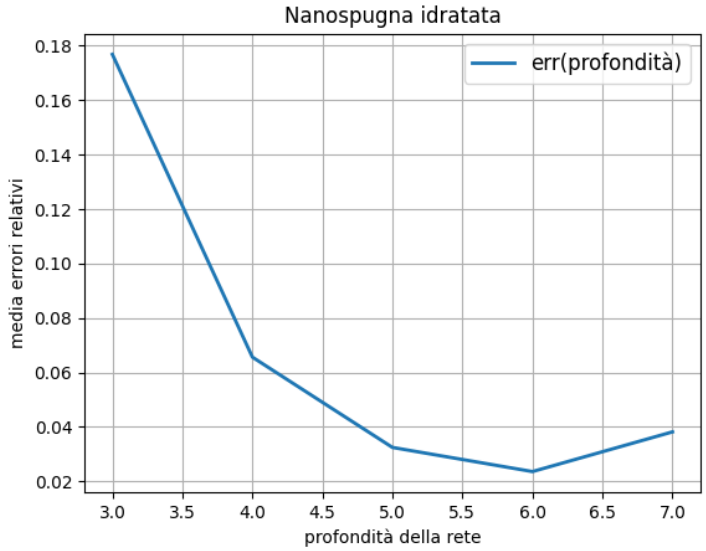


Figura 4.1: Media degli errori relativi al variare della profondità della rete.

4.2 Confronto tra MAE e MSE

Ricordiamo che la funzione di perdita che desideriamo utilizzare durante l'addestramento della rete è una combinazione convessa di parametro `alpha` tra la *loss* eseguita sui parametri e quella riferita al campionamento della funzione.

Come riportato nel capitolo 1.1.3, durante la fase di confronto tra i valori in output e quelli desiderati, possiamo tenere conto del valore assoluto o quadratico dell'errore.

Quindi, concordemente a quanto visto in 3.2.1, analizziamo una nanospugna di ciclodestrina idratata e addestriamo la rete su 10000 epoche. Però, in questo caso, usiamo come funzione di perdita la combinazione convessa di parametro `alpha=0.5` tra la media degli errori quadratici commessi sulla riproduzione dei parametri $\mathbf{\Psi}^*$ e quella riferita alla ricostruzione della funzione R_1.

Riportiamo il valore dei parametri reali e predetti.

Parametro	Reale	Predetto
R_0	3.242	2.704
C^{NH}	63.803	65.875
$\sin^2(\Theta)$	0.759	0.747
$\sin^2(\Phi)$	0.585	0.543
τ_Q	0.718	0.674
ω_-	15.865	15.911
ω_+	19.863	20.026

Tabella 4.2: Confronto dei parametri reali e predetti dalla rete (MSE).

In questo caso, la media degli errori relativi commessi è

$$\frac{1}{7}\sum_{i=0}^{6}\left(\frac{|\Psi_i^* - \Psi_i|}{|\Psi_i^*|}\right) = 5.14 \times 10^{-2} > 2.36 \times 10^{-2},$$

ed è quindi più alta di quella ottenuta in 3.2.1, usando MAE. Il parametro predetto meglio è ω_- con errore relativo 2.9×10^{-3}, al contrario, il parametro predetto peggio è R_0, il cui errore relativo è 1.7×10^{-1}.
A parità di modello, tenendo conto del valore quadratico, anziché di quello assoluto, dell'errore, si ottengono risultati meno performanti. Perciò decidiamo di usare MAE anche negli esperimenti successivi.

4.3 Scelta del parametro `alpha`

Per quanto concerne l'`elastic net`, combinazione convessa tra la media degli errori assoluti commessi sulla previsione dei parametri e sulla ricostruzione del grafico del profilo NMRD (vedi 3.2), siamo interessati a capire se esiste un valore del parametro `alpha` che, a prescindere dal campione preso in analisi, permetta alla rete di essere il più performante possibile.

Mostriamo la media degli errori relativi ottenuta variando il tipo di campione tra i disponibili e usando reti neurali addestrate rispetto ad una funzione di costo elastic net di parametro `alpha` $\in \{0, 0.3, 0.5, 0.7, 1\}$

`alpha`	Nanospugna secca	Nanospugna idratata	Parmigiano	Formaggio
0	$\mathbf{1.96 \times 10^{-2}}$	4.56×10^{-2}	9.15×10^{-2}	$\mathbf{8.64 \times 10^{-2}}$
0.3	4.06×10^{-2}	2.53×10^{-2}	$\mathbf{4.88 \times 10^{-2}}$	1.08×10^{-1}
0.5	4.80×10^{-2}	$\mathbf{2.36 \times 10^{-2}}$	1.36×10^{-1}	1.06×10^{-1}
0.7	2.19×10^{-2}	3.56×10^{-2}	1.65×10^{-1}	2.44×10^{-1}
1	4.78×10^{-2}	4.38×10^{-2}	1.78×10^{-1}	2.10×10^{-1}

Tabella 4.3: Confronto della media degli errori relativi ottenuta variando campione e parametro.

Come possiamo osservare dalla tabella, non esiste un **alpha** ottimale, univoco per tutti i campioni, che permette di addestrare la rete affinché questa riesca a prevedere i parametri $\mathbf{\Psi^*}$ nella maniera migliore possibile. Possiamo, però, scegliere il parametro **alpha** in dipendenza del campione selezionato.

Al variare di `alpha` $\in \{0, 0.3, 0.5, 0.7, 1\}$, osserviamo la variazione della media degli errori relativi per quanto riguarda la nanospugna idratata.

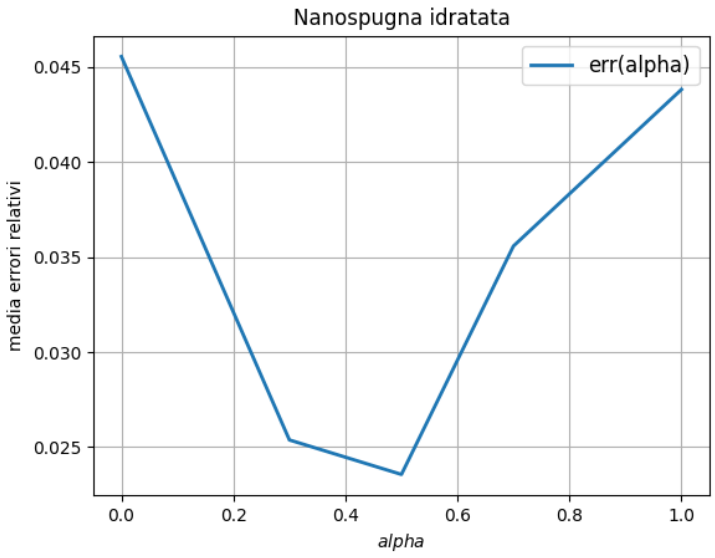


Figura 4.2: Media degli errori relativi al variare di `alpha`.

In questo campione è interessante notare come ogni combinazione convessa non banale delle due *loss* dia una media degli errori relativi più bassa di quella ottenuta altrimenti. Questo comportamento dipende dal fatto che il problema sia mal condizionato, ovvero le soluzioni di questo sono molto sensibili a piccole perturbazioni dei dati iniziali. Per questo motivo, usare degli approcci "puri", che tengono conto o solo della forma del grafico o solo dei suoi parametri può non essere la strategia migliore.

4.4 Perturbazione della curva di dispersione

All'interno del dataset in Excel è presente anche l'errore sperimentale misurato commesso dal relaxometro durante le misurazioni in laboratorio.

Similmente alla sezione 3.2.1, consideriamo come campione una nanospugna idratata e perturbiamo la curva utilizzando il metodo Monte Carlo. Nello specifico, per ogni punto della funzione R_1, estraiamo un valore casuale da una distribuzione Gaussiana avente come media 0 e come varianza l'errore puntuale associato. Usiamo tali valori per perturbare l'immagine dei punti del dominio all'interno del range di frequenze

$$\left[\nu_-^* - \frac{\nu_+^* - \nu_-^*}{2}, \nu_+^* + \frac{\nu_+^* - \nu_-^*}{2}\right].$$

Tale intervallo contiene i punti del dominio a cui sono associati i picchi quadrupolari. Decidiamo di non perturbare ogni punto della curva poiché nella nostra trattazione il vettore $\boldsymbol{f}$ è fissato, quindi la variazione del profilo può avvenire solo in corrispondenza dei picchi.

Confrontiamo i parametri veri con quelli predetti dalla rete quando forniamo in input la curva perturbata.

Parametro	Reale	Predetto
R_0	3.242	3.191
C^{NH}	63.803	67.439
$\sin^2(\Theta)$	0.759	0.724
$\sin^2(\Phi)$	0.585	0.609
τ_Q	0.718	0.724
ω_-	15.865	15.985
ω_+	19.863	19.909

Tabella 4.4: Confronto dei parametri reali e predetti dalla rete.

In questo caso, la media degli errori relativi commessi è

$$\frac{1}{7}\sum_{i=0}^{6}\left(\frac{|\Psi_i^* - \Psi_i|}{|\Psi_i^*|}\right) = 2.55 \times 10^{-2},$$

ed è, quindi, molto simile al risultato ottenuto in 3.2.1.
I parametri predetti meglio e peggio continuano ad essere gli stessi, ovvero ω_+ e C^{NH}, con rispettivamente errori relativi 2.3×10^{-3} e 5.7×10^{-2}.

Infine, osserviamo il grafico della curva di dispersione reale e la sua previsione,

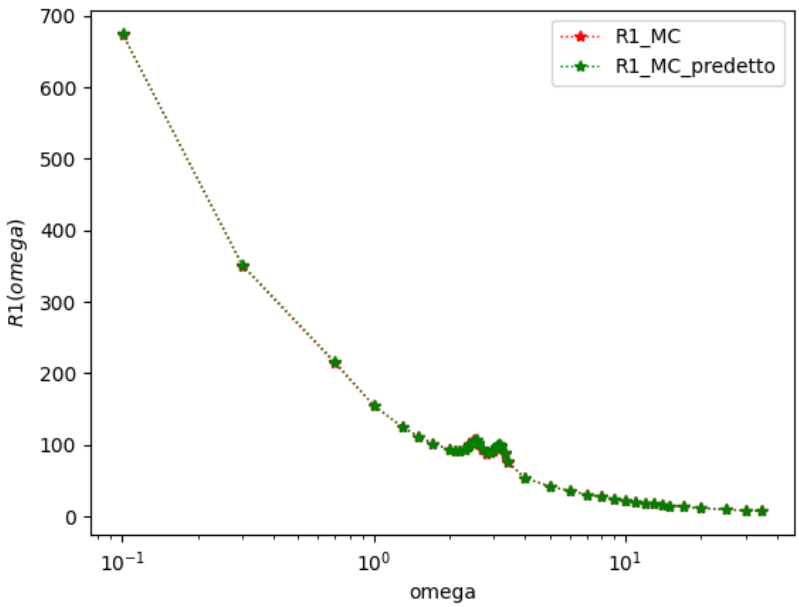


Figura 4.3: Confronto tra la curva di dispersione e la sua previsione.

dove l'asse x è in scala logaritmica.

Conclusioni e Prospettive future

Sommario dei Principali Risultati

In quest'elaborato è stato compiuta un'indagine preliminare sull'impiego di tecniche di apprendimento automatico dei parametri quadrupolari che caratterizzano i profili NMRD, ottenuti per mezzo della FFC-NMR.
Nello specifico, si è ricorso all'uso di percettroni multistrato (MLP), che ricordiamo essere dei modelli fondamentali di rete neurale artificiale. Questi sono composti da molteplici strati di nodi in un grafo orientato, dove ogni strato risulta completamente connesso al successivo.

L'obiettivo è stato quello di costruire MLP capaci di mappare ogni curva di dispersione in una previsione affidabile dei parametri quadrupolari che la caratterizzano.

Sono stati eseguiti degli esperimenti per dedurre quale potesse essere la configurazione migliore del modello.
Ad ogni nodo in uno stesso strato, è stata associata un'opportuna funzione di attivazione lineare a tratti, e il numero di strati è stato deciso confrontando le performance dei vari modelli.
Data la natura del problema inverso, non lineare e mal condizionato, si è posta particolare attenzione alla fase di addestramento del modello. Quindi, è stato individuato un opportuno ottimizzatore con tasso di apprendimento dinamico. Inoltre è stata scelta la funzione di costo migliore calcolando una combinzione convessa tra la media degli errori assoluti commessi sulla previsione dei parametri e sulla ricostruzione del grafico del profilo NMRD. Il motivo alla base di questa scelta implementativa è che le soluzioni del problema sono molto sensibili a piccole perturbazioni dei dati iniziali, perciò, è spesso opportuno far sì che la rete impari a prevedere i parametri tenendo conto anche della forma della curva corrispondente.

Infine il modello è stato testato sia su curve di dispersione reali fornite in seguito ad esperimenti in laboratorio, sia su delle opportune perturbazioni di questi, e i risultati di previsione dei parametri ottenuti sono sempre soddisfacenti.

Suggerimenti per Ricerche Future

È possibile generalizzare ulteriormente il lavoro svolto. Infatti, ogni volta che generiamo una nuova rete, assumiamo sempre di conoscere a priori il tipo di campione preso in esame. Quest'assunzione fa sì che il vettore sparso $\boldsymbol{f}$, ottenuto campionando la funzione di distribuzione $f(\tau)$ in 100 punti, sia noto. Al contrario, nella formulazione del modello discreto del profilo NMRD 2.6, $\boldsymbol{f}$ è una variabile del problema al pari di $\boldsymbol{\Psi}$ e R_0.

La difficoltà nell'estendere il numero di incognite in tale metodologia risidede nella lunghezza del vettore $\boldsymbol{f}$ e dal fatto che la funzione oggetto sia non lineare e mal condizionata. Aumentare sensibilmente il numero di parametri da prevedere impatta negativamente sulle performance del modello. Tale problema potrebbe essere gestito sfruttando la sparsità di $\boldsymbol{f}$, tenendo conto che il numero di valori non nulli (o in generale non prossimi allo 0) sia "piccolo" (ad esempio minore o uguale a 5). Seguendo tale idea bisogna tener conto contemporaneamente non solo del valore degli elementi significativi ma anche della loro posizione all'interno del vettore.

Una volta generalizzata tale procedura, è possibile testare il modello fornendo in entrata delle curve ottenute perturbando i profili di dispersione reali in ogni punto, e non solo in prossimità dei picchi quadrupolari.

Bibliografia

[1] Conte Pellegrino (2021), *Applications of fast field cycling NMR relaxometry*, Elsevier.
https://www.sciencedirect.com/science/article/abs/pii/S0066410321000168

[2] Hansen, D Flemming (2019), *Using deep neural networks to reconstruct non-uniformly sampled NMR spectra*, Springer.
https://link.springer.com/article/10.1007/s10858-019-00265-1

[3] Osheter Tatiana, Campisi Pinto Salvatore, Randieri Cristian, Perrotta Andrea, Linder Charles, Weisman Zeev (2023), *Semi-Autonomic AI LF-NMR Sensor for Industrial Prediction of Edible Oil Oxidation Status*, Sensors.
https://www.mdpi.com/1424-8220/23/4/2125

[4] Ian Goodfellow, Yoshua Bengio, Aaron Courville (2016), *Deep Learning*, MIT Press.
http://www.deeplearningbook.org

[5] Aston Zhang, Zack C. Lipton, Mu Li, Alex J. Smola (2023), *Dive into Deep Learning*, Cambridge University Press.
https://D2L.ai

[6] Sara Brown (2021), *Machine learning, explained*, MIT Sloan School of Management.
https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

[7] Jason Brownlee (2021), *Regression Metrics for Machine Learning*, Guiding Tech Media.
https://machinelearningmastery.com/regression-metrics-for-machine-learning/

[8] Margherita Porcelli (2022), *Matrix and Tensor Techniques for Data Science - Part 2*, Alma Mater Studiorum Università di Bologna.
https://virtuale.unibo.it/pluginfile.php/1231133/mod_resource/content/1/DataScience_2021-22_marghe_completa.pdf

[9] Jason Brownlee (2017), *A Gentle Introduction to Mini-Batch Gradient Descent and How to Configure Batch Size*, Guiding Tech Media.

[10] *Adam*, PyTorch 2.0 documentation.
https://pytorch.org/docs/stable/generated/torch.optim.Adam.html

[11] Anil Johny, K. N. Madhusoodanan (2021), *Dynamic Learning Rate in Deep CNN Model for Metastasis Detection and Classification of Histopathology Images*, PubMed Central.
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8563135/

[12] *CosineAnnealingLR*, PyTorch 2.0 documentation. https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.CosineAnnealingLR.html

[13] Léon Bottou, Frank E. Curtis, Jorge Nocedal (2018), *Optimization Methods for Large-Scale Machine Learning*, arXiv.
https://arxiv.org/pdf/1606.04838v3.pdf

[14] Catherine F. Higham and Desmond J. Higham. (2019), *Deep Learning: An Introduction for Applied Mathematicians*, Society for Industrial and Applied Mathematics.
https://epubs.siam.org/doi/epdf/10.1137/18M1165748

[15] Joseph P. Hornak (2021), *The Basics of MRI.*
https://www.cis.rit.edu/htbooks/mri/inside.htm

[16] Giovanni Vito Spinelli (2020), *Magnetic Resonance Fingerprinting (MRF) for low field NMR: preliminary study*, Alma Mater Studiorum Università di Bologna.
https://amslaurea.unibo.it/23206/

[17] Leonardo Brizi (2018), *Fisica Applicata alla Medicina 1 – NMR*, Alma Mater Studiorum Università di Bologna.

[18] Leonardo Brizi (2018), *Fisica Applicata alla Medicina 2 – NMR: T_1 e T_2*, Alma Mater Studiorum Università di Bologna.

[19] G. Landi, G.V. Spinelli, F. Zama, D. Chillura Martino, P. Conte, P. Lo Meo, V. Bortolotti (2023), *An automatic L_1-based regularization method for the analysis of FFC dispersion profiles with quadrupolar peaks*, Applied Mathematics and Computation.
https://www.sciencedirect.com/science/article/abs/pii/S0096300322008773

[20] Paolo Lo Meo, Samuele Terranova, Antonella Di Vincenzo, Delia Chillura Martino, and Pellegrino Conte (2021), *Heuristic Algorithm for the Analysis of Fast Field Cycling (FFC) NMR Dispersion Curves*, American Chemical Society.
https://pubs.acs.org/doi/10.1021/acs.analchem.1c01264

[21] B. Halle, H. Jóhannesson, K. Venu (1998), *Model-Free Analysis of Stretched Relaxation Dispersions*, Journal of Magnetic Resonance.
https://www.sciencedirect.com/science/article/abs/pii/S1090780798915348

[22] B. Halle (2009), *The physical basis of model-free analysis of NMR relaxation data from proteins and complex fluids*, The Journal of chemical physics.
https://pubs.aip.org/aip/jcp/article-abstract/131/22/224507/949750/The-physical-basis-of-model-free-analysis-of-NMR?redirectedFrom=fulltext

[23] Danuta Kruk, Elzbieta Masiewicz, Anna M. Borkowska, Pawel Rochowski, Pascal H. Fries, Lionel M. Broche, David J. Lurie (2019), *Dynamics of Solid Proteins by Means of Nuclear Magnetic Resonance Relaxometry*, PubMed Central.
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6920843/

[24] T. Jeoh, N. Karuna, N.D. Weiss, L.G. Thygesen (2017), *Two-Dimensional 1H-Nuclear Magnetic Resonance Relaxometry for Understanding Biomass Recalcitrance*, ACS Sustainable Chemistry & Engineering.
https://pubs.acs.org/doi/10.1021/acssuschemeng.7b01588

[25] Pellegrino Conte, Luciano Cinquanta, Paolo Lo Meo, Francesca Mazza, Anna Micalizzi, Onofrio Corona (2021), *Fast field cycling NMR relaxometry as a tool to monitor Parmigiano Reggiano cheese ripening*, ScienceDirect.
https://www.sciencedirect.com/science/article/abs/pii/S096399692030870X

[26] Elif Gokcen Ates, Valentina Domenici, Małgorzata Florek-Wojciechowska, Anton Gradišek, Danuta Kruk, Nadica Maltar-Strmečki, Mecit Oztop, Emin Burcin Ozvural, Anne-Laure Rollet (2021), *Field-dependent NMR relaxometry for Food Science: Applications and perspectives*, ScienceDirect.
https://www.sciencedirect.com/science/article/abs/pii/S0924224421001333

[27] Kazufumi Ito, Bangti Jin, Tomoya Takeuchi (2022), *A Regularization Parameter for Nonsmooth Tikhonov Regularization*, SIAM Journal on Scientific Computing.
https://epubs.siam.org/doi/10.1137/100790756

[28] *torch*, PyTorch 2.0 documentation.
https://pytorch.org/docs/stable/torch.html

[29] *NumPy user guide*.
https://numpy.org/doc/stable/user/index.html#user

[30] *User Guide*, pandas.
https://pandas.pydata.org/docs/user_guide/index.html#user-guide

[31] *Pyplot tutorial*, matplotlib.
https://matplotlib.org/stable/tutorials/pyplot.html

[32] *random — Generate pseudo-random numbers*, Python.
https://docs.python.org/3/library/random.html