UNIVERSITÀ DEGLI STUDI DI PADOVA

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA MECCATRONICA

DIPARTIMENTO DI TECNICA E GESTIONE DEI SISTEMI INDUSTRIALI

TESI DI LAUREA TRIENNALE

Modellazione tramite reti neurali della conduttività termica nei nanofluidi

RELATORE: **Dott.ssa MONICA REGGIANI**
Dipartimento di Tecnica e Gestione dei Sistemi Industriali

CORRELATORE: **Dott.ssa ELENA CESERACCIU**
Dipartimento di Tecnica e Gestione dei Sistemi Industriali

LAUREANDO: **ALEX BATTISTON**

ANNO ACCADEMICO 2010-2011

INDICE

SOMMARIO

Il presente lavoro ha l'obiettivo di sviluppare un sistema a reti neurali per effettuare la previsione dei valori di conduttività termica dei nanofluidi, ricercando in particolare la struttura di rete neurale più adatta.
Le previsioni effettuate dalla rete neurale tramite software vengono successivamente confrontate con le misurazioni effettuate in laboratorio per verificare la capacità previsionale.
Nel primo capitolo vengono trattati i nanofluidi, descrivendone le proprietà fondamentali e come queste influiscono sulla conduttività termica. Vengono inoltre trattati alcuni modelli che cercano di predire il comportamento termico dei nanofluidi e spiegate le motivazioni per cui le stime con tali modelli analitici non si avvicinano ai risultati sperimentali ottenuti.
Nel secondo capitolo le reti neurali sono oggetto di studio per capire il loro funzionamento e la loro struttura. Viene inoltre trattato l'algoritmo di backpropagation utilizzato per l'addestramento delle reti neurali.
Nel terzo capitolo vengono descritti la raccolta dei dati e il codice sorgente del software utilizzato.
Nel quarto i risultati ottenuti sono sottoposti ad analisi tramite grafici comparativi e tabelle numeriche.
Nel quinto e ultimo capitolo vengono tratte le conclusioni sul lavoro effettuato.

INTRODUZIONE

Il presente lavoro ha l'obiettivo di predire la conduttività termica di un nanofluido tramite l'utilizzo di una tecnica di apprendimento supervisionato quale la rete neurale.
La conduttività termica è una delle proprietà più importanti di un nanofluido e ne determina l'efficienza nel trasferimento di calore [1].
Tuttavia, non è ancora chiaro quali siano i legami tra le proprietà dei componenti che costituiscono il nanofluido, e ciò non consente la preparazione di nanofluidi stabili, con caratteristiche e proprietà desiderate.
I modelli analitici, di correlazione e di simulazione sviluppati per la predizione della conduttività termica sottostimano o sovrastimano i risultati sperimentali, dunque non consentono di ottenere predizioni soddisfacenti. Finora, l'unico metodo per ottenere risultati certi o molto attendibili è il metodo sperimentale, ma risulta molto dispendioso sia in termini di tempo e di costi.
La scelta delle reti neurali può risultare un efficiente metodo per la risoluzione dei problemi sopra descritti in quanto, tramite la ricerca di una struttura della rete neurale appropriata, consente di simulare tramite computer la relazione ingresso-uscita che consente la predizione della conduttività termica, al variare dei parametri utilizzati.
Dopo una breve trattazione delle metodologie di predizione, ci si focalizza sullo studio delle reti neurali.
I dati raccolti durante la prima fase, quella sperimentale, vengono utilizzati come input per il lavoro di sviluppo della rete neurale, per far imparare alla rete a riconoscere la relazione incognita che lega le variabili d'ingresso (concentrazione di volume della particella e la temperatura) e d'uscita (conduttività termica del nanofluido), diventando in grado di effettuare previsioni anche dove l'uscita non è nota a priori.
Il software utilizzato è stato scritto con il linguaggio C++ e si basa sulla libreria Shark, che è una libreria orientata agli oggetti per la progettazione di sistemi adattativi, utilizzata perciò per la creazione e l'utilizzo della rete neurale.
L'analisi del codice sorgente offre la possibilità di osservare quali siano i parametri per la configurazione delle rete neurale. Il successo della previsione dipende sia dalla tipologia e dalla struttura di rete neurale utilizzata, sia dal set di dati utilizzati per l'addestramento. Dopo la fase di addestramento, la rete neurale consente di ottenere un'adeguata capacità di generalizzazione, ovvero la capacità di predire un risultato di output il più possibile vicino a quello reale, senza che lo stimolo d' ingresso sia uguale agli esempi forniti.
Durante lo svolgimento della tesi, sono state utilizzate diverse combinazioni di strutture e due diversi set di dati (Al_2O_3/Acqua e TiO_2/Acqua) per trovare il modello di rete neurale più adatto per effettuare la migliore previsione di conduttività termica del nanofluido. Per ciascun modello sono stati stimati l'errore di addestramento (capacità di rappresentare la relazione input-output tra i dati forniti) e di validazione (capacità predittiva).
I risultati ottenuti dall'elaborazione del software i sono stati infine rappresentati, sia tramite grafici che tramite tabelle numeriche, confrontando l'andamento dell'output misurato e dell'output predetto.

CAPITOLO 1

Problematica

NANOFLUIDI

Nomenclatura:
Cp = calore specifico, J/kgK
k = conduttività termica, W/mK
m = massa, Kg
V = volume, m3
v = concentrazione di volume
ρ = densità, kr/m3
μ = viscosità, kg/ms

Pedici:
m = matrice fluido base
p = particella
e = effettiva

1.1 Introduzione

I nanofluidi sono una nuova classe di fluidi adibiti allo scambio di calore, generati attraverso una sospensione di particelle della dimensione di nanometri in convenzionali fluidi attraverso raffinati processi nanotecnologici.
La *nanofluid technology* è un campo interdisciplinare di grande importanza che raggruppa la nanoscienza, la nanotecnologia e la fisica tecnica.
In questi recenti anni, l'attività di ricerca e sviluppo per questa nuova tecnologia è notevolmente aumentata, soprattutto per le notevoli applicazioni dove può essere utilizzata.
Il concetto e sviluppo di nanofluidi è direttamente collegato all'andamento della miniaturizzazione e delle nanotecnologie. L'efficacia di un nanofluido rispetto ad un altro può essere diversa e dipende dal tipo di processo utilizzato per generarlo e dalle caratteristiche della nanoparticella e del fluido.
L'idea di piccole particelle metalliche per incrementare la conduttività termica dei fluidi proviene da Maxwell (1873), ma con l'utilizzo di particelle della dimensione di millimetri o micrometri si riscontravano problematiche di sedimentazione, intasamento, abrasione e incremento della caduta di pressione.
Choi, nel 1995, inserì in un fluido delle nanoparticelle e chiamò questo fluido con il nome di "Nanofluid", riscontrando un aumento della conduttività termica dimostrando il lavoro teorico che Maxwell enunciò cento anni prima [2].

	Microparticelle	Nanoparticelle
Stabilità	Sedimentazione	Stabile (rimangono in sospensione quasi infinitamente)
Superficie/Volume rapporto	1	1000 volte più larga rispetto alla microparticella
Conduttività	Bassa	Alta
Ostruzione microcanali	Si	No
Erosione	Si	No
Fenomeni in nanoscale	No	Si

Tabella 1 : Comparazione microparticelle e nanoparticelle

1.2 Composizione e struttura

Le nanoparticelle hanno una dimensione che varia da 1 a 100nm.
Il materiale con cui sono costituite può essere un ossido (Al_2O_3, ZrO_2, SiO_2, CuO), semiconduttore (TiO_2), metallo (Au, Cu), nitruro (AlN, SiN), nanotubi o carburi.
Il fluido base, nel quale vengono incluse le particelle, può essere acqua, glicole etilenico, olio o altri lubrificanti, biofluidi, soluzioni polimeriche e altri fluidi comuni.

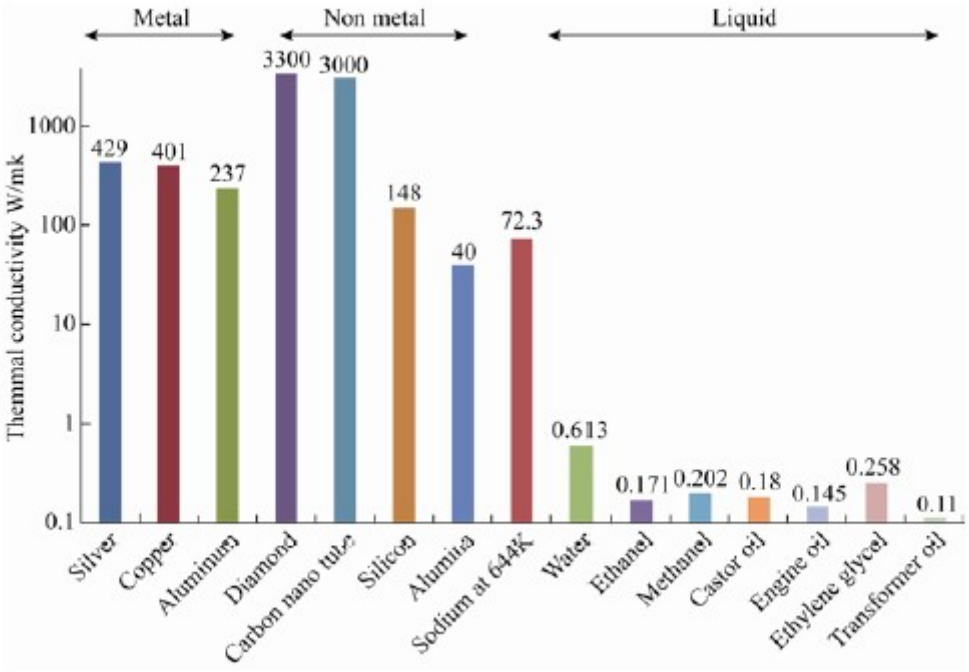

Figura 1: Comparazione della conduttività termica di materiali diversi

In Figura 1 vengono messi in evidenza i diversi valori di conduzione termica per metalli, non metalli e liquidi.
I più comuni fluidi tradizionalmente utilizzati per il trasferimento di calore hanno una bassa conduzione di calore a causa della bassa conduttività termica. Le proprietà fisiche e chimiche dei

nanocomponenti solidi, i quali hanno una alta conduttività termica, consentono di modificare il comportamento dei fluidi in cui sono sospesi, aumentando la conduttività termica del fluido.

Ref.No	Reference	Base fluid	Nano particle& Diameter	Max. Concentration (Vol %)	Maximum Enhancement in thermal conductivity (k) (%)
22	Masuda et al (1993)	Water	Al_2O_3, 13 nm	4.3	30
23	Eastman et al (1996)	Water	Al_2O_3, 33 nm	5	30
24	Pak and Cho (1998)	Water	Al_2O_3, 13 nm	4.3	32
25	Wang et al (1999)	Water	Al_2O_3, 28 nm	4.5	14
25	Wang et al (1999)	Ethylene Glycol	Al_2O_3, 28 nm	8	40
25	Wang et al (1999)	Pump oil	Al_2O_3, 28 nm	7	20
25	Wang et al (1999)	Engine oil	Al_2O_3, 28 nm	7.5	30
26	Lee et al (1999)	Water	Al_2O_3, 24.4 nm	4.3	10
26	Lee et al (1999)	Ethylene Glycol	Al_2O_3, 24.4 nm	5	20
27	Das et al (2003)	Water	Al_2O_3, 38 nm	4	25
28	Xie (2002)	Water	Al_2O_3, 60 nm	5	20
28	Xie (2002)	Ethylene Glycol	Al_2O_3, 60 nm	5	30
28	Xie (2002)	Pump oil	Al_2O_3, 60 nm	5	40
29	Prashar et al (2005)	Water	Al_2O_3, 10 nm	0.5	100
30	Krishnamurthy et al (2006)	Water	Al_2O_3, 20 nm	1	16
26	Lee et al (1999)	Water	CuO, 18.6 nm	4.3	10
26	Lee et al (1999)	Ethylene Glycol	CuO, 18.6 nm	4	20
25	Wang et al (1999)	Water	CuO, 23 nm	10	35
25	Wang et al (1999)	Ethylene Glycol	CuO, 23 nm	15	55
31	Liu et al (2006)	Ethylene Glycol	CuO, 25 nm	5	22.4
27	Das et al (2003)	Water	CuO, 28.6 nm	4	36
24	Pak and Cho (1998)	Water	TiO_2, 27 nm	4.35	10.7
32	Murshad et al (2005)	Water	TiO_2, 15 nm	5	33
33	Li et al (2010)	Water	Al_2O_3, 33 nm	2	25
36	Paul et al (2010)	water	Au, 20 nm	2.6	48

Tabella 2: Comparazione dell'incremento della conduttività termica in funzione del liquido base, della nanoparticella e della massima concentrazione

1.3 Applicazioni

Sono molti i settori industriali in cui l'utilizzo dei nanofluidi può portare un incremento di rendimento e di prestazioni.

Trasporto:
I nanofluidi possono essere utilizzati per i liquidi di raffreddamento, consentendo la rimozione del calore prodotto dal motore e permettendo la riduzione delle dimensioni dell'area frontale di sistemi di raffreddamento del radiatore almeno del 10% .
Inoltre, l'impiego di nanofluidi come additivi nel carburante consente una maggiore efficienza di combustione (10-25%) e una riduzione di emissioni [3].

Scambiatori di calore:
Negli scambiatori di calore che utilizzano fluidi convenzionali, il coefficiente del calore trasferito può aumentare solamente con l'incremento della velocità del fluido e questo richiede un incremento significativo della potenza di pompaggio. Utilizzando i nanofluidi, nelle stesse condizioni, si ottiene un incremento della conduttività termica risparmiando una consistente potenza di pompaggio [1].

Elettronica:

Il calore prodotto dai circuiti integrati viene attualmente dissipato, per la maggior parte dei casi, da ventole. Il forte aumento della densità di potenza, che avrà luogo nei prossimi anni, richiederà una elevata dissipazione termica. Il raffreddamento a liquido è una tecnica che sta sostituendo quella ad aria. La ricerca è indirizzata verso sistemi di dissipazione termica avanzati, quali fluidi a singola fase, fluidi a doppia fase e a nanofluidi [4].

Altri settori sono la biomedicina e il campo energetico.
Nella biomedicina [5] [6], i nanofluidi con nanoparticelle di ossido di ferro possono essere utilizzati come mezzo di trasporto di farmaci o radiazioni all'interno del corpo umano, utilizzando magneti per guidare le particelle nella zona interessata, senza danneggiare i tessuti sani circostanti.
L'impiego dei nanofluidi nel campo energetico comporta un aumento della conduttività termica e consente una maggiore efficienza di energia trasferita e quindi un maggior risparmio energetico, oppure sistemi con stessi rendimenti, ma di dimensioni inferiori.

1.4 Tecniche di realizzazione

Le tecniche per realizzare i nanofluidi sono due: one-step e two-step [1].
La two-step ha inizio con la produzione di nanoparticelle da un processo fisico (esempio l'evaporazione) o chimico (processo di condensazione di gas inerti) e procede con la loro dispersione all'interno del fluido.
La difficoltà di questa tecnica consiste nella tendenza delle nanoparticelle ad agglomerarsi rapidamente prima della dispersione. L'agglomerazione è causata dalle forze di attrazione di van der Waals tra le particelle, e tale agglomerazione tende a disporsi velocemente verso la superficie del fluido.
Questa tecnica è solitamente la più utilizzata, e la sua buona riuscita comporta la monodispersione delle nanoparticelle all'interno del liquido e prestazioni performanti di calore trasferito.
Il metodo two-step viene messo a dura prova con alte concentrazioni di volume di nanoparticelle, perché si vengono a creare le condizioni per generare una elevata agglomerazione di particelle, diminuendo di conseguenza il trasferimento di calore.

La tecnica one-step esegue contemporaneamente la realizzazione e la dispersione dei nanoelementi direttamente nel liquido base attraverso un processo di condensazione delle particelle nel vapore del fluido in una camera vuota. Si ottengono in questo modo particelle non agglomerate che rimangono stabilmente sospese e uniformemente disperse nel fluido base. Questa tecnica è la migliore per i nanofluidi con particelle metalliche perché si previene l'ossidazione delle particelle.
In un'altra sofisticata tecnica one-step, all'interno di una camera vuota, le particelle di TiO_2, CuO e Cu sono prodotte dal riscaldamento di materiale solido da un elettrodo, generando un arco di scintilla e condensando il materiale nel liquido.
Il processo one-step non è molto utilizzato perché richiede una camera vuota che limita la produzione ed è molto dispendioso.

Esistono altre tecniche che vengono utilizzate, e dipendono dalla particolare combinazione di materiali (ad esempio geometrie, densità, carica e superficie delle nanoparticelle) e fluidi.

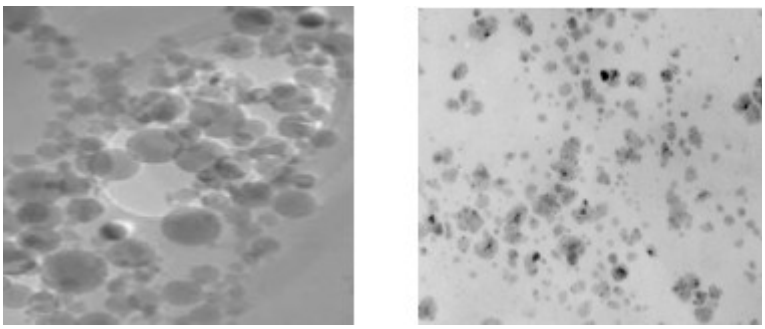

Figura 2: Agglomerazione di nanoparticelle tramite la tecnica two-step (sinistra) e one-step (destra) [7]

1.5 Conduttività termica

Nello sviluppo dell'efficienza nel trasferimento di calore, il ruolo della conduttività termica è un parametro fondamentale [8].
Dalla letteratura e dai dati sperimentali raccolti, sono stati individuati alcuni parametri tramite i quali è possibile effettuare una stima delle proprietà finali del nanofluido (sopratutto conduttività termica e viscosità).
I parametri che influenzano la conduttività termica sono otto [6]:

1. Concentrazione di volume della particella
   La conduttività termica aumenta all'aumentare della concentrazione di volume.

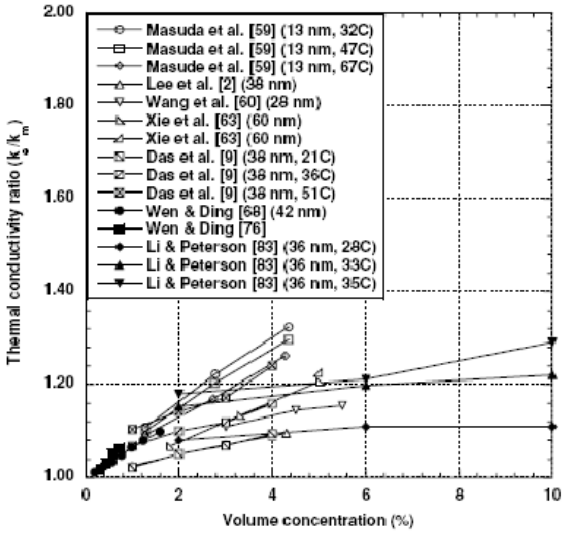

Figura 3: Conduttività termica in funzione della concentrazione di volume (Al_3O_2 in acqua)

2. Materiale della particella
   L'utilizzo di particelle metalliche, rispetto agli ossidi, consente un maggiore scambio termico avendo una più alta conduttività termica. Tuttavia nel processo di creazione della nanoparticella è difficile evitare l'ossidazione, che, inevitabilmente, fa diminuire l'incremento della conduttività termica.

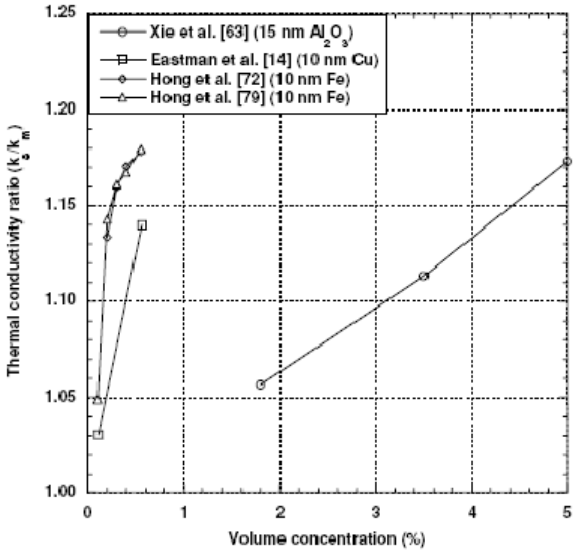

Figura 4: Conduttività termica in funzione del materiale della particella. Particelle di glicole etilenico, rame e ferro

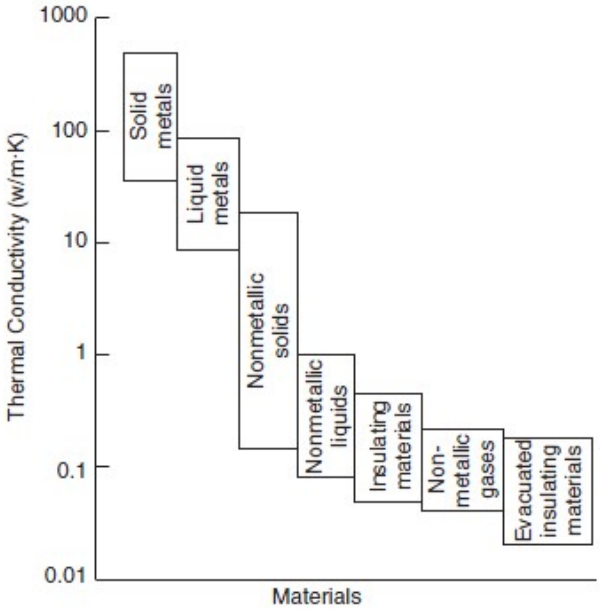

Figura 5: Conduttività termica in funzione del materiale

3. Dimensione della particella
   L'andamento della conduttività termica in funzione di questo parametro non è chiaro. Alcuni ricercatori riportano che, all'aumentare del diametro, si ha un incremento della conduttività termica, ma in altri casi i dati sperimentali mostrano risultati differenti.

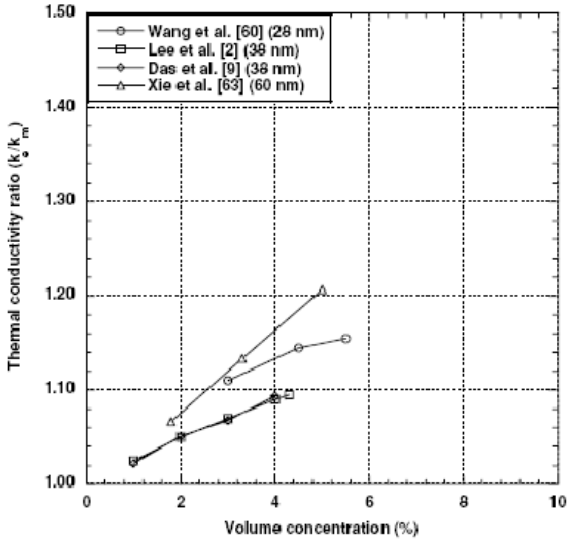

Figura 6: Conduttività termica in funzione della dimensione della particella (Al_2O_3 in acqua)

4. Forma della particella
   In questo caso, tutti gli esperimenti confermano che se l'elongazione è superiore alla sfericità, la conduttività termica aumenta.

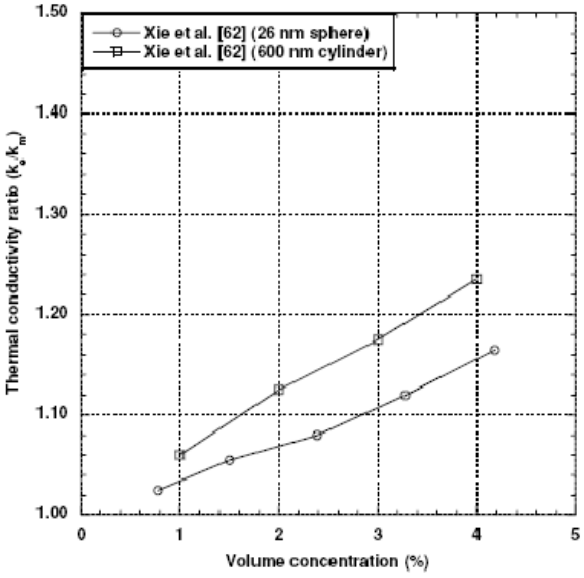

Figura 7: Conduttività termica in funzione della forma della particella, SiC in acqua

5. Materiale del fluido base
   I risultati sperimentali hanno mostrato un maggiore incremento della conduttività termica con fluidi che consentono un basso trasferimento di calore.

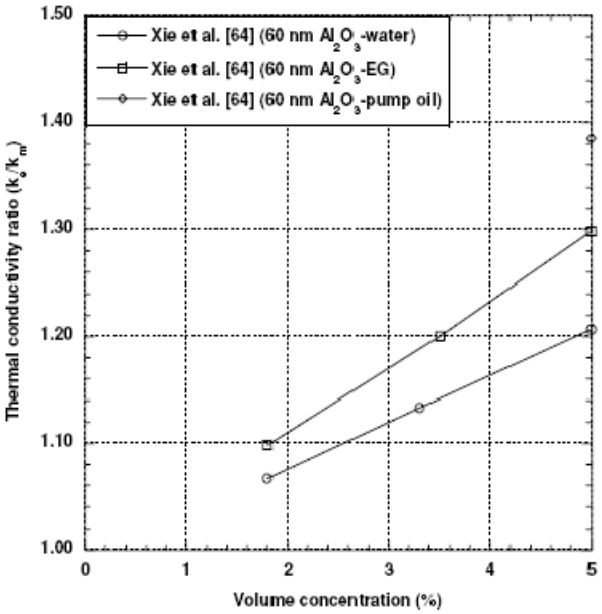

Figura 8: Conduttività termica in funzione del fluido base, Al_2O_3 in fluidi

6. Temperatura
   L'andamento della conduttività termica aumenta con l'incremento della temperatura.

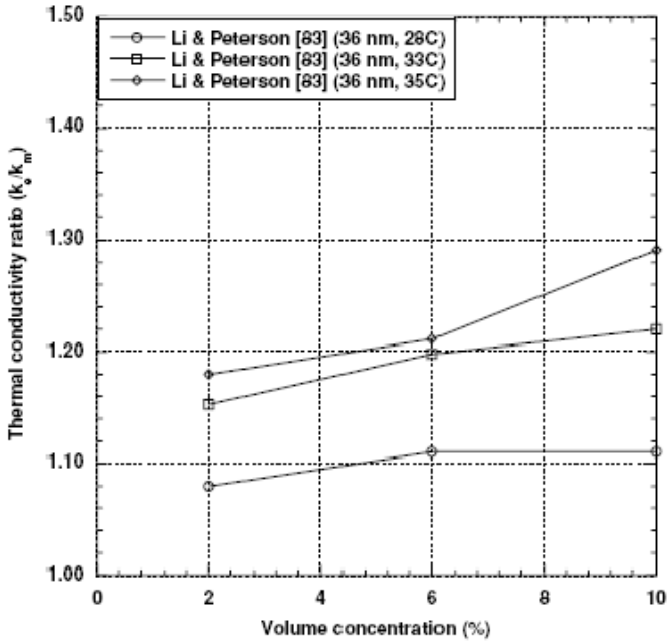

Figura 9: Conduttività termica in funzione della temperatura, Al_2O_3 in fluidi

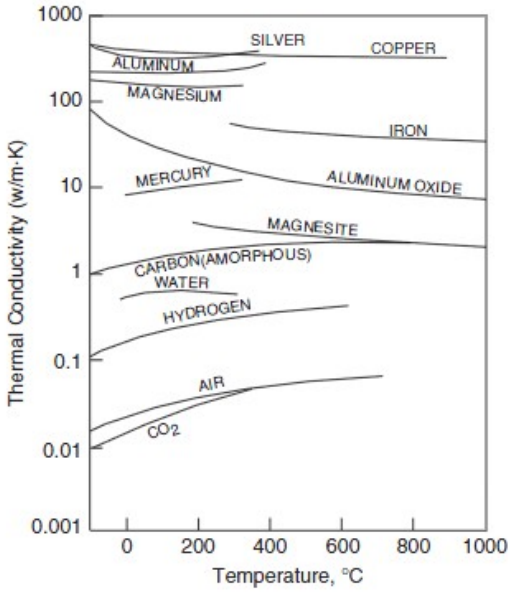

Figura 10: Conduttività termica del materiale in funzione della temperatura

7. Additivi
   L'utilizzo degli additivi serve per mantenere in sospensione le nanoparticelle ed evitare l'agglomerarsi di queste. I risultati sperimentali dimostrano che la conduttività termica aumenta.

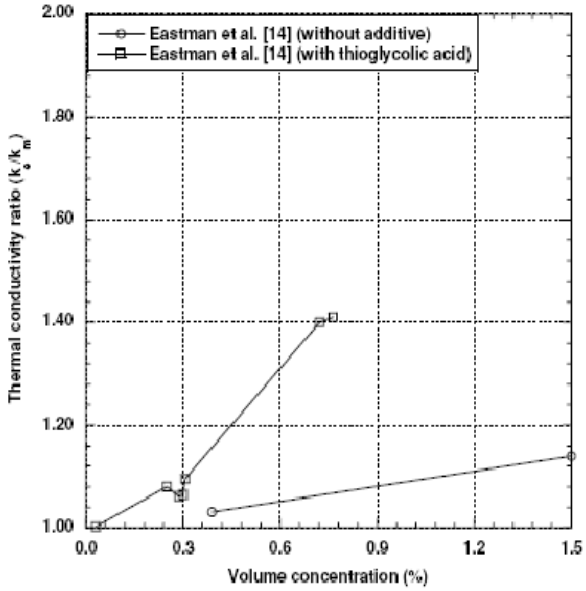

Figura 11: Conduttività termica in funzione dell'additivo, Cu in glicole etilenico

8. Acidità
   I limitati studi effettuati su questa caratteristica dimostrano che, all'aumentare dell'acidità, si ha un incremento di conduttività termica.

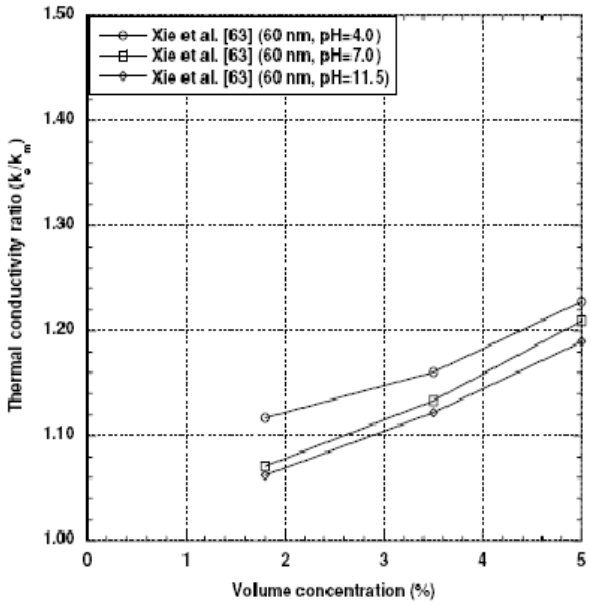

Figura 12: Conduttività termica in funzione dell'acidità

La conoscenza di come i parametri interagiscano tra di loro è di fondamentale importanza, perché consente la preparazione di nanofluidi stabili, con caratteristiche e proprietà desiderate.

1.6 Modelli di predizione

Tuttora, i meccanismi responsabili per un unico comportamento termico dei nanofluidi e la predizione delle loro proprietà non sono ancora chiare. Negli ultimi anni, la ricerca si è focalizzata nel correlare e predire tali proprietà, soprattutto la conduttività termica.
Gli studi si sono concentrati sullo sviluppo di un modello per la predizione della conduttività termica dei nanofluidi, e sono stati elaborati metodi analitici, di correlazione e simulazione.

Modelli analitici:
Molti sono stati i modelli proposti per ottenere una equazione basata sulla teoria.
Il primo studio sulle microparticelle è stato condotto da Hamilton e Crosser, però si è rivelato non adatto alle nanoparticelle che compongono il nanofluido.
I successivi studi si sono basati sulle proprietà del nanofluidi come la densità e il calore specifico, la conduttività termica e la viscosità [6].

- Densità e calore specifico:
  Il calcolo dell'effettiva densità e dell'effettivo calore specifico di un nano fluido, possono essere stimati sulla base dei principi fisici della miscela.

$$\rho_e = \left(\frac{m}{V}\right)_e = \frac{m_m + m_p}{V_m + V_p} = \frac{\rho_m V_m + \rho_p V_p}{V_m + V_p} = (1 - v_p)\rho_m + v_p \rho_p = (1 - v_p)(\rho C_p)_m + v_p (\rho C_p)_p$$

$$C_{pe} = \frac{(1 - v_p)(\rho C_p)_m + v_p (\rho C_p)_p}{(1 - v_p)\rho_m + v_p \rho_p}$$

- Conduttività termica:
  Maxwell è stato uno dei primi a far ricerca analitica sulla conduzione termica attraverso microparticelle sospese di dimensione sferica, senza però considerare l'interazione tra di esse .

$$k_e = k_m + 3v_p \frac{k_p - k_m}{2k_m + k_p - v_p(k_p - k_m)} k_m$$

Tale equazione può essere ottenuta risolvendo l'equazione di Laplace per la temperatura esterna alla particella in due modi: considerando una grande sfera contenente tutte le particelle sferiche con una effettiva conduttività termica k_e inserita in una matrice con una conduttività termica k_m, oppure considerando tutte le particelle sferiche con una conduttività termica k_p inserita in una matrice con una conduttività termica k_m.
Questa formula di Maxwell è considerata come l'equazione di partenza per i successivi sviluppi che considerano i fattori della forma, distribuzione delle particelle, concentrazione, struttura concava e resistenza di contatto.
I risultati predetti con queste equazioni, presentate nella tabella 4, sottostimano o sovrastimano i risultati sperimentali. Questo accade perché le equazioni sono state sviluppate dalla formulazione iniziale di Maxwell elaborata per le microparticelle.
Il comportamento di una nanoparticella è diverso da una microparticella, perciò per migliorare la stima della conduttività termica si deve tenere in considerazione anche gli effetti delle nanoparticelle.

Model type	Year	Author(s)	Notes
Analytical	1962	Hamilton-Crosser [31]	Micro-dimensions, various particle shapes
Analytical	1973	Jeffrey [32]	Micro-dimensions, spheres
Analytical	1986	Davis [34]	Micro-dimensions, spheres
Analytical	1996	Lu-Lin [35]	Micro-dimensions, spheres
Analytical	1987	Hasselman-Johnson [33]	Micro-dimensions, spheres
Analytical	1980	Yamada-Ota [36]	Micro-dimensions, parallelepiped
Analytical	2004	Kumar *et al.* [37]	Nanospheres
Analytical	2003	Wang-Zhou-Peng [38]	Nano-dimensions, network of non-metallic spheres
Analytical	2003	Xuan-Li-Hu [39]	Nano-dimensions, network of spheres
Analytical	2005	Prasher *et al.* [29]	Nanospheres
Analytical	2003	Yu-Hull-Choi [40]	Nanospheres
Analytical	2003	Nan-Shi-Lin [41]	Nano-dimensions, carbon nanotubes suspensions
Correlation	2004	Jang-Choi [15]	Nanospheres
Correlation	2004	Yu-Choi [23]	Nanospheres
Correlation	2005	Xue-Xu [25]	Nanospheres with interfacial shells
Correlation	2003	Xue [26]	Nanospheres and nanotubes with interfacial shells
Correlation	2005	Prasher *et al.* [29]	Nanospheres
Correlation	2003	Wang-Zhou-Peng [38]	Network of nanospheres with interfacial shells
Correlation	2004	Koo-Kleinstreuer [42]	Nanospheres
Simulation	2004	Bhattacharya et al. [43]	Brownian dynamics
Simulation	2005	Xuan-Yao [44]	Lattice Boltzmann
Simulation	2004	Xue *et al.* [27]	Non-equilibrium molecular dynamics
Simulation	2004	Shenogin *et al* [28].	Classical molecular dynamics
Simulation	2005	Present work	Finite Elements

Tabella 3: Modelli per il calcolo della conduttività termica [8]

Maxwell 1873 (spheres)

$$k_e = k_m + 3v_p \frac{k_p - k_m}{2k_m + k_p - v_p(k_p - k_m)} k_m$$

Rayleigh 1892 (spheres, simple cubic arrays)

$$k_e = k_m + 3v_p \frac{k_p - k_m}{2k_m + k_p - v_p\left(1 + 3.939 v_p^{10/3} \frac{k_p - k_m}{4k_m + 3k_p}\right)(k_p - k_m)} k_m$$

Wiener 1912 (arbitrary particles)

$$k_e = k_m + (u+1)v_p \frac{k_p - k_m}{uk_m + k_p - v_p(k_p - k_m)} k_m, \quad 0 \le u \le \infty$$

Fricke 1924; Fricke 1953 (ellipsoids)

$$k_e = k_m + \frac{1}{3} v_p \sum_{i=a,b,c} \frac{k_p - k_m}{k_m + d_{pi}(k_p - k_m)} k_m$$

Bruggeman 1935 (spheres)

$$1 - v_p = \frac{k_p - k_e}{k_p - k_m} \left(\frac{k_m}{k_e}\right)^{1/3}$$

Bruggeman 1935; Böttcher 1945; Landauer 1952 (spheres)

$$(1 - v_p)\frac{k_m - k_e}{2k_e + k_m} + v_p \frac{k_p - k_e}{2k_e + k_p} = 0$$

Polder and van Santen 1946; Taylor 1965; Taylor 1966 (ellipsoids)

$$k_e = k_m + \frac{1}{3} v_p \sum_{i=a,b,c} \frac{k_p - k_m}{k_e + d_{pi}(k_p - k_e)} k_e$$

Kerner 1956; Pauly and Schwan 1959; Schwan, et al. 1962; Lamb, et al. 1978; Benveniste and Milch 1991; Yu and Choi 2003; Wang, et al. 2003 (spheres with shells)

$$k_e = k_m + 3v_c \frac{k_c - k_m}{2k_m + k_c - v_c(k_c - k_m)} k_m$$

Van de Hulst 1957; Lu and Song 1996; Xue 2000 (spheres with shells)

$$(1 - v_c)\frac{k_m - k_e}{2k_e + k_m} + v_c \frac{k_c - k_e}{2k_e + k_c} = 0$$

Hamilton and Crosser 1962 (arbitrary particles)

$$k_e = k_m + 3\Psi^{-1} v_p \frac{k_p - k_m}{(3\Psi^{-1} - 1)k_m + k_p - v_p(k_p - k_m)} k_m$$

Jeffrey 1973 (spheres)

$$k_e = k_m + 3v_p \left[1 + v_p \frac{\sigma_1(2k_m + k_p) + (k_p - k_m)}{2k_m + k_p}\right] \frac{k_p - k_m}{2k_m + k_p} k_m$$

Granqvist and Hunderi 1977; Granqvist and Hunderi 1978 (ellipsoids)

$$(1 - v_p)\frac{k_m - k_e}{2k_e + k_m} + \frac{1}{9} v_p \sum_{i=a,b,c} \frac{k_p - k_e}{k_e + d_{pi}(k_p - k_e)} = 0$$

Landauer 1978 (spheres)

$$(1 - v_s - v_p)\frac{k_m - k_e}{2k_e + k_m} + v_s \frac{k_s - k_e}{2k_e + k_s} + v_p \frac{k_p - k_e}{2k_e + k_p} = 0$$

Davis 1986 (spheres)

$$k_e = k_m + 3v_p(1 + v_p\sigma_1)\frac{k_p - k_m}{2k_m + k_p - v_p(k_p - k_m)} k_m$$

Benveniste 1987; Hasselman and Johnson 1987 (spheres with contact resistance)

$$k_e = k_m + 3v_p \frac{k_p^R - k_m}{2k_m + k_p^R - v_p(k_p^R - k_m)} k_m$$

Xue 2000 (ellipsoids)

$$(1-v_p)\sum_{i=a,b,c}\frac{k_m-k_e}{k_e+d_{mi}(k_m-k_e)}+v_p\sum_{i=a,b,c}\frac{k_p-k_e}{k_e+d_{pi}(k_p-k_e)}=0$$

Nan, et al. 2003 (carbon nanotubes)

$$k_e=k_m+\frac{1}{3}v_p k_p$$

Wang, et al. 2003 (spheres)

$$k_e=k_m+\frac{3v_p\int_0\frac{k_p(r)-k_m}{2k_m+k_p(r)}n(r)dr}{(1-v_p)+3v_p\int_0\frac{k_m}{2k_m+k_p(r)}n(r)dr}k_m,\ \text{where}\ n(r)=\frac{1}{\sqrt{2\pi}(\ln\sigma)r}e^{-\left[\ln(r/r_p)\right]/\left(\sqrt{2}\ln\sigma\right)^2}\ \text{and}\ \sigma=1.5.$$

Xuan, et al. 2003 (spheres with Brownian motion and cluster effect)

$$k_e=k_m+3v_p\frac{k_p-k_m}{2k_m+k_p-v_p(k_p-k_m)}k_m+\frac{1}{2}\rho_p C_{pp}v_p\sqrt{\frac{k_B T}{3\pi\mu_m r_c}}$$

Jang and Choi 2004, (spheres)

$$k_e=(1-v_p)k_m+v_p k_p+3c(r_m/r_p)v_p\left(\frac{k_B T}{3\pi\mu_m v_m l_m}\right)^2\mathrm{Pr}k_m,\ \text{where}\ c\ \text{is an empirical parameter.}$$

Koo and Kleinstreuer 2004 (spheres with Brownian motion effect for CuO based nanofluids)

$$k_e=k_m+3v_p\frac{k_p-k_m}{2k_m+k_p-v_p(k_p-k_m)}k_m+5\times10^4\beta\rho_m C_{pm}v_p\sqrt{\frac{k_B T}{2\rho_p r_p}}\left[(-134.63+1722.3v_p)+(0.4705-6.04v_p)T\right],$$

where the particle motion related empirical parameter $\beta=\begin{cases}0.0137(100v_p)^{-0.8229} & v_p<0.01\\ 0.0011(100v_p)^{-0.7272} & v_p>0.01\end{cases}$

Kumar, et al. 2004 (spheres)

$$k_e=k_m+c\left(\frac{r_m}{r_p}\right)\left(\frac{v_p}{1-v_p}\right)\left(\frac{k_B T}{2\pi\mu_m r_p^2}\right),\ \text{where}\ c\ \text{is an empirical parameter.}$$

Nan, et al. 2004 (carbon nanotubes with contact resistance)

$$k_e=k_m+\frac{1}{3}v_p\frac{k_p}{1+Rk_p/a}$$

Yu and Choi 2004 (ellipsoids with shell)

$$k_e=k_m+\Psi^{-1}v_c\sum_{a,b,c}\frac{k_{ci}-k_m}{(3\Psi^{-1}-1)k_m+k_{ci}-v_c(k_{ci}-k_m)}k_m$$

Prasher, et al. 2005; Prasher, et al. 2006a (spheres with contact resistance and Brownian motion effect)

$$k_e=\left[1+cv_p\left(\frac{9k_B T}{\pi\rho_p v_m^2 r_p}\right)^m\mathrm{Pr}^{0.333}\right]\left\{k_m+3v_p\frac{k_p^R}{2k_m+k_p^R-v_p(k_p^R-k_m)}k_m\right\},\ \text{where}\ c\ \text{and}\ m\ \text{are empirical parameters.}$$

Xue 2005 (carbon nanotubes)

$$k_e=k_m+\frac{\frac{4}{\pi}v_p\frac{k_p-k_m}{\sqrt{k_m k_p}}\arctan\sqrt{\frac{k_p}{k_m}}}{(1-v_p)+\frac{4}{\pi}v_p\frac{k_m}{\sqrt{k_m k_p}}\arctan\sqrt{\frac{k_p}{k_m}}}k_m\qquad k_e=k_m+\frac{2v_p\ln\left(\frac{k_p+k_m}{2k_m}\right)}{(1-v_p)+2v_p\frac{k_m}{k_p-k_m}\ln\left(\frac{k_p+k_m}{2k_m}\right)}k_m$$

Yu and Choi 2005 (spheres with shells, simple cubic arrays)

$$k_e=\left\{1-\sqrt[3]{\frac{3v_c}{4\pi}}\left[2-\frac{\sqrt[3]{16/(9\pi v_c^2)}k_m}{k_1k_2}\ln\frac{(k_2+k_1)\left(k_2-\sqrt[3]{v_r}k_1\right)}{(k_2-k_1)\left(k_2+\sqrt[3]{v_r}k_1\right)}+\frac{\sqrt[3]{16/(9\pi v_c^2)}k_m}{k_3k_4}\ln\frac{k_4+\sqrt[3]{v_r}k_3}{k_4-\sqrt[3]{v_r}k_3}\right]\right\}^{-1}k_m,$$

where $k_1=\sqrt{k_s-k_m}$, $k_2=\sqrt{\sqrt[3]{16/(9\pi v_c^2)}k_m+(k_s-k_m)}$, $k_3=\sqrt{k_p-k_m}$, and $k_4=\sqrt{\sqrt[3]{16/(9\pi v_c^2)}k_m+(k_s-k_m)+\sqrt[3]{v_r^2}(k_p-k_s)}$

Tabella 4: Equazioni analitiche per il calcolo della conduttività termica

Effetti delle nanoparticelle:

- Effetti dello strato di nanoparticelle
  Le molecole liquide racchiuse da una superficie solida formano uno strato costituito da nanoparticelle solide e massa liquida. Questo strato intermedio consente di avere una maggiore conduttività termica rispetto alla sola componente liquida. Sulla base di questa assunzione, l'equazione di Maxwell è stata rielaborata considerando la presenza dello strato solido-liquido e successivamente considerando la composizione di particelle non sferiche e di concentrazioni diverse [6].
- Effetti di nanoparticelle nel moto Browniano
  La dimensione delle particelle influenza il moto Browniano e di conseguenza anche l'incremento della conduttività termica.
  Anche in questo caso, l'equazione di Maxwell è la base di partenza per le successive rielaborazioni.
  Alcuni modelli elaborati tengono in considerazione il raggio di curvatura di nanoparticelle aggregate tra loro, che però deve essere determinato empiricamente.
  Un altro modello dinamico sviluppato considera la convezione indotta da una singola particella Browniana, ma purtroppo questa equazione non è verosimile nella realtà nei nanofluidi e la presenza di un parametro empirico è un ulteriore fattore che limita la predizione della conduttività termica [9] [6].
- Effetti delle nanoparticelle aggregate
  Le nanoparticelle, all'interno dei nanofluidi, aggregandosi tra loro producono un filtro. Questa struttura generata è un fattore positivo per l'incremento della conduttività termica. La dimensione delle aggregazioni di nanoparticelle è un fattore critico, perché, d'altro canto, maggiori sono le loro dimensioni, maggiore è la possibilità che queste si spostano verso la superficie del fluido riducendo gli effetti positivi dell'aggregazione [6].

Viscosità:

- Einstein (1906), è stato il primo a calcolare l'effettiva viscosità di piccole sfere sospese all'interno di un fluido.
  $\mu_e = (1 + 2.5 v_p)\mu_m$
  La viscosità effettiva è lineare rispetto alla viscosità del fluido base contenente piccole particelle sospese.
  L'equazione formulata da Einstein venne progressivamente estesa considerando piccole concentrazioni di particelle e particelle non sferiche.
  I parametri per l'utilizzo di tali equazioni sono limitati (temperatura, concentrazione, ecc.). Tuttora, gli esperimenti dimostrano che l'andamento dell'effettiva viscosità dei nanofluidi è superiore a quella teorica predetta. Per supplire a questa mancanza, sono state introdotte equazioni adattate a specifiche combinazioni particelle-fluidi base, ma purtroppo queste non corrispondono con l'equazione di Einstein per concentrazioni di volume molto basse, pertanto non hanno solide basi fisiche.

Modelli di correlazione:

Questi modelli sono basati sulla correlazione dei parametri e consentono di avere una miglior comprensione dei meccanismi coinvolti nel trasferimento di calore nei nanofluidi. La correlazione dipende da un parametro costante per cui il valore è determinato sperimentalmente, osservando l'incremento della conduttività termica. Questo parametro può essere lo spessore del liquido, la conduttività dello strato di fluido, una funzione che descrive le proprietà del fluido e l'interazione

della particella.
Tali modelli sono adeguati nel sistema in cui sono stati sviluppati, ma risultano imprecisi se le condizioni iniziali variano.

Modelli simulati:
Un altro approccio teorico per lo studio della conduttività termica è l'uso della simulazione. Sono presenti le simulazioni Browniane, le simulazioni numeriche ad elementi finiti, il metodo numerico di Boltzmann per ricercare la distribuzione delle particelle in un nanofluido e la dinamica molecolare classica che studia la resistenza termica tra i nanotubi in carbonio e il fluido [6].

CAPITOLO 2

Stato dell'arte

RETI NEURALI

Nomenclatura:
N = numero di neuroni
P = numero di esempi
a_i = attivazione dell'i-esimo neurone
E = errore quadratico medio
o_i = uscita presunta dell'i-esimo neurone
t_j = uscita reale dell'i-esimo neurone
w_{ij} = peso della connessione tra l'i-esimo neurone e il j-esimo neurone
δ_k = valore di delta
η = valore costante e positivo, chiamato fattore di apprendimento
θ = esempio di training set

2.1 Introduzione

La rete neurale artificiale (*Artificial Neural Network* o *ANN*, o anche *NNT*) è un *modello matematico* prodotto dall'intelligenza artificiale, ispirato alla struttura ed al funzionamento del sistema nervoso umano.
Le reti neurali risultano utili laddove si è in possesso di una discreta quantità di dati, ma non è chiara la relazione che lega gli input con gli output. Le ANNs, tramite la computazione parallela e iterativa dei dati d'ingresso, producono le proprie regole che mettono in relazione l'ingresso e l'uscita, fornendo in output una previsione.
L'inserimento di nuovi dati d'input permette alle reti neurali di aggiornare le regole che connettono i dati memorizzati all'interno della rete, raffinando la relazione tra ingresso ed uscita. Questa procedura viene chiamata apprendimento o addestramento della rete.

2.2 Vantaggi, svantaggi e campi di applicazione

L'intelligenza artificiale (IA), supportata dall'utilizzo di elaboratori, consente la ricerca di soluzioni a problemi non strutturati o problemi troppo complessi non risolvibili con altri metodi.

Le condizioni in cui si rivela più adatto l'utilizzo di reti neurali rispetto ai metodi tradizionali sono [10]:
- grande numero di variabili che devono essere tenute in considerazione contemporaneamente;
- set di esempi il più possibile ampio e preciso, soprattutto per trovare relazioni complesse;
- la struttura delle relazioni del modello sottostante il dominio non è chiaramente definita;
- le variabili o il modello posto in esame non sono stabili nel tempo a causa di mutamenti nell'ambiente;
- i costi associati ai dati sperimentali o a previsioni errate sono alti;

– stabilità dell'output rispetto a valori d'input: incompleti, con rumore, non ben noti, con un grado di errore o di variazione.

I risultati ottenuti mediante le reti neurali sono buoni, ma non sono spiegabili in modo chiaro, si usa perciò considerare la rete neurale come una "*black box*".
Gli svantaggi che si possono riscontrare sono:
– in applicazioni complesse, specialmente con set di esempi molto vasti, l'addestramento può richiedere molto tempo di elaborazione da parte di un computer moderno;
– non esistono teoremi o modelli che stabiliscano la struttura di una "rete perfetta", ma la realizzazione di una struttura adatta al problema da risolvere tuttora dipende molto dal creatore;
– i valori di output hanno un margine di errore;
– non sempre esiste una rete che risolve il problema, perché non sempre esiste un algoritmo di apprendimento che converge dando un output della rete con basso errore.

I campi applicativi delle ANNs sono molteplici:
– pianificazione autonoma: agenti remoti che sono in grado di raggiungere un obiettivo pianificando autonomamente procedure e azioni;
– robotica: le applicazioni al mondo reale sono numerose e sempre più affidabili e consolidate. Le applicazioni vanno da quelle spaziali e quelle nei fondali marini, a quelle in ambienti ostili e/o insicuri per l'uomo, alla robotica di servizio o domestica;
– diagnosi: in questo campo ci sono programmi in grado di effettuare diagnosi in specifiche aree della medicina, oppure d'individuare guasti in sistemi fisici complessi;
– riconoscimento di pattern e voce: in questo settore applicativo l'utilizzo delle reti neurali è molto diffuso e il livello di sofisticazione dei programmi è tale da avere risultati, in certi settori, superiori a quelli degli esseri umani.

2.3 Rete neurale biologica

L'idea di una rete neurale artificiale formata da unità d'informazione collegate tra loro, con collegamenti simili a quelle dei neuroni, viene proposta per la prima volta da *McCulloc e Pitts (1943)* con l'esplicito intento di simulare il funzionamento del cervello.

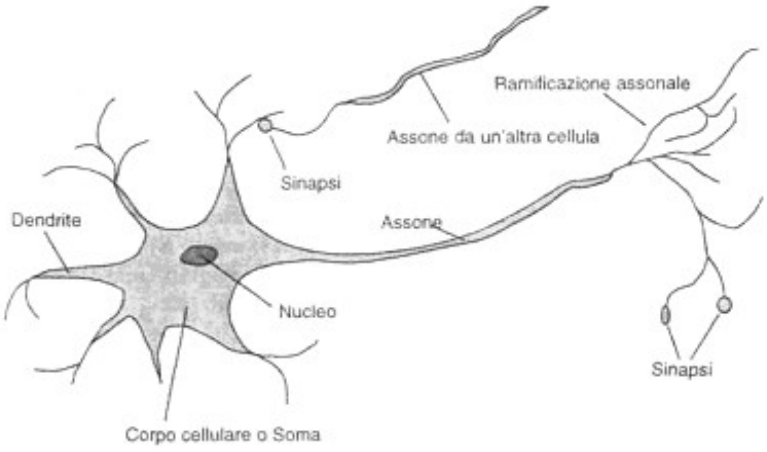

Figura 13: Rappresentazione del neurone biologico

Un *neurone* del cervello è formato da un *corpo* cellulare, chiamato *soma*, e da molti prolungamenti ramificati, detti *dendriti*, attraverso i quali il neurone riceve segnali elettrici da altri neuroni.
Ogni neurone ha anche un prolungamento filamentoso chiamato *assone* che alla sua estremità si ramifica formando piccole protuberanze, dette *sinapsi*, attraverso le quali i segnali elettrici vengono trasmessi ad altre cellule (ad esempio ai dendriti di altri neuroni).

Un neurone si "attiva", cioè trasmette un impulso elettrico lungo il suo assone, se la somma delle correnti che arriva presso la base dell'assone tramite i suoi dendriti supera una certa *soglia di attivazione*, rilasciando una certa quantità di sostanze chimiche, dette *neurotrasmettitori*, che generano un impulso (detto spike) di corrente di breve durata (2-5 millisecondi) che procede verso l'assone propagandosi verso altri neuroni. Se invece la somma non supera la soglia di attivazione, il neurone resta in stato di quiete e non trasmette alcun impulso elettrico.

2.4 Rete neurale artificiale

In una rete neurale i neuroni sono rappresentati come delle *unità* poste in parallelo tra loro, in grado di elaborare i segnali che ricevono in ingresso da altri neuroni. Le sinapsi sono rappresentate come delle *connessioni* tra le unità. Il segnale elettrico che viaggia lungo l'assone è rappresentato con un numero. La conduttività delle sinapsi è rappresentata da un numero detto *peso della connessione*. Prima che il segnale venga trasmesso al neurone successivo, il peso della connessione viene moltiplicato per un coefficiente variabile che dipende da una funzione di attivazione. Il risultato delle moltiplicazioni viene sommato e se la somma supera un valore numerico (*soglia di attivazione*) il neurone si attiva attivando la sua uscita. I pesi delle connessioni non sono prefissati, ma vengono fatti evolvere nel tempo sulla base di opportune regole, in modo tale che la rete "apprenda" per svolgere il compito che le è stato richiesto. Per lo più, all'inizio i pesi delle connessioni sono attribuiti in maniera casuale. Dopo di che le reti vengono "addestrate": i pesi vengono modificati sulla base delle regole di apprendimento, in modo che i risultati approssimino via via quelli desiderati. La "conoscenza" di una rete è quindi immagazzinata nei pesi delle connessioni.

2.4.1 Modello matematico

Il neurone di McCulloch e Pitts è un'idealizzazione semplificata del neurone biologico: dal punto di vista matematico, un neurone può essere visto come una struttura caratterizzata da un insieme d'inputs, un insieme di pesi e un output [11].

Nel modello di neurone rappresentato in Figura 14, ad ogni ingresso x_1... x_n, il quale rappresenta un dendrite biologico, viene associato un peso della connessione w_1...w_n, che rappresenta l'efficacia ed il segno della sinapsi.
Se $w_i > 0$ il canale è detto *eccitatorio*, invece se $w_i < 0$ il canale è detto *inibitorio*.
L'uscita y, che rappresenta l'uscita dell'assone, è calcolata applicando la *funzione di attivazione* alla somma pesata degli ingressi, e nella maggior parte dei casi assume valori compresi tra 0 e 1 o -1 e 1. Questa operazione emula il comportamento dei neuroni biologici i quali, sulla base dell'integrazione dei segnali post-sinaptici, stabiliscono se inviare o meno un impulso nervoso lungo i loro assoni.

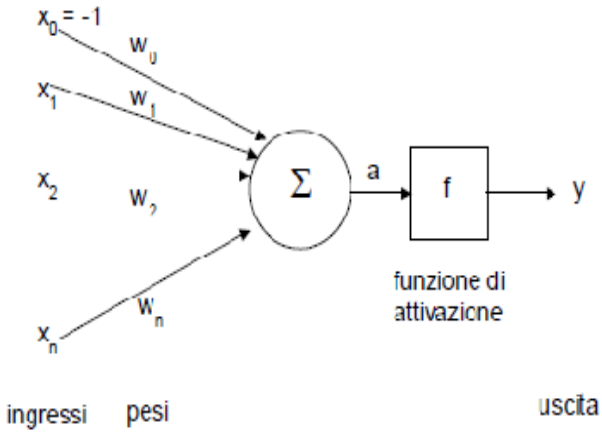

Figura 14: Rappresentazione del modello matematico di un neurone

L'ingresso x_0, denominato *bias*, ha l'effetto, a seconda della sua positività o negatività, di aumentare o diminuire il livello di soglia del neurone.
Le principali funzioni di attivazione che vengono utilizzate in un neurone artificiale sono del tipo: Lineare, Lineare con Saturazione, Hard Limiter, Sigmoide e tangente iperbolica.

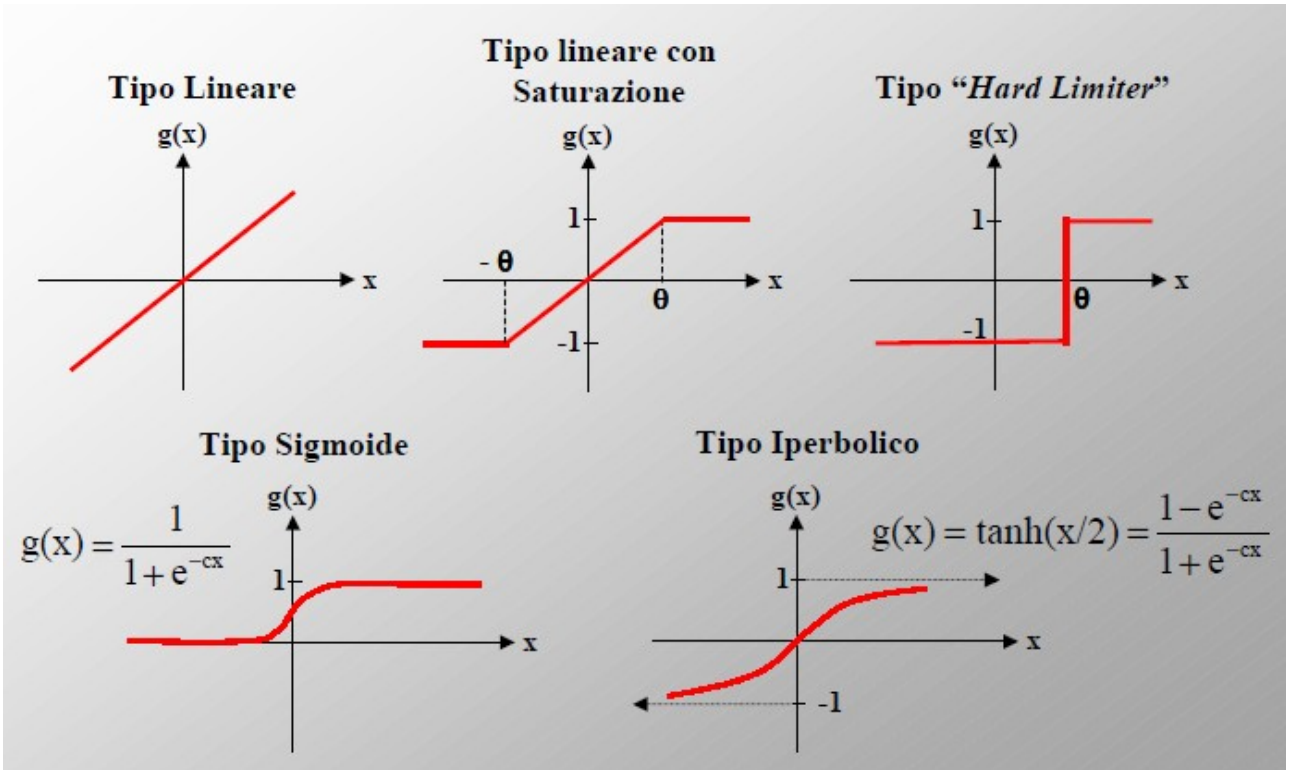

Figura 15: Funzioni di attivazione

Una rete di neuroni artificiali di McCulloch e Pitts può apprendere dall'esperienza se si consente che i pesi delle connessioni possano essere modificati, sulla base dei dati di esperienza, con l'obiettivo di riuscire a generare, dopo un periodo di apprendimento, le risposte desiderate a date sollecitazioni.

Prima dell'addestramento, i pesi vengono inizializzati con valori casuali e si cominciano a presentare, uno alla volta, gli input dell'insieme di addestramento (*training set*). Per ogni esempio presentato si calcola l'*errore* commesso dalla rete, cioè la differenza tra l'uscita desiderata e l'uscita effettiva della rete. L'errore viene poi utilizzato per modificare i pesi.

Il processo viene ripetuto presentando alla rete, in ordine casuale, tutti gli esempi del training set, lasciandola evolvere finché l'errore commesso su tutto il training set (oppure l'errore medio sul training set) non risulti inferiore ad una soglia prestabilita.

2.4.2 Classificazione

In letteratura esistono vari modelli di reti neurali che possono essere raggruppati da alcune caratteristiche principali comuni [12]:

- l'utilizzo che ne viene fatto (memorie associative, approssimazioni di funzioni matematiche complesse, classificatori)
- le aree di applicazione (medicina, robotica, economia, ecc.)
- le modalità di apprendimento (supervisionato / non supervisionato)
- l'algoritmo di apprendimento (feedforward o feedback)
- l'architettura dei collegamenti tra processi neurali (semplici o complessi)

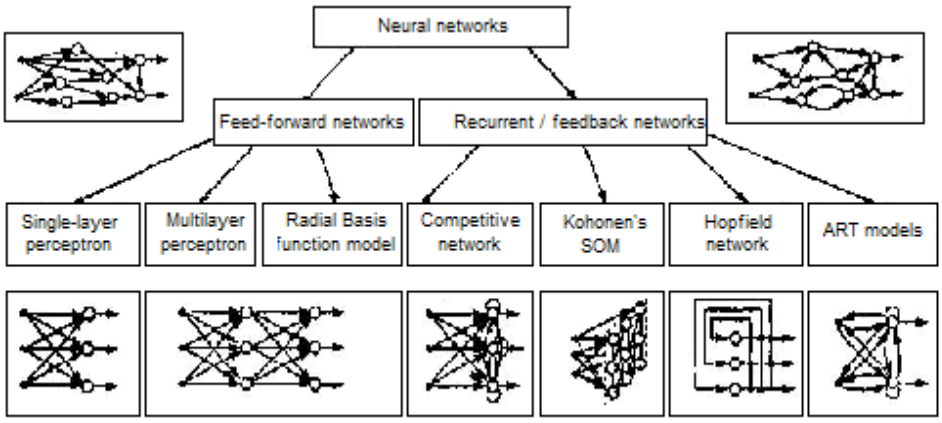

Figura 16: Classificazione delle reti neurali artificiali

In Figura 16 viene mostrata una importante classificazione per algoritmo: feed-forward e feed-back.
La rete neurale ha una architettura feed-back se sono previste connessioni tra i neuroni di uscita e neuroni di ingresso, viceversa si ha un'architettura detta feed-forward.

Tra le reti di tipo *feed-back*, sono presenti quelle proposte da Boltzmann e Hopfield.
Hopfield, nel 1982, propose una rete neurale artificiale caratterizzata da connessioni simmetriche e cicli di feedback tra uscite e ingressi {1}.
Un altro tipo di apprendimento è quello implementato dalla macchina di Boltzmann che utilizza un algoritmo di apprendimento stocastico basato sulle proprietà della distribuzione di probabilità di Boltzmann, la stessa utilizzata in termodinamica (1984) {2} .

Nella classificazione di tipo feed-forward, vi è il single-layer (Perceptron) che è una rete neurale costituita da un singolo strato in grado di classificare opportune categorie di stimoli attraverso una forma di apprendimento supervisionato.
A partire dal neurone semplice, si sono sviluppati modelli più complessi che hanno portato all'inserimento di ulteriori strati computazionali che permettono di avere più output tra loro indipendenti gli uni dagli altri.
Il *Multi-Layer Perceptron* (MLP) è una rete feed-forward formata da uno o più strati di unità interne (*hidden layer*) i quali si trovano tra le unità d'ingresso dove vengono forniti i dati (strato formato da neuroni d'input), e quelle di uscita che forniscono la risposta (strato formato da neuroni di output). I neuroni che compongono lo strato d'ingresso e gli strati nascosti sono connessi con tutti i livelli successivi, senza retroazioni, e rielaborano i dati forniti nello strato d'ingresso.

Un'altra importante classificazione è quella che riguarda il tipo di apprendimento, che si suddivide in *supervisionato* e non *supervisionato*.

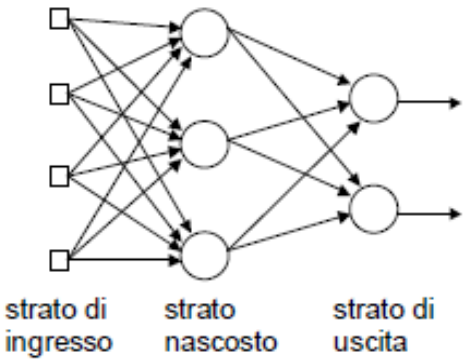

Figura 17: Struttura del multilayer perceptron

Entrambi seguono la legge di apprendimento di Hebb (1949), secondo la quale la sinapsi che connette due neuroni incrementa la propria connettività se i due neuroni sono contemporaneamente attivati dallo stimolo in ingresso. La variazione di connettività, ovvero di peso per quella connessione, è quindi calcolata come funzione del prodotto dei due valori di attivazione associati ai rispettivi neuroni.

Nell'apprendimento supervisionato, alla rete viene presentato un training set preparato da un supervisore esterno, e composto da molte coppie significative di valori (input e output atteso):

– La rete riceve l'input e calcola il suo corrispondente output;
– Per un certo input, l'errore è dato dalla differenza tra l'output della rete e l'output atteso; serve a supervisionare l'apprendimento per far capire alla rete quanto si sbaglia nel calcolare tale output;
– La rete modifica i pesi in base all'errore cercando di minimizzarlo e commetterà sempre meno errori;

I modelli che usano questo apprendimento sono il perceptron, multi-layer perceptron e il radial basis function.

Invece, nell'apprendimento non supervisionato, alla rete vengono presentati solo i valori d'input e la rete li divide autonomamente in gruppi usando misure di similarità, senza usare confronti con output noti e cercando di mettere input simili nello stesso gruppo. E' un apprendimento autonomo e non c'è controllo esterno sull'errore; è adatto per ottimizzare risorse e se non si conoscono a priori i gruppi in cui dividere gli input.
Fra i tipi di reti neurali che adottano un tipo di apprendimento non supervisionato ci sono le Hopfield e le Self Organizing Map (Kohonen), che effettuano un clustering non lineare sui dati così da ridurre la complessità n-dimensionale del vettore in ingresso.

2.4.3 Feed Forward e backpropagation

L'algoritmo di *back-propagation* (Rumelhart, Hinton e Williams 1986) è il più diffuso per la sua semplicità, efficacia e potenza. Ha l'obiettivo di minimizzare l'errore quadratico medio fra l'output corrente e quello desiderato.
Essendo di tipo feed forward, il flusso di attivazione procede sempre in avanti a partire dalle unità d'input verso le unità di output. Una volta confrontato l'output ottenuto con quello atteso, si effettua la "propagazione all'indietro" (back propagation) del gradiente dell'errore tramite la modifica dei pesi sinaptici .

Il punto cruciale della back-propagation sta nel calcolo del "*termine delta*" per le unità nascoste, ovvero l'errore delle unità di output trasportato all'indietro dalle connessioni sinaptiche e moltiplicato per la derivata prima della funzione di output di ciascuna unità nascosta.
Questo metodo è ricorsivo e può essere applicato a un qualsiasi numero di strati di unità nascoste.

L'algoritmo di retropropagazione dell'errore si svolge nel modo che segue:
- Si inizializzano i pesi con valori casuali (in genere con valori non troppo elevati);
- Si presenta un ingresso e si calcolano le uscite di tutti i neuroni della rete.
- Si confrontano i dati ottenuti dalla rete in output, con quelli che dovevano risultare dal training set; si calcola l'errore e la variazione di pesi di ciascun neurone dell'ultimo strato.
- Si retropropagano gli errori per calcolare quelli dei neuroni intermedi.
- Si effettua la modifica di tutti i pesi della rete in base alle variazioni calcolate.
- Si calcola l'errore medio sul training set (oppure l'errore globale). Se l'errore è al di sotto di una soglia prefissata, l'algoritmo termina (in quanto si è raggiunta la convergenza), altrimenti si ripete un intero ciclo di addestramento.

L'algoritmo di back-propagation elabora le informazioni in modo tale che la rete diminuisca l'errore globale durante le iterazioni di apprendimento, tuttavia questo non garantisce che il minimo globale venga raggiunto. La presenza delle unità nascoste e la non-linearità della funzione di output fa sì che l'andamento dell'errore all'interno dello spazio multidimensionale dei pesi sinaptici sia molto complesso e abbia molti minimi locali.
L'algoritmo di backpropagation può dunque arrestarsi in corrispondenza di un minimo locale, fornendo una soluzione subottima.

Di norma l'errore decresce sempre sul training set (ovvero migliora la capacità di rappresentare la relazione input-output tra i dati forniti) perché la rete sta apprendendo, mentre sul validation set (che misura le capacità predittive), da un certo valore in poi, può crescere a causa dell'overfitting. In questo caso, la rete si comporta come una memoria della distribuzione dei campioni di training perdendo la sua capacità di generalizzazione sui campioni sconosciuti: impara il training set, ma non il modello statistico.
Ciò risulta in un'alta accuratezza di classificazione per i campioni di training e una bassa accuratezza di classificazione per i campioni sconosciuti.
Non esistono criteri d'arresto ben definiti, questi che seguono sono i più utilizzati.
L'algoritmo di back-propagation può essere interrotto quando:
- è nelle vicinanze di un minimo locale;
- la percentuale di variazione dell'errore quadrato medio tra due epoche consecutive è sufficientemente piccola;
- la capacità di generalizzazione è adeguata.

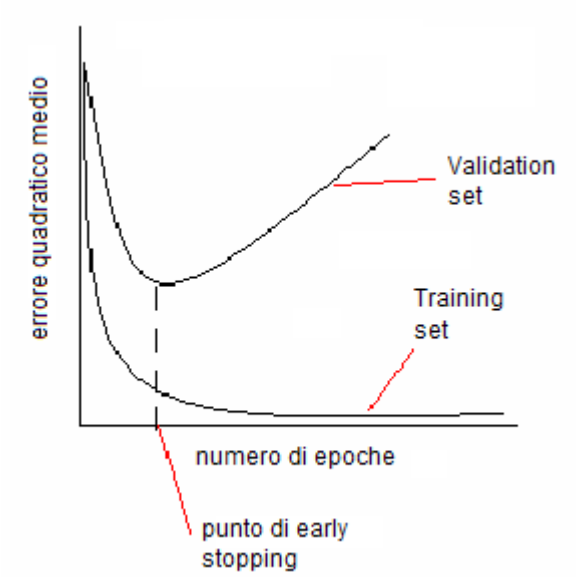

Grafico 1 : Andamento degli errori di "Validation set" e "Training set"

2.4.3.1 Backpropagation, approccio matematico

L'algoritmo di backpropagation utilizzato è un apprendimento di tipo *batch*, perché l'insieme dei dati per l'addestramento è disponibile prima che l'addestramento abbia inizio.

L'algoritmo di apprendimento backpropagation per reti feedforward a più strati [13], ha l'obiettivo di minimizzare, modificando i parametri liberi (pesi delle connessioni), la funzione d'errore E:

$$E=\sum_{\theta} E^{\theta}=\frac{1}{2}\sum_{\theta}\sum_{j}(t_j^{\theta}-o_j^{\theta})^2$$

L'errore quadratico medio viene calcolato tramite sommatorie estese di tutti gli esempi di training set e di tutte le uscite. Ogni esempio contribuisce all'errore globale E, con un termine E^{θ}, calcolato effettuando la differenza tra l'output atteso e l'output presunto:

$$E^{\theta}=\frac{1}{2}\sum_{j}(t_j^{\theta}-o_j^{\theta})^2$$

L'aggiustamento dei pesi viene eseguito in base all'errore E^{θ} calcolato per ogni pattern presentato alla rete.

Essendo l'algoritmo basato sulla tecnica della discesa del gradiente, dopo il calcolo di E^{θ}, ogni peso delle connessioni viene aggiornato in verso opposto al gradiente:

$$w_{ij}=w_{ij}-\eta\frac{\partial E^{\theta}}{\partial w_{ij}}$$

Si consideri il peso w_{ij} della connessione tra l'uscita del neurone i e l'ingresso del neurone j:

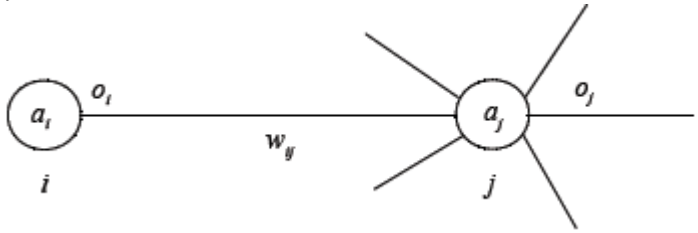

Figura 18: Rappresentazione della connessione tra due neuroni

Il gradiente dipende dal peso della connessione w_{ij} solo attraverso a_j, perciò viene calcolato tramite:

$$\frac{\partial E^{\theta}}{\partial w_{ij}} = \frac{\partial E^{\theta}}{\partial a_j^{\theta}} \frac{\partial a_j^{\theta}}{\partial w_{ij}}$$

Siccome il valore dell'attivazione è determinato da

$$a_j^{\theta} = \sum_k w_{kj} o_k^{\theta}$$

si ottiene

$$\frac{\partial a_j^{\theta}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}} \sum_k w_{kj} o_k^{\theta} = o_j^{\theta}$$

Ponendo

$$\delta_j = \frac{\partial E^{\theta}}{\partial a_j^{\theta}}$$

il calcolo del valore δ_j dipende dalla posizione del neurone all'interno della struttura della rete neurale.

Se il neurone j è una unità nascosta, non si ha una specifica risposta desiderata. Il valore di errore, dunque, deve essere determinato ricorsivamente dal segnale di errore di tutti i neuroni ai quali questo neurone nascosto è connesso:

$$\delta_j = \frac{\partial E^{\theta}}{\partial a_j^{\theta}} = \sum_k \frac{\partial E^{\theta}}{\partial a_k^{\theta}} \frac{\partial a_k^{\theta}}{\partial a_j^{\theta}} = \sum_k \delta_k \frac{\partial}{\partial a_j^{\theta}} \sum_m w_{mk} o_m^{\theta} = \sum_k \delta_k w_{jk} \frac{\partial o_j^{\theta}}{\partial a_j^{\theta}}$$

dove la sommatoria su k è estesa ai neuroni il cui ingresso è connesso all'uscita del neurone j, mentre la sommatoria su m è estesa ai neuroni la cui uscita è connessa all'ingresso del neurone k.

Se invece j è un neurone di uscita, δ_j può essere calcolato come segue:

$$\delta_j = \frac{\partial E^{\theta}}{\partial a_j^{\theta}} = \frac{\partial E^{\theta}}{\partial o_j^{\theta}} \frac{\partial o_j^{\theta}}{\partial a_j^{\theta}}$$

La variazione di peso sinaptico tra il neurone i e j relativo all'n-esimo esempio del training set viene determinata tramite:

$$w_{ij} = w_{ij} - \eta \delta_j o_j^{\theta}$$

In sintesi, una volta calcolata l'uscita della rete per un dato esempio, la derivata $\partial E^{e}/\partial w_{ij}$ può essere calcolata direttamente se w_{ij} è il peso di una connessione a un neurone di uscita; in caso contrario, richiede la conoscenza dei termini δ_k dello strato successivo.

2.5 Scelta della struttura

La struttura di una rete neurale influisce sulle prestazioni.

Se il numero di strati nascosti o di neuroni è troppo basso, la rete sarà incapace di approssimare con precisione la funzione desiderata.

Se invece la rete è troppo grande, si potrebbe facilmente avere overfitting.

E' stato dimostrato che una rete neurale con un numero sufficiente di unità nascoste si può approssimare con precisione arbitraria:

– qualsiasi funzione continua, con un solo strato nascosto;
– qualsiasi funzione, anche discontinua, con due strati nascosti;

tuttavia non è possibile determinare a priori il numero necessario di unità nascoste, dato che la funzione da approssimare non è mai nota a priori.

In generale, non esistono tecniche standard per la determinazione della struttura di una rete

neurale, ma esistono diverse tecniche di tipo euristico.
Possibili tecniche possono essere :

- calcolo della VC-dim[1] dell'architettura per stabilire quanti esempi di un dato dominio sono necessari (in rapporto al numero dei pesi) per ottenere buone capacità di apprendimento e generalizzazione;
- utilizzo della cross-validation e di tecniche trial-and-error;
- utilizzo di algoritmi genetici di selezione nello spazio delle architetture;
- algoritmi di accrescimento o growing
  es. algoritmi Tiling, Tower, Pyramid e Cascade Cor-relation;
- algoritmi di potatura o pruning
  es. algoritmi Optimal Brain Damage e Optimal Brain Surgeon.

1 Vapnik–Chervonenkis dimension: è la misura della capacità di un algoritmo di classificazione statistica

CAPITOLO 3

Soluzioni proposte e implementazioni

SVILUPPO DI UNA SISTEMA A RETI NEURALI

3.1 Introduzione

Lo sviluppo software della rete neurale e la configurazione dei suoi parametri per ottenere la predizione della conduttività termica dei nanofluidi al, è argomento di trattazione in questo capitolo.
La buona riuscita di questa tecnica informatica, consente:

- l'utilizzo di un minor numero di dati sperimentali che possono essere costosi o lunghi da ottenere;
- lo studio e analisi di dati soggetti ad errore;
- la predizione di un output, senza la conoscenza della relazione che lega ingresso ed uscita.

3.2 Raccolta dei dati sperimentali

I nanofluidi Al_2O_3/Acqua e TiO_2/Acqua sono stati forniti dalla Nanostructured & AmorphousMaterials Inc. USA [14].
Al_2O_3 è una struttura alpha, con una elevato grado di purezza superiore al 99,9%; il diametro medio delle particelle dichiarato dal fornitore è di 30 ± 10 nm e la frazione di massa delle nanoparticelle nella miscela è di circa il 15%.
Il TiO_2 è una struttura di tipo rutilo, con una purezza superiore al 99,9%; il diametro medio della particelle dichiarato è di 30-50 ± 10 nm e la concentrazione di volume delle nanoparticelle è di circa il 25%.
Entrambi i nanofluidi sono stati preparati con una tecnica two step (produzione delle particelle in polvere e successiva dispersione nel fluido base) e non comprendono additivi tensioattivi per ridurre l'agglomerazione delle particelle, perciò è stato necessario un trattamento specifico per garantire la dispersione e ridurre l'agglomerazione.
Per verificare l'efficienza di dispersione dei nanocomponenti dei differenti trattamenti meccanici, ogni nanofluido è stato suddiviso in due parti: il primo è stato soggetto solamente ad una agitazione meccanica per 1 ora (stirring), il secondo è stato sonicato a 25 kHz per 48 ore (sonicated).
Il processo di sonicazione è stato effettuato utilizzando un sonicatore Hielscher UP200S instrumentato con un sonotrodo di 14 millimetri S14D.
I nanofluidi sono stati successivamente analizzati dallo strumento di misurazione Malvern Zetasizer Nano ZS per misurare la distribuzione delle dimensioni dei cluster.
In Figura 19 e Figura 20 sono mostrati i risultati delle misurazioni e si può notare che il trattamento a ultrasuoni mostra una migliore efficienza di dispersione rispetto alla semplice agitazione meccanica sia per Al_2O_3 e TiO_2.

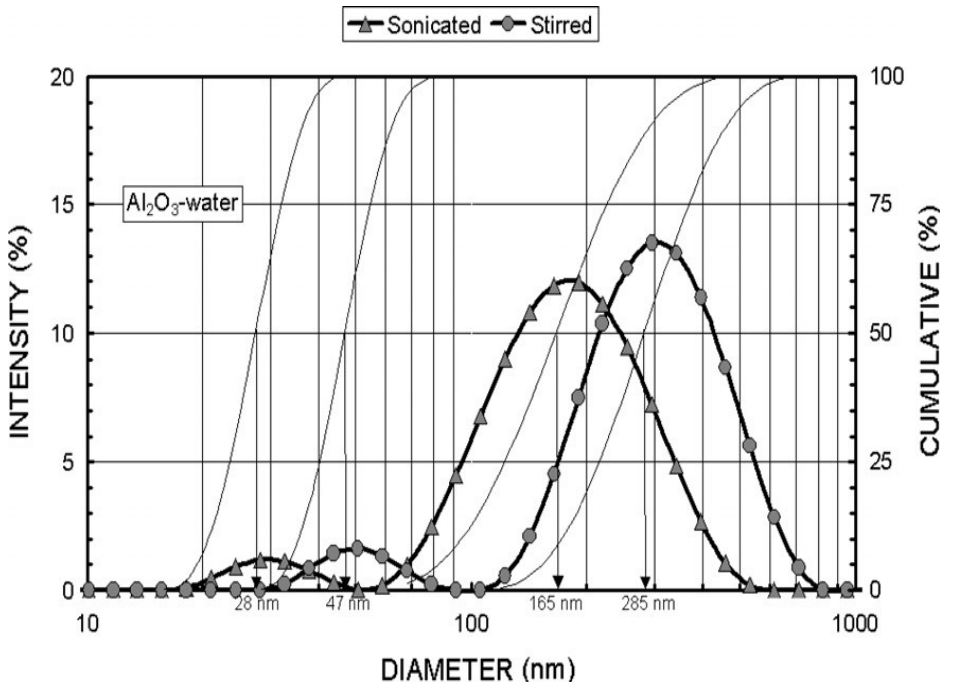

Figura 19: Misurazione della distribuzione dei cluster nel nanofluido Al2O3

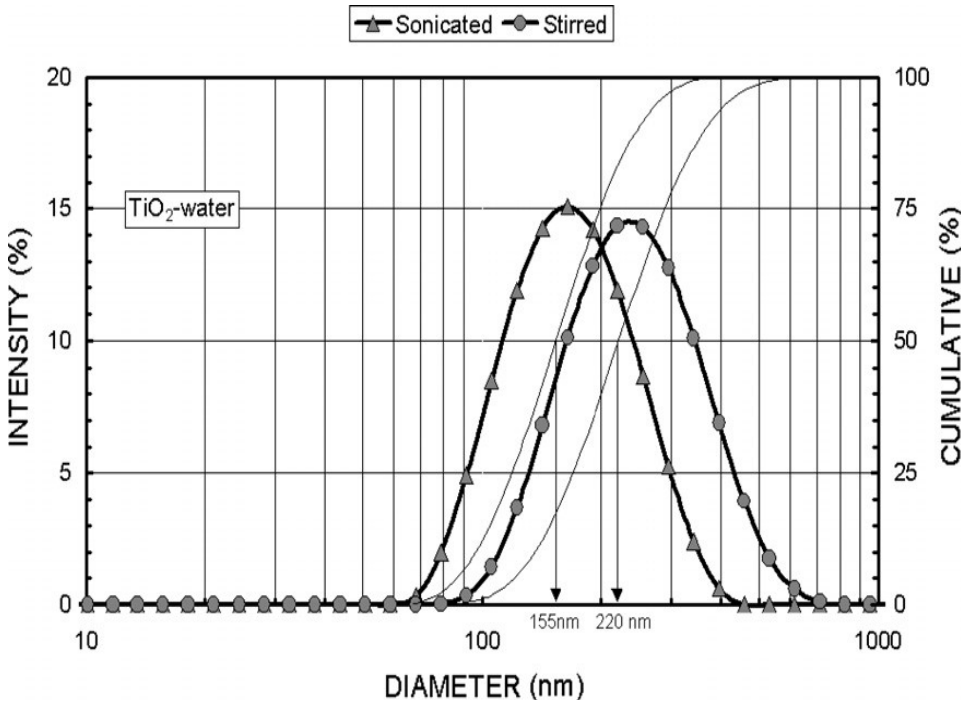

Figura 20: Misurazione della distribuzione dei cluster nel nanofluido TiO_2

I campioni di nanofluidi con concentrazioni differenti sono stati ottenuti diluendo con acqua distillata i nanofluidi stirred o sonicated: per Al_2O_3-Acqua sono state ottenute tre diverse concentrazioni di volume (1, 2, 4%), mentre per il TiO_2-Acqua sono state quattro le concentrazioni di volume ottenute (1, 2, 4, 6%).

La concentrazione del volume di particelle è stata verificata da un densimetro digitale Anton Paar (incertezza (=2k) entro ± 10-4g / cm3); la misurazione della conduttività termica, invece, è stata effettuata utilizzando un Transient Hot Disk TPS 2500S per mezzo di una sonda 7577 con

un'incertezza massima(k= 2) inferiore a ± 5,0% della lettura.
Il campione di nanofluido da misurare è messo in una cella chiusa posta in un bagno a temperatura costante (uniformità 0.01 K, stabilità 0.01 K). L'operazione di misurazione viene effettuata dalla sonda Hot Disk che viene immersa verticalmente nel fluido, che circonda completamente il sensore, così il calore sviluppato dalla sonda Hot Disk può liberamente diffondersi in tutte le direzioni. Per evitare gli effetti naturali della convezione, sono state applicate basse potenze termiche (0.025–0.04 W) e brevi tempi di misurazione (3–4 s).
Ogni misura della conducibilità termica è stata ripetuta 10 volte e il valore medio di tutti i dati misurati è stato calcolato considerando la deviazione standard.
Nelle figure Figura 21, Figura 22, Figura 23 e Figura 24 sono graficati i valori delle conduttività termiche dei campioni misurati.

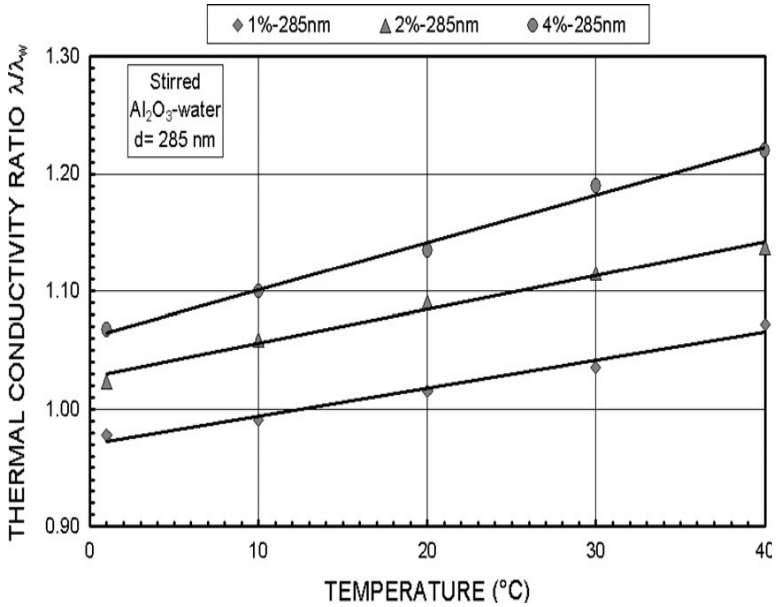

Figura 21: Rapporto tra la conduttività termica tra uno stirred Al_2O_3–Acqua nanofluido (d = 285 nm) e la temperatura e la concentrazione di volume della particella

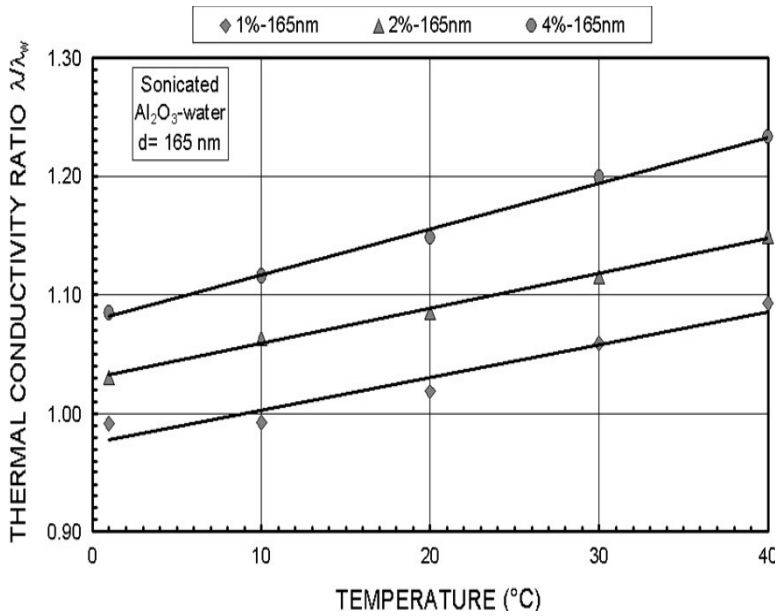

Figura 22: Rapporto tra la conduttività termica tra uno sonicated Al_2O_3–Acqua nanofluido (d = 165 nm) e la temperatura e la concentrazione di volume della particella

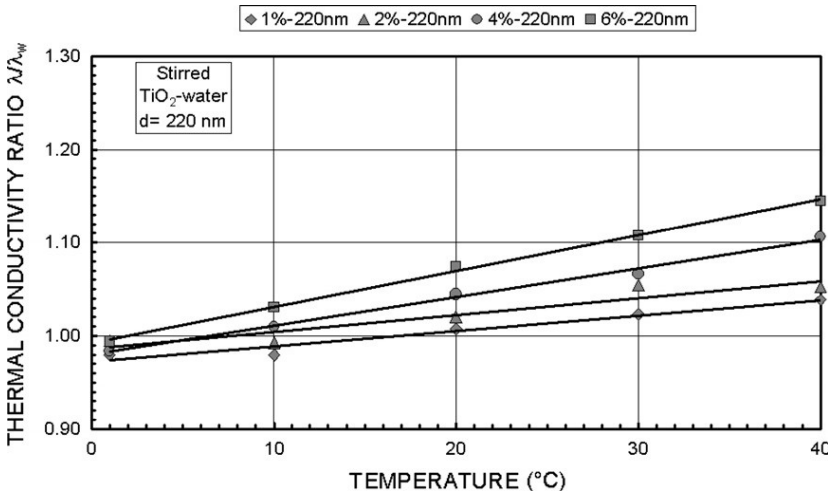

Figura 23: Rapporto tra la conduttività termica tra uno stirred TiO_2–Acqua nanofluido (d = 220 nm) e la temperatura e la concentrazione di volume della particella

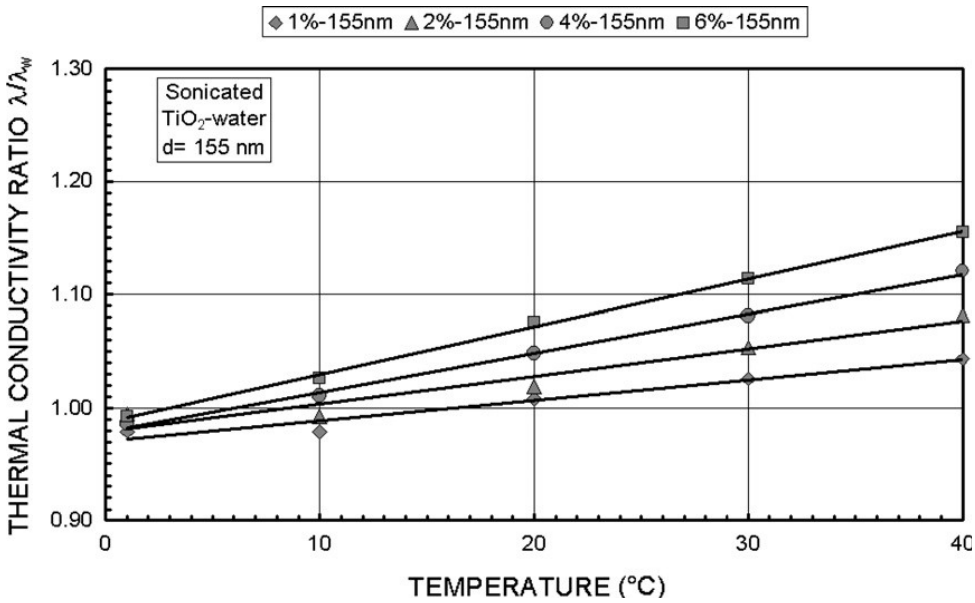

Figura 24: Rapporto tra la conduttività termica tra uno sonicated TiO_2–Acqua nanofluido (d = 155 nm) e la temperatura e la concentrazione di volume della particella

3.3 Libreria Shark

Lo sviluppo dello studio di applicazione di reti neurali artificiali per la modellizzazione delle proprietà dei nanofluidi è basato sull'utilizzo della libreria Shark { 3 }.
Shark è una libreria software, in linguaggio C++ ed orientata ad oggetti, che viene utilizzata per lo sviluppo di "learning machine" e algoritmi di ottimizzazione.
La libreria comprende quattro moduli principali:

- ReClaM: framework per la regressione e la classificazione, include metodi di apprendimento tramite reti neurali e kernel;
- AELib: framework utilizzato per l'ottimizzazione di problemi discreti e continui;
- MOO-EALib: estensione della AELib per una multi-ottimizzazione;
- Fuzzy: framework per l'implementazione della logica Fuzzy;

il modulo ReClaM è quindi quello da noi utilizzato per lo sviluppo delle reti neurali.
Il suo impiego ci consente di ottenere alte performance di velocità, grazie al linguaggio C++, e flessibilità, permettendoci di creare neural networks con diverse tipologie di rete, differenti metodi per la valutazione di errori e diversi algoritmi di apprendimento.

3.3.1 ReClaM

L'architettura del modulo ReClaM è progettata per la soluzione di problemi generici di ottimizzazione, che compaiono nella formazione di modelli parametrici. Il modello, il problema di ottimizzazione e la strategia di soluzione sono le tre classi astratte che compongono la struttura di ReClaM, e prendono il nome di: Model, ErrorFunction e Optimizer.

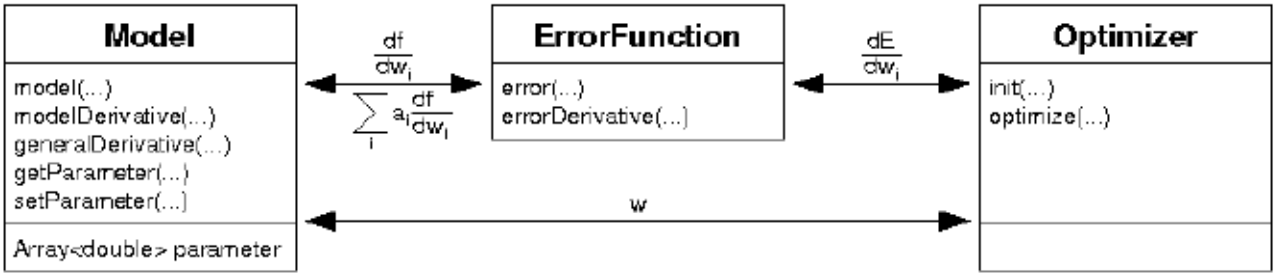

Figura 25: Architettura del modello ReClaM

Model

La classe Model ha un duplice scopo: definire una famiglia parametrizzata di funzioni, ad esempio una funzione lineare o una rete neurale feed forward, e fornire un'interfaccia standardizzata per l'accesso e la manipolazione di questi parametri. Nel contesto di apprendimento della rete, come nel nostro caso, l'oggetto Model calcola la previsione dell'output sulla base del modello selezionato.

Le funzioni appartenenti alla classe Models che sono state utilizzate in questo programma sono initWeights e model.

InitWeights inizializza, con numeri random compresi tra un valore minimo e un valore massimo, i pesi delle connessioni all'interno delle rete.

La funzione *model* utilizza i campioni di input per predire i valori di uscita e memorizzarli in un vettore di output. I valori di ingresso vengono propagati in avanti attraverso la rete per tutti i campioni in modo sequenziale. Dopo ogni propagazione, il risultato dell'output del neurone viene memorizzato nel vettore di output.

Error Function

La classe ErrorFunction definisce un'interfaccia generale per il calcolo di varie misure di errore del modello utilizzato, confrontando le predizioni (output) fornite dal modello con le misure sperimentali.

Durante la fase di apprendimento, l'errore viene utilizzato per ottimizzare i parametri del modello, al fine di aumentare l'efficacia di predizione. L'errore calcolato può essere, ad esempio, l'errore quadratico medio in un problema di regressione se si utilizzano le reti di feedforward, oppure il numero di previsioni sbagliate in una classificazione.

L'obiettivo dell'apprendimento è quindi quello di trovare i parametri all'interno del modello che minimizzano l'errore.

Le funzioni error ed errorDerivative sono le due funzioni principali della classe ErrorFunctions: la prima calcola l'errore, la seconda calcola derivata dell'errore.

La classe MeanSquaredError, utilizzata all'interno del programma, calcola l'errore quadratico medio tra l'output del modello e l'output reale. La valutazione dell'errore viene effettuata tramite la misurazione della distanza euclidea tra l'output del modello model(input), calcolato dal vettore di input e il vettore di target contenente i valori reali. Il risultato viene poi normalizzato tenendo in considerazione il numero dei neuroni e il numero degli esempi.

$$E = \frac{1}{PN} \sum_{p=1}^{P} \sum_{i=1}^{N} (model(input) - target)^2$$

E = errore quadratico medio

N = numero di neuroni

P = numero di esempi

Optimizer
La classe Optimizer ha il compito di ricercare i valori dei parametri che riducano l'errore del modello.
Le sue due funzioni sono: *init* e *optimize*.
La prima funzione effettua una inizializzazione dei parametri con valori di default, in base al Model per cui è stato creato, mentre la seconda ha il fondamentale compito eseguire la minimizzazione; per gli algoritmi iterativi, viene eseguito un ciclo.

3.3.2 Reti neurali

ReClaM mette a disposizione alcune reti neurali predefinite, le quali si possono classificare in base al tipo di rete, alla funzione di attivazione dei neuroni nell'hidden layer e nell'output, e alla funzione di errore utilizzata durante l'addestramento.
Nella Tabella 5, vengono definiti i modelli della classe Model.

Name	Type	Activation Functions		Training Error Measure
		Hidden Neurons	Output Neurons	
FFNet	Feed Forward	$1/(1+e^{-a})$	$1/(1+e^{-a})$	none
MSEFFNet	Feed Forward	$1/(1+e^{-a})$	$1/(1+e^{-a})$	Mean Squared Error
LinOutFFNet	Feed Forward	$1/(1+e^{-a})$	a	none
LinOutMSEBFFNet	Feed Forward	$1/(1+e^{-a})$	a	Mean Squared Error
TanhNet	Feed Forward	$2/[(1+e^{-2a})-1]$	$2/[(1+e^{-2a})-1]$	Squared Error
LinearOutputTanhNet	Feed Forward	$2/[(1+e^{-2a})-1]$	a	Squared Error
ProbenNet	Feed Forward	a/(1+|a|)	a	Mean Squared Error
ProbenBNet	Feed Forward	a/(1+|a|)	a	Mean Squared Error
MSERNNet	Recurrent	$1/(1+e^{-a})$	$1/(1+e^{-a})$	Mean Squared Error
RBFNet	Radial Basis Function	special	special	none
MSERBFNet	Radial Basis Function	special	special	Mean Squared Error

Tabella 5: modelli definiti dalla classe Model

3.3.3 MyNet

In questa tesi è stata utilizzata la classe MyNet derivata del modello FFNet ((F)eed-(F)orward (Net)works) che mette a disposizione le funzioni per creare e lavorare con una rete neurale di tipo feedforward:
class MyNet : public FFNet

La classe MyNet ridefinisce la funzione di attivazione a tangente iperbolica per i neuroni nascosti attraverso il codice:

```
double g(double a) {
 return tanh(a);
 }
```

Viene definita la funzione derivata con:

```
double dg(double ga) {
 return 1 - ga * ga;
 }
```

Per i neuroni di output, la funzione che viene definita è lineare (con derivata pertanto pari a 1):

```
double gOutput(double a) {
 return a;
 }
double dgOutput(double ga) {
 return 1.;
 }
```

Attraverso la ridefinizione di queste funzioni, vengono sostituite alcune generiche proprietà della rete.
Il modello selezionato è considerato il più adeguato per la predizione dell'output, ovvero quella della conduttività termica dei nanofluidi.

3.4 Programma nanonetwork

Il programma, sviluppato, compilato ed eseguito in ambiente Linux, per la sua esecuzione necessita come parametro il nome di un file di testo che contiene alcuni criteri di elaborazione.

3.4.1 Esecuzione

Comando di esecuzione del programma dalla Shell dei comandi:
./nanonetworks fileConfigurazione

Il file fileConfigurazione, che nel caso dell'esempio deve trovarsi nella stessa cartella dell'eseguibile nanonetworks, contiene importanti parametri per l'addestramento: le variabili di input da considerare, la variabile di output da predire ed infine il percorso del file csv di ingresso da esaminare.

Contenuto del file fileConfigurazione:

```
// Names of input variables
Xv T
// Names of ouput variables (to predict)
lnanofluid
// Names of data files (one line each)
/home/NanofluidDataCsv/Al2O3_Therm_Cond.csv
```

3.4.2 Analisi del programma

Il programma è strutturato in due fasi principali: la fase di acquisizione dei dati dal csv e quella di addestramento della rete neurale.
La fase di acquisizione dati ha il compito di prelevare i valori di input dal csv e attraverso un algoritmo di parsing renderli disponibili al software, il quale li collocherà in una struttura adatta per le successive manipolazioni.
La fase di addestramento della rete neurale feedforward creata utilizzando la libreria Shark, invece, ha il compito di istruire la rete tramite l'apprendimento per effettuare successivamente la predizione della conduttività termica dei nanofluidi.

3.4.2.1 Fase di acquisizione dati

La fase di immissione dei dati di input ed output nella rete consiste nel fornire agli strati di ingresso i valori delle proprietà dei componenti che caratterizzano il nanofluido e allo strato di neuroni di uscita, i valori di conduttività termica del nanofluido ottenuti sperimentalmente.

File di ingresso e file di uscita

Tutti i dati a nostra disposizione sono raggruppati all'interno di diversi file di ingresso di tipo csv, suddivisi per tipo di nanofluido (Al_2O_3 e TiO_2) e modalità di preparazione (stirred e sonicated).
L'organizzazione dei dati nei file csv, suddivisi per nanofluido e modalità di preparazione, non varia.

I dati sperimentali studiati per il nanofluido Al_2O_3/Acqua e TiO_2/Acqua sono disposti in colonne secondo la seguente tabella:

n	particle	fluid	dparticle	dcluster	T	Xv	lparticle	lfluid	lnanofluid

n: numero di dato sperimentale
particle: tipo di nanocomponente;
fluid: tipo di fluido;
dparticle: dimensione del nanocomponente in nm;
T: temperatura del fluido, °C;
Xv: concentrazione di volume della particella, %vol;
lparticle: conduttività termica del nanocomponente, W/mK;
lfluid: conduttività termica del fluido, W/mK;
lnanofluid: conduttività termica del nanofluido, W/mK;

Nel formato csv i dati a disposizione vengono rappresentati nel seguente modo, rispettivamente per il Al_2O_3/Acqua e TiO_2/Acqua:

n,Particle,Fluid,dparticle(nm),dcluster(nm),T(°C),Xv(%vol),lparticle(W/mK),lfluid(W/mK),lnanofluid(W/mK)
1,Al2O3,water,30,285,1,4.00%,41.6,0.563,0.601
2,Al2O3,water,30,285,10,4.00%,40.4,0.58,0.638

n,Particle,Fluid,dparticle(nm),dcluster(nm),T(°C),Xv(%vol),lparticle(W/mK),lfluid(W/mK),lnanofluid(W/mK)
1,TiO2,water,30-50,220,1,1.00%,11.7,0.563,0.552
2,TiO2,water,30-50,220,10,1.00%,11.7,0.58,0.568

Tra tutti i dati che il file csv mette a disposizione, i parametri di input effettivamente utilizzati per l'addestramento della rete neurale sono due: la concentrazione di volume della particella (Xv) e la temperatura del fluido (T).

L'elevata flessibilità del programma consente di rendere dinamica la funzione di reclutamento dei dati dal file csv. Ciò avviene associando ai nomi delle colonne dei dati di input, Xv(%vol) ed T(°C) (escludendo le unità di misura), i due parametri elencati in fileConfigurazione, in questo caso Xv e T.
Lo stesso procedimento avviene per la variabile da predire, che nel caso della conduttività termica corrisponde a lnanofluid, ovvero la conduttività termica del nanofluido.

Il path che segue, /home/NanofluidDataCsv/Al2O3_Therm_Cond.csv, è invece il nome del csv di ingresso da dove i dati devono essere acquisiti; ovviamente è da rispettare il legame tra i nomi delle colonne del file csv di ingresso e i nomi delle variabili da utilizzare per l'addestramento.

L'output che il software genera, si suddivide in due tipologie di file di testo. Il primo file, chiamato *previsione*, contiene tante righe quanti sono i dati sperimentali posti sotto esame. Per ciascuna riga sono presenti due valori di input, nel nostro caso, quello della concentrazione di volume della particella e quello della temperatura, entrambi normalizzati, e due valori di output, rispettivamente il valore della conduttività termica misurata e quella predetta dalla rete neurale.
Esempio di una riga di output del file previsione:
Input: -1.45769 -1.21479 Output: 2.032 1.81504

La seconda tipologia di file che il programma genera, è quella focalizzata sull'errore. Vengono generati tanti file di testo quanti sono i dati sperimentali in esame. Quindi per ogni dato sperimentale vi è un file denominato *trajectory*, seguito da un valore numerico che rappresenta la sequenza del dato sperimentale computato. All'interno di ciascun file, sono presenti tante righe, quanti sono i cicli apprendimento, e considerando tutti i valori all'interno del file, viene osservato l'andamento dell'errore previsionale nella rete addestrata.
Ad esempio, per il file trajectory19, vengono visualizzati il numero di cicli, in questo caso da 1 a 5000, e i due errori, quello legato all'apprendimento nella seconda colonna e quello legato alla previsione nella terza colonna:
1 4.48108 3.00353
2 4.38284 2.94098
...
5000 0.054192 0.193493

Struttura dei dati
Il codice che segue rappresenta la struttura denominata NanofData.
Tale struttura viene generata all'esecuzione del programma nella fase di acquisizione dati e successivamente viene riempita inserendovi tutti i valori dal csv di input, per poi essere utilizzata nella fase successiva di addestramento.

```
struct NanofData{
 std::string name;
 std::vector<double> inputValues;
 std::vector<double> outputValues;
};
```

NanofData è composta:
- Stringa *name*: rappresenta il nome dal nanofluido che viene acquisito dalla prima riga del csv dei dati di ingresso;
- Array di double dai valori di ingresso *inputValues*: dati prelevati dal csv, e posti come parametri di ingresso per la rete neurale dal file di configurazione;
- Array di double dai valori di uscita *outputValues*: dati prelevati dal csv, e posti come parametri di uscita per la rete neurale dal file di configurazione.

3.4.2.2 Fase di addestramento e validazione

La fase di addestramento supervisionato ha l'obiettivo di generare una relazione interna nella rete neurale che lega gli input con gli output, e consentire la previsione del valore di uscita rispetto a uno stimolo di ingresso sconosciuto, basandosi su un numero limitato di esempi.
La valutazione delle performance della rete viene effettuata attraverso la tecnica leave-one-out cross validation; questa prevede che, per ogni campione, la rete venga addestrata con tutti gli altri campioni e venga valutato l'errore in base al valore attribuito al campione stesso.
Questa fase è strutturata su due cicli: il primo si ripete tante volte quanti sono gli esempi, il secondo ciclo (training) viene eseguito per un numero di volte pari al numero di ottimizzazioni che si vuole effettuare.
Nel ciclo di esempio, vengono impostati dei piccoli pesi casuali tra le connessioni dei neuroni (inizializzazione), invece nel ciclo di training avviene il vero a proprio apprendimento della rete neurale.
La rete, dopo la fase di addestramento, deve essere dotata di un'adeguata capacità di generalizzazione, che permette di produrre uscite a partire da dati in ingresso non presenti nel training set. Ogni rete è in grado di farlo, ma tuttavia ciò che conta è la coerenza con le uscite, vale a dire la relazione che sussiste con i dati forniti durante l'addestramento. Le prestazioni di una rete neurale perciò dipendono dall'insieme di esempi scelti per l'addestramento. Tali esempi devono essere rappresentativi della realtà che la rete deve apprendere e in cui essa verrà adoperata, utilizzando ingressi il più possibile distribuiti omogeneamente e il più possibile fitti per poter descrivere l'andamento della relazione ingressi-uscite.

3.4.2.3 Analisi del codice sorgente

Il file main.cpp è il file principale del programma nanonetworks e contiene entrambe le fasi di acquisizione dati e addestramento.
La libreria Shark è il punto di partenza per lo sviluppo del codice, perciò le prime istruzioni sono l'inclusione di alcuni suoi file per il suo utilizzo:

```
// from shark
#include <ReClaM/FFNet.h>
#include <ReClaM/FFNetSource.h>
#include <ReClaM/Rprop.h>
#include <ReClaM/MeanSquaredError.h>
#include <ReClaM/createConnectionMatrix.h>
```

Utilizzando la classe ReClaM sono presenti tutti gli elementi che consentono la creazione e l'utilizzo della rete neurale di tipo feed-forward chiamata FFNet.

Includendo invece:

```
#include <Array/Array.h>
#include <Array/ArrayIo.h>
```

Viene utilizzata la classe Array, che Shark mette a disposizione, perché permette l'utilizzo di molte funzioni che consentono di avere una maggiore gestione dinamica e facilità d'uso di array e matrici all'interno del programma. Le strutture degli array che contengono i dati possono essere modificate anche durante l'esecuzione del programma effettuando operazioni di ridimensionamento e i parametri possono essere passati sia per valore che per indirizzo.

Successivamente vengono inclusi
#include "ParseNanofData.h"
che consente di effettuare la fase di acquisizione dei dati dal csv per metterli a disposizione del software, e
#include "Normalizer.h"
che offre le routine di normalizzazione dei dati acquisiti.

Terminate le operazioni di inclusione di file esterni, viene ora analizzata la funzione main.
La prima operazione che viene affrontata è la fase di acquisizione dei dati che viene eseguita tramite un costruttore, creando l'oggetto *trainingData* appartenente alla classe *ParseNanofData* vista in precedenza.

ParseNanofData trainingData(argv[1]);

Al costruttore di trainingData viene passato argv[1] che rappresenta il primo parametro inserito durante il lancio del programma, ovvero il nome e il percorso del file di configurazione utilizzato per immettere dagli esempi.
TrainingData è un oggetto della classe ParseNanoData e contiene al suo interno i dati di input e output prelevati dai file csv selezionati.

Il numero di campioni in trainingData e la dimensionalità degli spazi di input e di output vengono successivamente resi espliciti tramite un assegnamento a tre differenti variabili:

const unsigned numberTrainingPatterns = trainingData.getNumberPatterns();
const unsigned numberInputFeatures = trainingData.getNumberInputFeatures();
const unsigned numberOutputFeatures = trainingData.getNumberOutputFeatures();

Il numero di righe del file csv è rappresentato dalla variabile *numberTrainingPatterns* e verrà utilizzato come parametro di condizione all'interno del primo ciclo for per generare le sottomatrici di trainInput e trainTarget contenenti i dati di training (tutti tranne il campione corrente) e valInput e valTarget (dati di testing, ovvero il campione corrente).
Le variabili *numberInputFeatures* e *numberOutputFeatures* sono invece il numero delle caratteristiche, rispettivamente di input e di output, da far apprendere alla rete neurale; nel nostro caso sono la concentrazione di volume della particella (%vol) e la temperatura del fluido (°C) per quanto riguarda i parametri di input, la conduttività termica del nanofluido (W/mK) per le caratteristiche di output.

Nel codice che segue, vengono dichiarati *trainInput e trainTarget:* array bidimensionali che verranno utilizzati per archiviare i dati di training. Entrambi hanno le medesime righe uguali al numero *numberTrainingPatterns-1* e un numero di colonne che dipendono dalle variabili *numberInputFeatures* e *numberOutputFeatures*.
Gli array *valInput* e *valTarget* sono invece array monodimensionali e verranno utilizzati per contenere i valori dello stimolo di input e l'output elaborato della rete.

// build the data structure for Shark
Array<double> trainInput(numberTrainingPatterns-1, numberInputFeatures);
Array<double> trainTarget(numberTrainingPatterns-1, numberOutputFeatures);

Array<double> valInput(1, numberInputFeatures);
Array<double> valTarget(1, numberOutputFeatures);

Da questo punto del codice sorgente, termina la fase di acquisizione dati e comincia la fase di apprendimento che sottoporrà ogni esempio all'addestramento per mezzo di un ciclo for che si ripeterà per *numberTrainingPatterns* volte.

for (int currentNanof = 0; currentNanof < trainingData.getNumberPatterns(); ++currentNanof) {
// choose what you want to learn
trainingData.fillInput(trainInput, trainTarget, valInput, valTarget, currentNanof);
//normalize input data
Normalizer normalizer;
normalizer.train(trainInput);
normalizer.normalize(trainInput);
normalizer.normalize(valInput);

Attraverso il metodo fillInput si esegue la funzione che consente di estrapolare da trainingData gli esempi raccolti e disporli nell'array di addestramento (trainInput e trainTarget) e nell'array con i valori di esempio (valInput, valTarget). L'indice *currentNanof* indica il numero del campione appartenente al dataset, i cui valori di ingresso ed uscita devono essere copiati all'interno degli array monodimensionali *valInput* e *valTarget.* In tutti gli altri casi, i valori di ingresso ed uscita vengono disposti all'interno degli array bidimensionali *trainInput* e *trainTarget*.
Il processo di normalizzazione, implementato tramite la classe *Normalizer*, viene utilizzato per evitare che gli ordini di grandezza delle variabili di input influenzino in modo diverso i pesi delle variabili di ingresso che hanno ordini di grandezza inferiori durante l'ottimizzazione. Con tal accorgimento, viene data la medesima importanza a tutti i valori, sia che questi appartengano a un range di input ampio o ristretto.
La normalizzazione viene effettuata calcolando la media e la deviazione standard di ciascun dato di input; successivamente, a tutti i campioni viene sottratta la media e vengono divisi per la deviazione standard.

La tipologia di struttura della rete neurale prescelta, basata sull'utilizzo della libreria Shark, è riportata nel codice che segue:

```
// define topology unsigned
unsigned inputs = numberInputFeatures;
unsigned firstHiddenLayer = numberInputFeatures + 1;
unsigned secondHiddenLayer = 0;
unsigned outputs = numberOutputFeatures;

//define learning cycles
unsigned numberOfLearningCycles = 5000;

Array<int> con;
createConnectionMatrix(con, inputs, firstHiddenLayer, secondHiddenLayer, outputs);

// feed-forward neural network
```

```
MyNet net(inputs, outputs, con), netMin(inputs, outputs, con);

// error function
MeanSquaredError error;

// optimizer
IRpropPlus optimizer;
optimizer.init(net);

// initialize the weights uniformly between -0.1 and 0.1
net.initWeights(-0.1, 0.1);
```

La rete neurale è costituita da un numero di neuroni di ingresso ed uscita che è stabilito nel file di configurazione, infatti, alle variabili *inputs* e *outputs* vengono associate le variabili precedentemente descritte *numberInputFeatures* e *numberOutputFeatures*. In seguito vengono definiti il numero di neuroni appartenenti ai due strati nascosti tramite le variabili *firstHiddenLayer* e *secondHiddenLayer,* e il numero di cicli di apprendimento che la rete dovrà sostenere tramite la variabile *numberOfLearningCycles* utilizzata all'interno di un ciclo for.

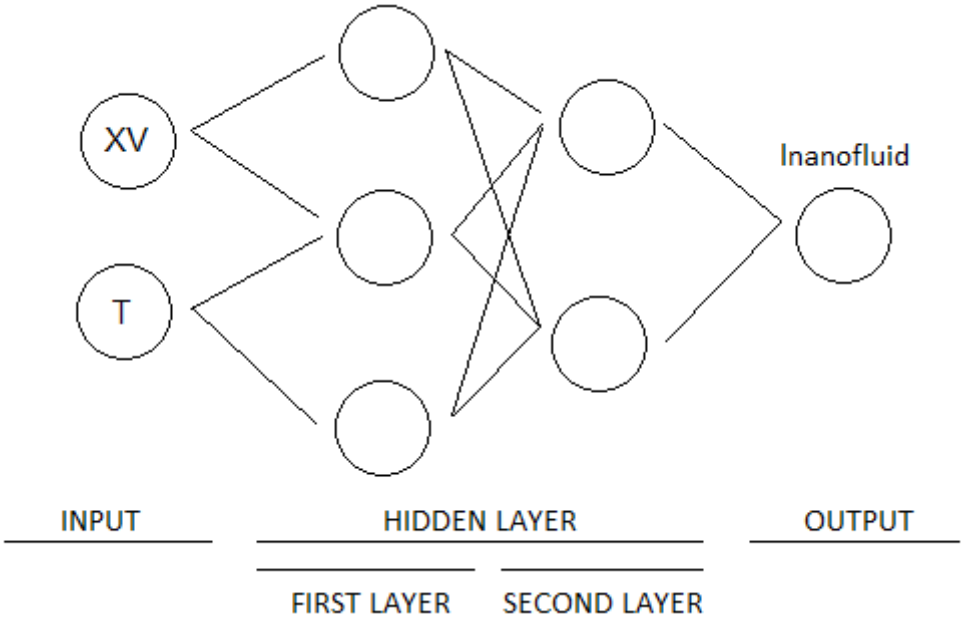

Figura 26: Rappresentazione della rete neurale artificiale costituita dal primo strato di ingresso, da due strati nascosti e lo strato di uscita

Le connessioni tra i neuroni vengono rappresentate tramite l'array *con;* per generarle in maniera opportuna, viene utilizzata la funzione createConnectionMatrix.
I modelli feedforward sono rappresentati dagli oggetti *net* e *netMin*, e con la procedura *initWeights* vengono inizializzati i pesi tra le connessioni con valori random compresi tra -0.1 e +0.1.
La funzione d'errore proviene dalla classe *MeanSquaredError.*
L'algoritmo di ottimizzazione utilizzato è il Resilient Backpropagation (Rprop) migliorato, simile al backpropagation descritto nei capitoli precedenti. Le differenze tra i due algoritmi sono minime:
Rprop utilizza i segni delle derivate parziali e non il valore assoluto dei valori per modificare i

parametri.
L'oggetto ottimizzatore *optimizer* è definito attraverso la classe IrpropPlus e inizializzato con parametri di default tramite la procedura init.

L'addestramento della rete neurale viene eseguito tramite un ciclo for chiamando la funzione *optimize* che eseguirà l'algoritmo *Rprop*, la quale modificherà i parametri delle connessioni tra i neuroni per cercar di diminuire gli errori di apprendimento e di previsione che sono forniti in uscita tramite un file di testo denominato trajectory seguito dal numero di esempio associato:

```
string trajFilename="trajectory" + to_string<int>(currentNanof);
std::ofstream trajectory(trajFilename.c_str());

// training loop
for (int t = 1; t <= numberOfLearningCycles; t++) {
 // train the network
 optimizer.optimize(net, error, trainInput, trainTarget);
 err = error.error(net, trainInput, trainTarget);
 if (err < errMin) {
   errMin = err;
   netMin = net;
  // trajectory << "Best network so far" << endl;
  }
 // write results
  trajectory << t << "\t" << error.error(net, trainInput, trainTarget)<< "\t" << error.error(net, valInput, valTarget) << std::endl;
}
trajectory.close();
```

Il file trajectory, per ogni esempio, visualizza un numero di righe pari a numberOfLearningCycles e in ciascuna di queste vengono riportati due errori.
Il primo valore, restituito dalla funzione error.error(net, trainInput,trainTarget), calcola l'errore tra l'output che la rete "net" produce a partire dai dati di input trainInput e la misura effettuata per quei dati di input (misura che e' la stessa utilizzata per il training). Il valore di errore restituito, indica quanto il modello è capace di rappresentare la relazione ingresso-uscita tra i dati forniti (se il valore che la rete produce per un certo dato di input e' completamente diverso da quello misurato, vuol dire che la struttura della rete o i suoi parametri non sono adeguati a rappresentare la relazione ingresso-uscita).
La funzione error.error(net, valInput, valTarget), invece, restituisce l'errore tra l'output della rete per valInput (valori di input per dati non utilizzati nel training) e la misura effettuata (valTarget); questo valore quantifica le capacità predittive della rete. Il confronto tra l'errore calcolato e l'errore errMin permette di rilevare l'errore minimo di training.

Le previsioni dell'output vengono generate e visualizzate tramite il file di testo *previsione*.

```
 Array<double> in(numberInputFeatures), out(numberOutputFeatures);
 for (int j= 0; j < numberInputFeatures; j++)
 in(j) = valInput(0, j);
```

```
net.model(in, out);
previsione << "Input:\t";
for (int j = 0; j < numberInputFeatures; j++)
previsione << valInput(0,j) << "\t";
previsione << "Output:\t" << valTarget(0) << "\t" << out(0) << endl;
```

Questa sezione di codice, che viene ciclata per un numero di volte pari al numero di esempi a disposizione, effettua la previsione dell'output in base all'input fornito.
Il metodo *model* elabora l'output del modello causato dallo stimolo dell'input. Alla funzione vengono forniti due parametri che corrispondono rispettivamente l'array che contiene i valori degli ingressi e l'array di output che conterrà la risposta del modello. I risultati ottenuti vengono visualizzati in un file di testo che conterrà il valore reale e quello predetto.

Dall'esecuzione di questo codice, come già precedentemente detto, si ottengono due file che verranno utilizzati per tracciare alcuni grafici utili per analizzare la bontà e le performance della struttura della rete utilizzata.

CAPITOLO 4

RISULTATI SPERIMENTALI

4.1 Introduzione

La prestazione della predizione di una rete neurale è influenzata da tre fattori:
a. le dimensioni del training set;
b. l'architettura della rete neurale;
c. la complessità del problema.
Siccome la complessità del problema è incognita, nell'analisi che segue verranno prese in esame la struttura delle rete feed-forward e l'insieme dei campioni da utilizzare durante la fase di addestramento.

a. Training set
Una rete progettata per generalizzare bene produce mappature input/output corrette. Se la rete viene sovra-addestrata con troppi esempi (overfitting), la rete rischia di memorizzare il training set perdendo la capacità di generalizzare in quanto risulta avere una elevata accuratezza di classificazione per i campioni di training e una bassa accuratezza di classificazione per i campioni sconosciuti.
Per limitare questo fenomeno si può utilizzare un sottoinsieme degli esempi come training set fermando l'addestramento quando l'errore di training comincia a crescere sugli esempi proposti.
Per le reti multistrato, a partire dai risultati sulla teoria statistica dell'apprendimento, sono state stabilite delle stime del numero minimo dei campioni di training che occorrono per addestrare una rete in modo tale che si abbia una adeguata capacità di generalizzazione.
Nella pratica, siccome le stime teoriche possono essere inadeguate, bisogna basarsi su opportune euristiche per la scelta della struttura e la definizione del training set.

b. Struttura della rete feed-forward
La struttura della rete feed-forward è composta da tre strati: ingresso, uscita e lo strato nascosto.
Lo strato nascosto è costituito da neuroni che codificano in modo distribuito i tratti più importanti del vettore dei dati di ingresso. In tal modo, al termine della fase di apprendimento, ogni nodo che compone lo strato nascosto tende a rappresentare parte dei tratti rilevanti dell'input in modo sfumato e non esclusivo.
E' stato dimostrato che una rete con un numero sufficiente di neuroni all'interno dell'hidden layer è in grado di approssimare con precisione arbitraria:
- qualsiasi funzione continua, con un solo strato nascosto;
- qualsiasi funzione, anche discontinua, con due strati nascosti;
Al momento non è possibile determinare a priori con una adeguata precisione il numero necessario di strati nascosti, ne il numero di neuroni che devono essere contenuti al suo interno per computare una funzione non lineare. Tuttora, nonostante alcune regole empiriche, ci si basa sull'esperienza e su alcuni metodi euristici per determinare la struttura della rete.
Se la struttura della rete neurale è costituita da un basso numero di strati nascosti o di neuroni, la rete non è in grado di approssimare con adeguata precisione la funzione incognita posta sotto esame, o perché la funzione è troppo complessa oppure perché l'algoritmo di backpropagation

cade all'interno di in un minimo locale. Se invece la rete è composta da un numero troppo elevato di strati nascosti o di neuroni, molto probabilmente si va incontro al problema di overfitting causando un peggioramento della capacità di generalizzazione.

4.2 Analisi dei dati di output

I dati elaborati dalla rete neurale sono stati raccolti facendo variare la struttura della rete e il training set.
La prima variazione effettuata è il raggruppamento dei dati per l'addestramento in base alla dimensione del cluster (che dipende la tipologia di produzione del nanofluido: stirred, sonicated o entrambe).
Successivamente è stata fatta variare la struttura della rete alterando il numero di neuroni del secondo strato nascosto e mantenendo sempre costante a tre i neuroni del primo hidden layer.
L'ultima modifica, che riguarda la fase di addestramento, è il numero cicli di apprendimento.
I nanofluidi che sono stati oggetto di studio sono l'Al_2O_3/Acqua e il TiO_2/Acqua .

In Tabella 6 sono rappresentate tutte le combinazioni utilizzate.

Csv	Cicli di apprendimento	Neuroni Second Hidden Layer
Al2O3_Therm_Cond	100	0 - 3
Al2O3_Therm_Cond_sonicated	1000	0 - 3
Al2O3_Therm_Cond_stirred	1000	0 - 3
Al2O3_Therm_Cond_sonicated	5000	0 - 3
Al2O3_Therm_Cond_stirred	5000	0 - 3
TiO2_Therm_Cond	100	0 - 3
TiO2_Therm_Cond_sonicated	1000	0 - 3
TiO2_Therm_Cond_stirred	1000	0 - 3
TiO2_Therm_Cond_sonicated	5000	0 - 3
TiO2_Therm_Cond_stirred	5000	0 - 3

Tabella 6: Combinazioni di struttura e dati utilizzati per l'addestramento delle reti neurali

4.2.1 Analisi dei risultati raccolti per Al_2O_3/Acqua

Le analisi che seguiranno tratteranno i dati di output del nanofluido Al_2O_3/Acqua.
Vengono ora presi in considerazione sei diverse combinazioni che si differenziano per training set (Al2O3_Therm_Cond, Al2O3_Therm_Cond_sonicated e Al2O3_Therm_Cond_stirred), numero di cicli di apprendimento (100, 1000 e 5000) e numero di neuroni nel secondo hidden layer (0 e 3).

Csv	Cicli di apprendimento	Second hidden layer	Nome Grafico
Al2O3_Therm_Cond	100	0	Al2O3_All_100_0
Al2O3_Therm_Cond_sonicated	100	0	Al2O3_Son_100_0
Al2O3_Therm_Cond_sonicated	1000	0	Al2O3_Son_1000_0
Al2O3_Therm_Cond_sonicated	1000	3	Al2O3_Son_1000_3
Al2O3_Therm_Cond_sonicated	5000	0	Al2O3_Son_5000_0
Al2O3_Therm_Cond_stirred	1000	0	Al2O3_Sti_1000_0

Tabella 7: Struttura e csv utilizzati per il confronto dei dati di output predetti e misurati

La rappresentazione tramite grafici degli output confronta la conduttività termica misurata sperimentalmente e quella invece predetta dalla rete neurale.
Nei diagrammi sono presenti due curve che rappresentano l'output misurato e l'output previsto. Le linee grafiche sono costruite dai valori forniti dal file *previsione* generato dal programma e raggruppati per concentrazione di volume della particella (%vol).

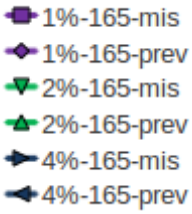

Figura 27: Legenda Al2O3: concentrazione in volume - dimensione nanoparticelle - valore misurato o predetto

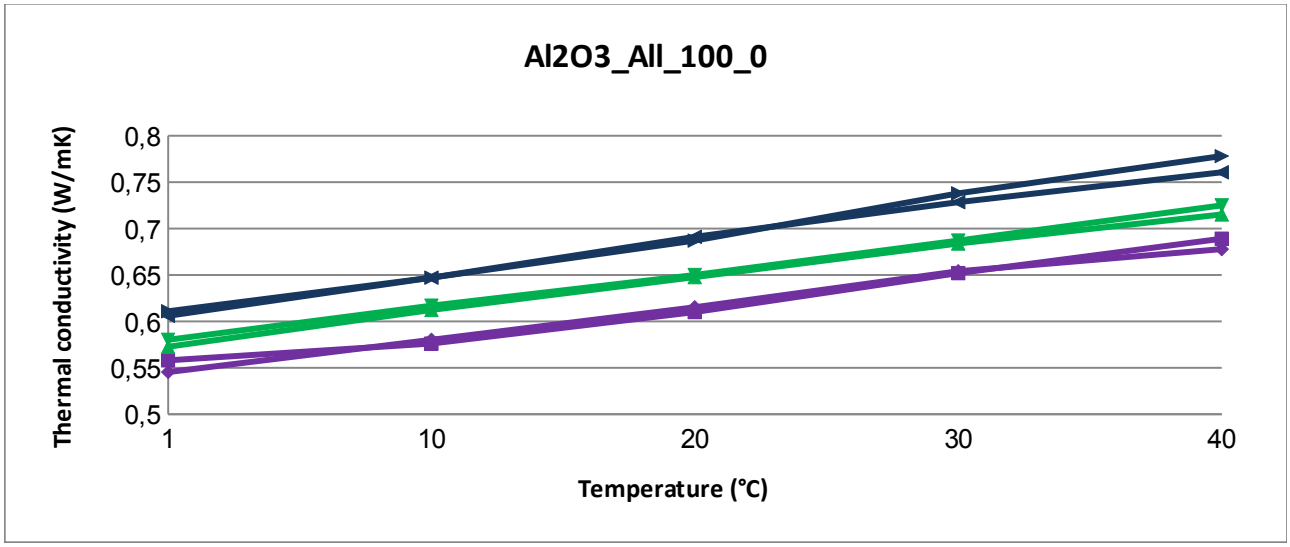

Grafico 2: Nanofluido: Al2O3, data set: sonicated e stirred, cicli di apprendimento: 100, neuroni secondo strato nascosto: 0

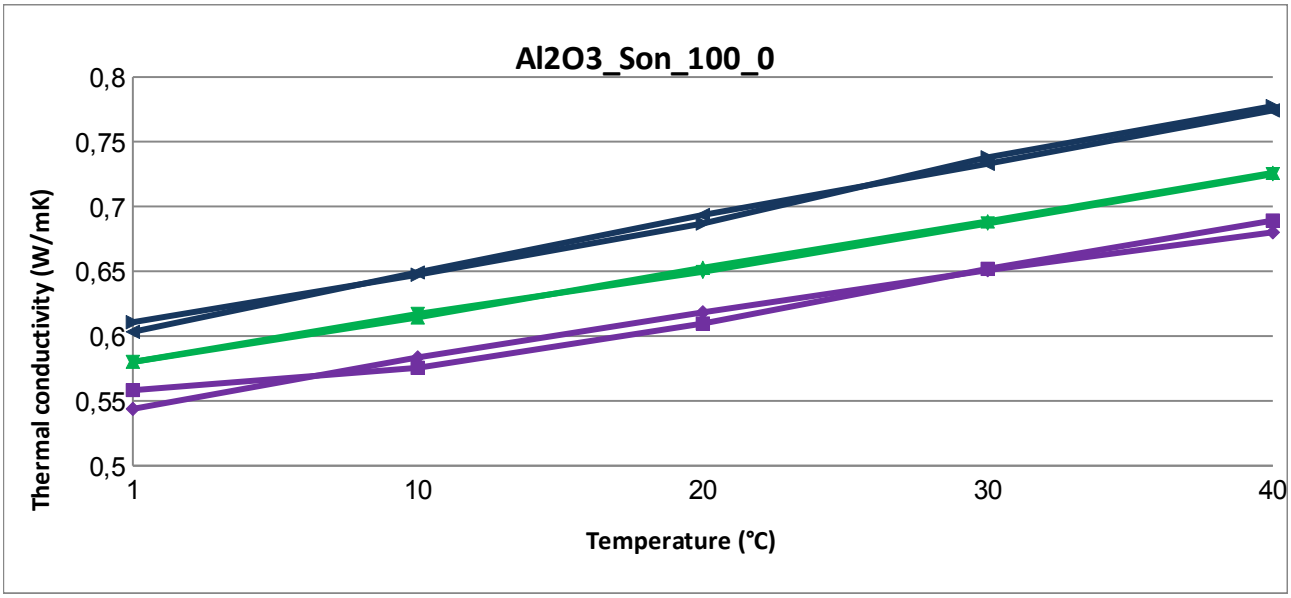

Grafico 3: Nanofluido: Al2O3, data set: sonicated, cicli di apprendimento: 100, neuroni secondo strato nascosto: 0

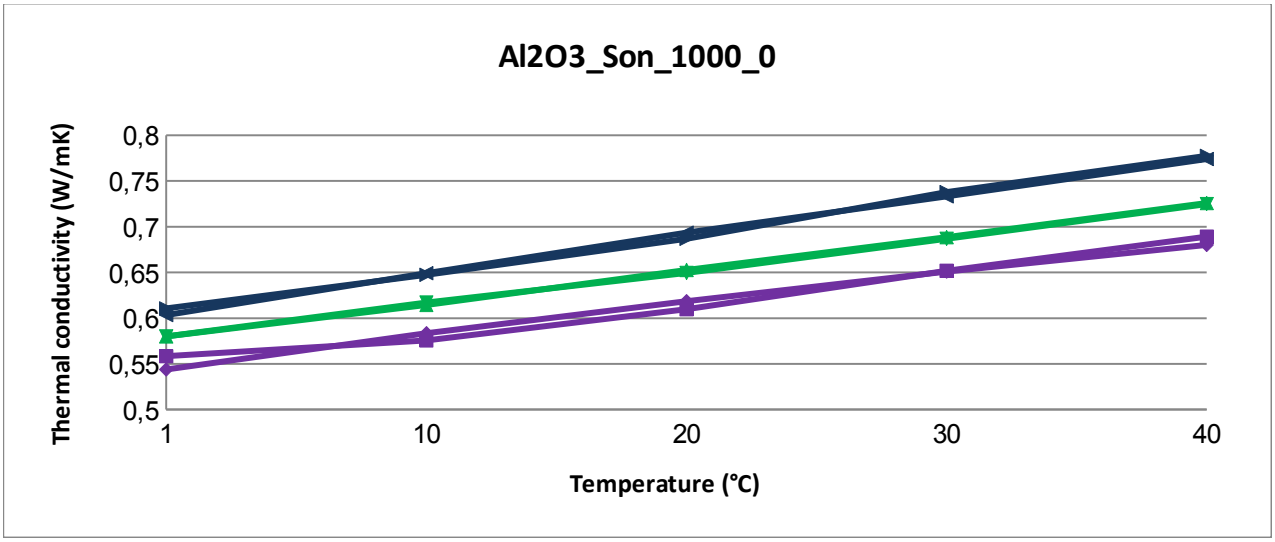

Grafico 4: Nanofluido: Al2O3, data set: sonicated, cicli di apprendimento: 1000, neuroni secondo strato nascosto: 0

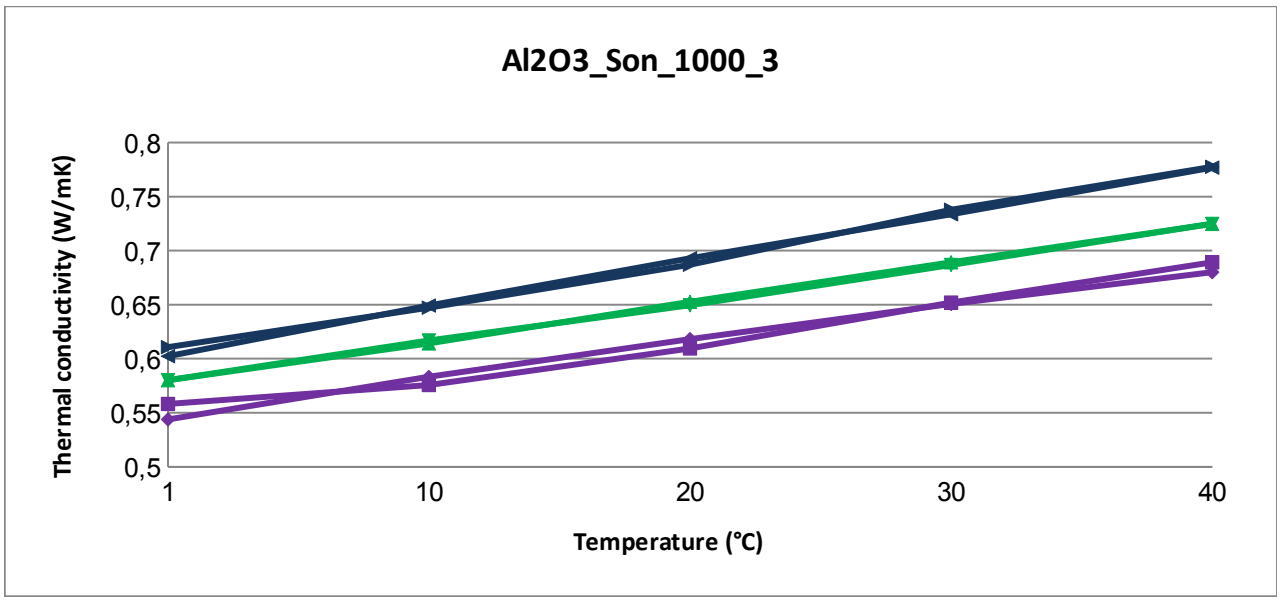

Grafico 5:Nanofluido: Al2O3, data set: sonicated, cicli di apprendimento: 1000, neuroni secondo strato nascosto: 3

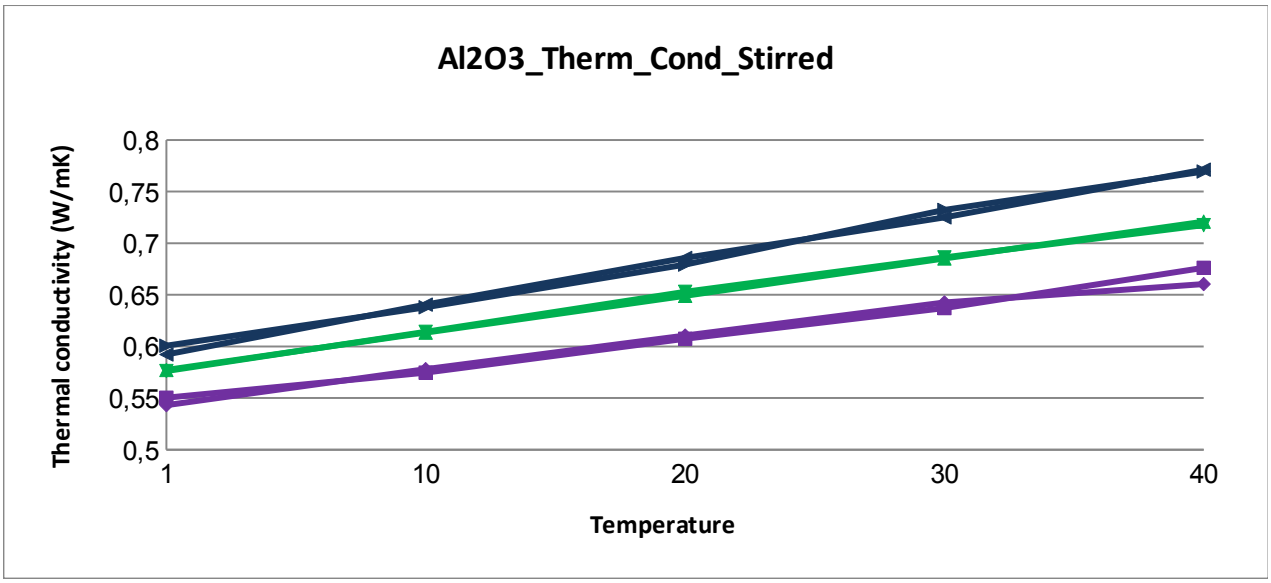

Grafico 7: Nanofluido: Al2O3, data set: sonicated, cicli di apprendimento: 1000, neuroni secondo strato nascosto: 0

I grafici 2, 3, 4, 5, 6 e 7 evidenziano che tutte le conduttività termiche predette tendono ad avere una buona accuratezza e quindi a seguire l'andamento delle conduttività termiche misurate. Lo discostamento dal valore reale è ridotto e le previsioni possono essere considerate buone.

4.2.1.1 Errore di Training

I valori che il file *trajectory* restituisce permette di confrontare, tra le varie combinazioni di strutture delle reti neurali e due diversi training set, quale tra le reti poste ad analisi ha una migliore predizione e quale esprime un maggiore legame tra le variabili di ingresso e quella di uscita.

Di seguito vengono mostrati i grafici di entrambi gli errori di tutte le analisi effettuate per trarre successivamente delle conclusioni più ampie e accurate.
I grafici che seguono rappresentano l'andamento dell'errore del training set riferiti ad un unico esempio. In questo caso è stato utilizzato l'ultimo file trajectory che il software ha generato.
In ogni grafico sono presenti quattro serie, ciascuna con un numero di neuroni di secondo livello diverso e tutti i grafici sono ottenuti con diversi numeri di cicli di apprendimento e due diversi training set.

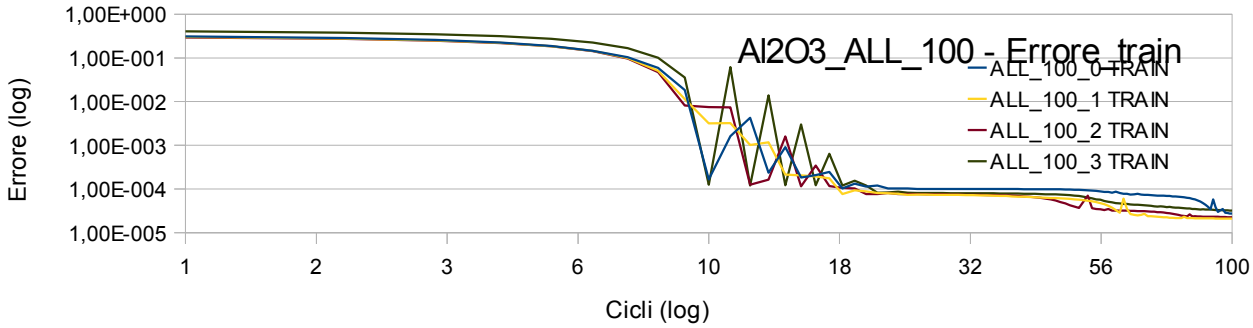

Grafico 8: Relazione input-output ottenuta con 100 cicli di learning, con training set composto da sonicated e stirred al variare del numero di neuroni del secondo strato nascosto

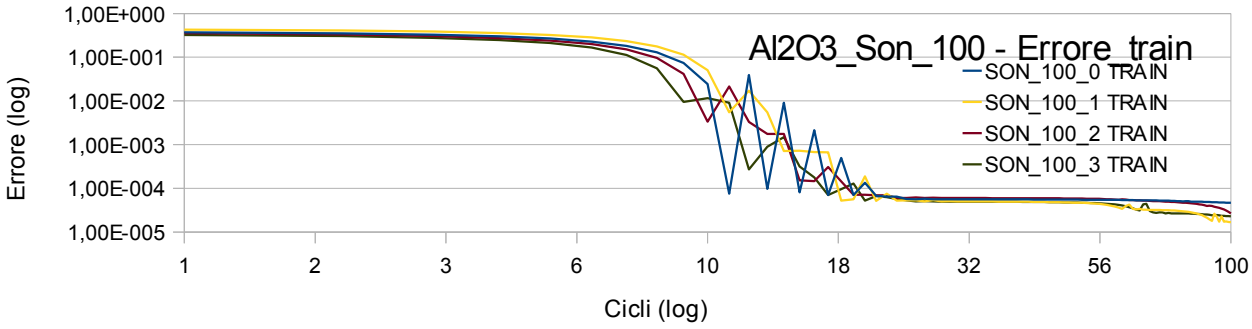

Grafico 9: Relazione input-output ottenuta con 100 cicli di learning, con training set composto da sonicated al variare del numero di neuroni nel secondo strato nascosto.

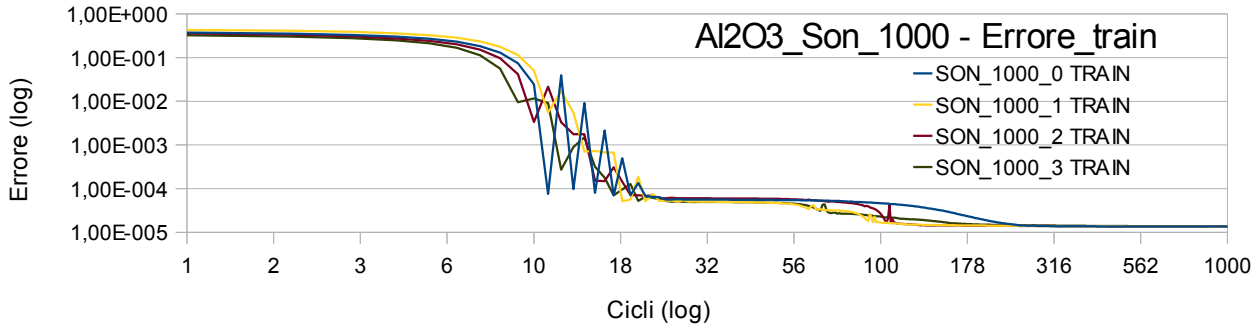

Grafico 10: Relazione input-output ottenuta con 1000 cicli di learning, con training set composto da sonicated al variare del numero di neuroni nel secondo strato nascosto.

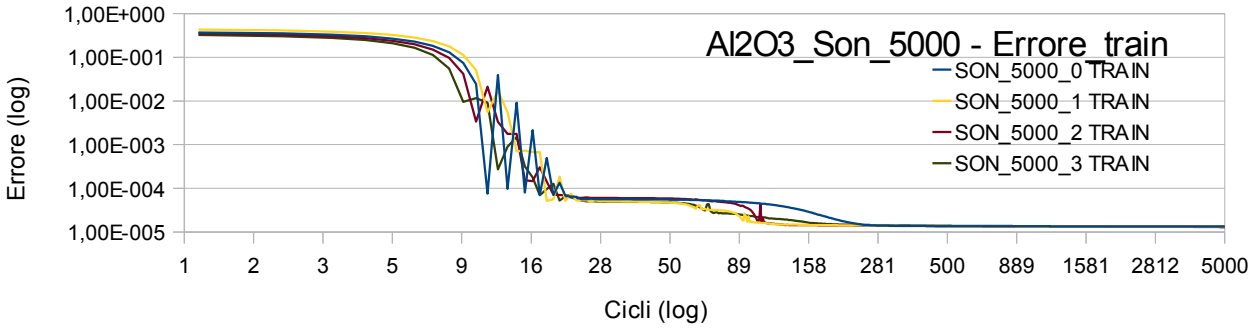

Grafico 11: Relazione input-output ottenuta con 5000 cicli di learning, con training set composto da sonicated al variare del numero di neuroni nel secondo strato nascosto.

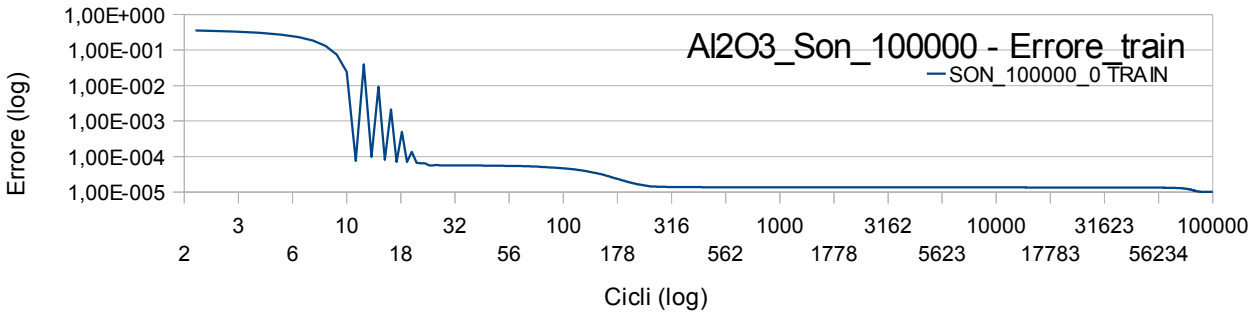

Grafico 12: Relazione input-output ottenuta con 100000 cicli di learning, con training set composto da sonicated con 0 neuroni al secondo strato nascosto.

Nei grafici 7, 8, 9, 10, 11 e 12, come aspettato, nel training set è presente un andamento decrescente all'aumentare del numero di cicli.
In tutte le analisi effettuate, l'andamento dell'errore è molto incostante tra i 10 e i 20 cicli, e si stabilizza dei successivi cicli con un andamento sempre a dente di sega, ma con ampiezze nettamente inferiori.

Vengono ora confrontati l'andamento dell'errore di addestramento di un singolo file trajectory e l'andamento dell'errore medio di addestramento di tutti i file trajectory.

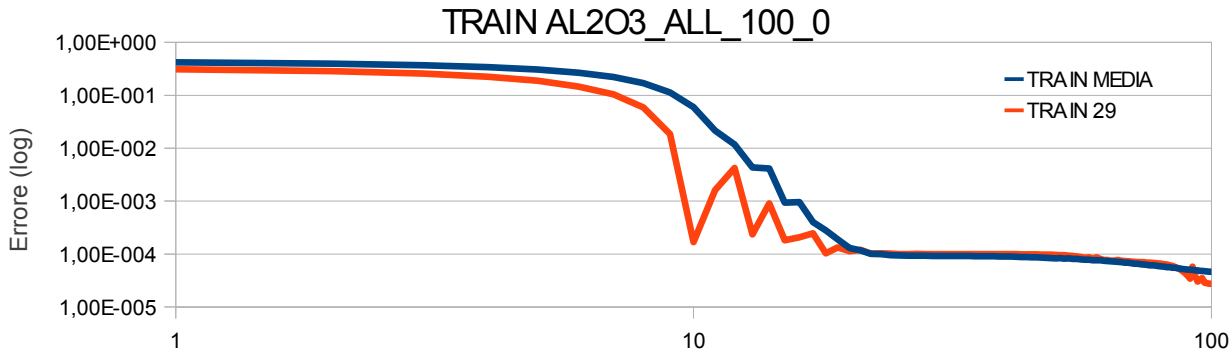

Grafico 13: Confronto tra la relazione input-output tra valori medi di tutti i trajectory e trajectory 29, ottenuta con 100 cicli di learning, con training set composto da sonicated e stirred con zero neuroni nel secondo strato nascosto.

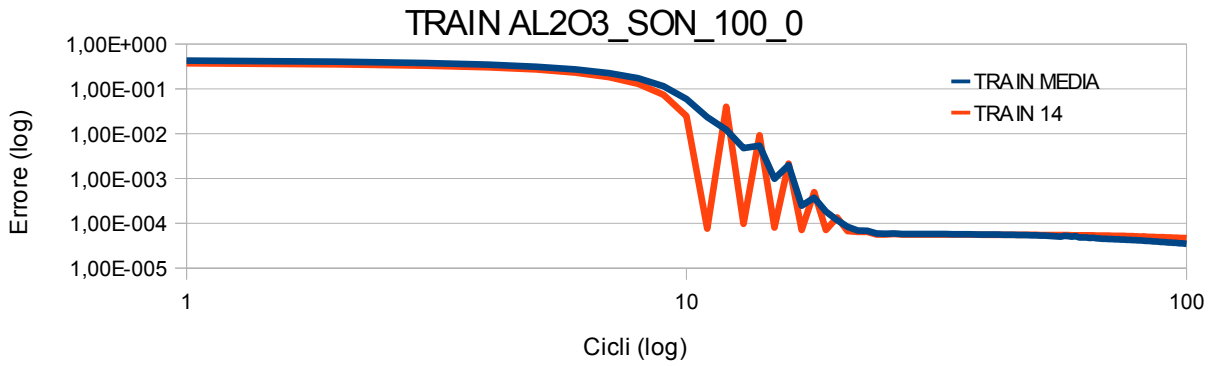

Grafico 14: Confronto tra la relazione input-output tra valori medi di tutti i trajectory e trajectory 14, ottenuta con 100 cicli di learning, con training set composto da sonicated e con zero neuroni nel secondo strato nascosto.

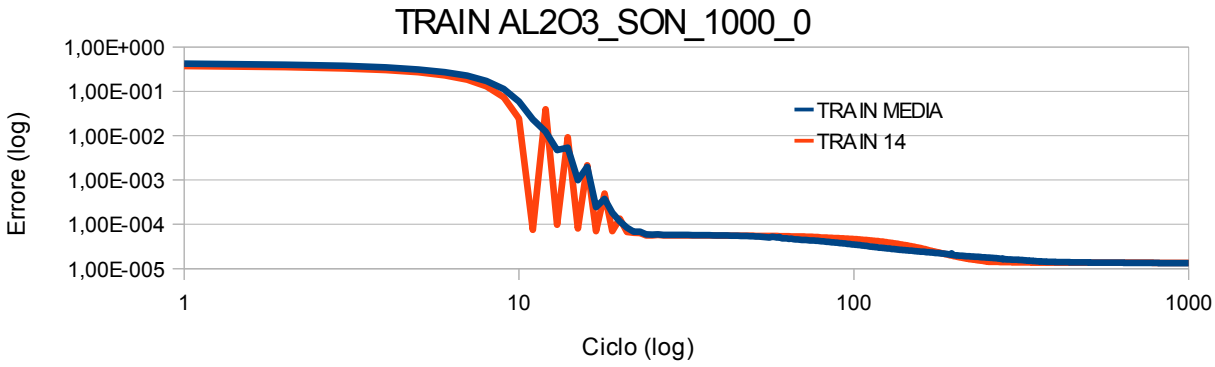

Grafico 15: Confronto tra la relazione input-output tra valori medi di tutti i trajectory e trajectory 14, ottenuta con 1000 cicli di learning, con training set composto da sonicated e con zero neuroni nel secondo strato nascosto.

In tutti e tre i casi analizzati, dopo circa 25 cicli, l'andamento della relazione input-output della media dei trajectory è quasi identica a quella del singolo trajectory . Si possono quindi considerare simili i due errori di training per un numero di cicli elevato.

4.2.1.2 Errore di Validation

I grafici della capacità di predizione vengono, anche in questo caso, suddivisi per numero di cicli di apprendimento utilizzati e training set. Per ciascun grafico sono presenti quattro serie di dati che rappresentano la variazione di neuroni dello strato nascosto. I valori degli errori di validation set sono stati prelevati dall'ultimo file trajectory che il programma ha creato.

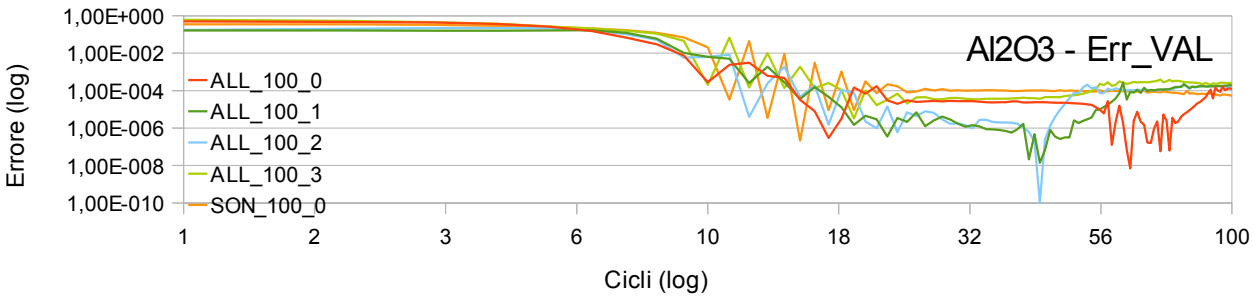

Grafico 16: Capacità di predizione della rete ottenuta con 100 cicli di apprendimento, con training set composto da sonicated e stirred al variare del numero di neuroni del secondo strato nascosto.

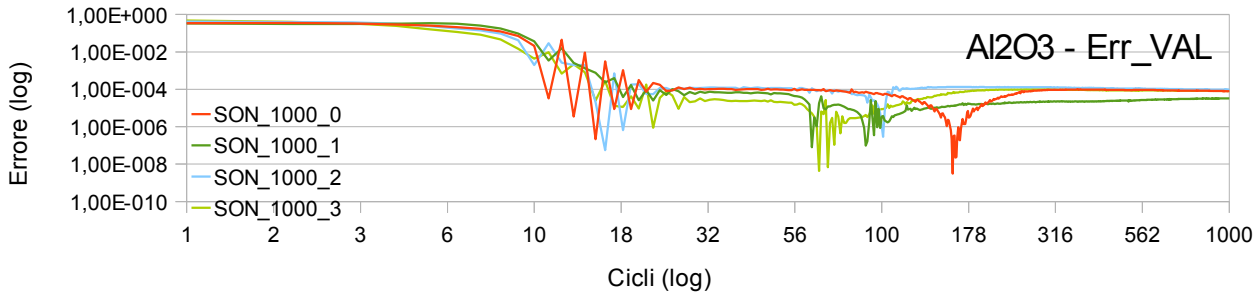

Grafico 18: Capacità di predizione della rete ottenuta con 1000 cicli di apprendimento, con training set composto da sonicated al variare del numero di neuroni del secondo strato nascosto.

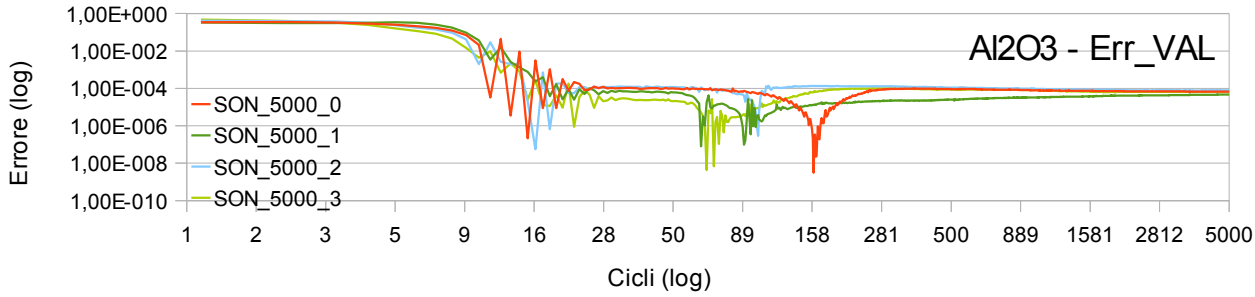

Grafico 19: Capacità di predizione della rete ottenuta con 5000 cicli di apprendimento, con training set composto da sonicated al variare del numero di neuroni del secondo strato nascosto.

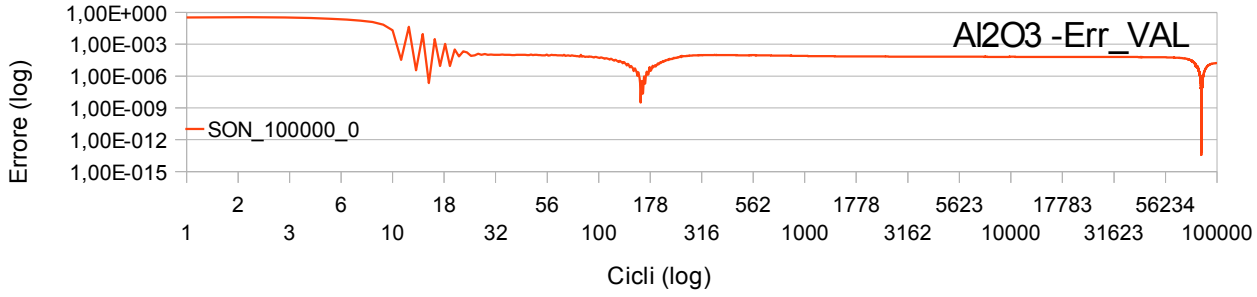

Grafico 20: Capacità di predizione ottenuta con 100000 cicli di learning, con training set composto da sonicated con nessun neurone al secondo strato nascosto.

Dai grafici 16, 17, 18, 19 e 20 si può notare che l'andamento dell'errore presenta un elevato numero di minimi relativi sopratutto con un basso numero di cicli di apprendimento.
Dai grafici 16 e 17 ottenuti con un numero di cicli pari 100 si può osservare, ad esclusione di ALL_100_2, che i punti di minimo assoluto hanno valori molto simili ai minimi locali e non sempre i punti di minimo assoluto corrispondono a un numero elevato di cicli.
Con un numero di 1000 cicli di apprendimento (grafico 18) si utilizza un campo di valori delle ascisse più esteso. Questo accorgimento genera un andamento dell'errore che consente di completare le code dei picchi di alcune serie (SON_100_1, SON_100_2 e SON_100_3) oppure di generare un picco di minimo assoluto (SON_100_0).
Aumentando il numero di cicli, nel caso successivo 5000 (grafico 19), non si riscontrano sostanziali cambiamenti. Tutti i valori di errore risultano pressoché costanti e comunque maggiori del picco minimo assoluto.
Utilizzando invece un numero di cicli estremamente grande, 100000 (grafico 20), si è voluto verificare se l'andamento dell'errore rimaneva invariato. Per molti cicli questo è avvenuto, ma a circa 80000 cicli si nota la presenza del nuovo picco di minimo assoluto.

Considerando la media di tutti valori della capacità di predizione dei file trajectory analizzati, viene di seguito effettuato il confronto con la capacità di predizione di un singolo file trajecory. Le due serie prendono rispettivamente i nomi “VAL MEDIA” e “VAL 29” oppure “VAL 14”.

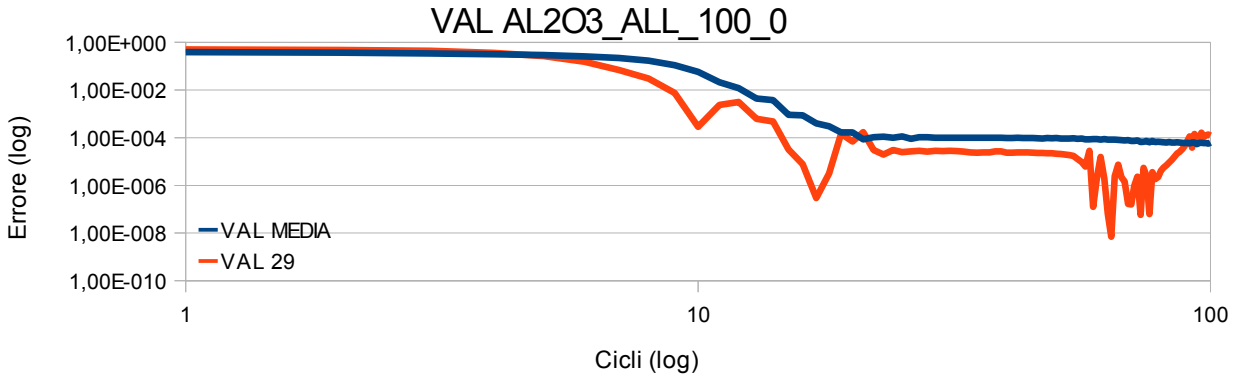

Grafico 21: Confronto dell'errore di predizione tra il valore medio di tutti i trajectory e trajectory 29, ottenuta con 100 cicli di learning, con training set composto da sonicated e stirred con zero neuroni nel secondo strato nascosto.

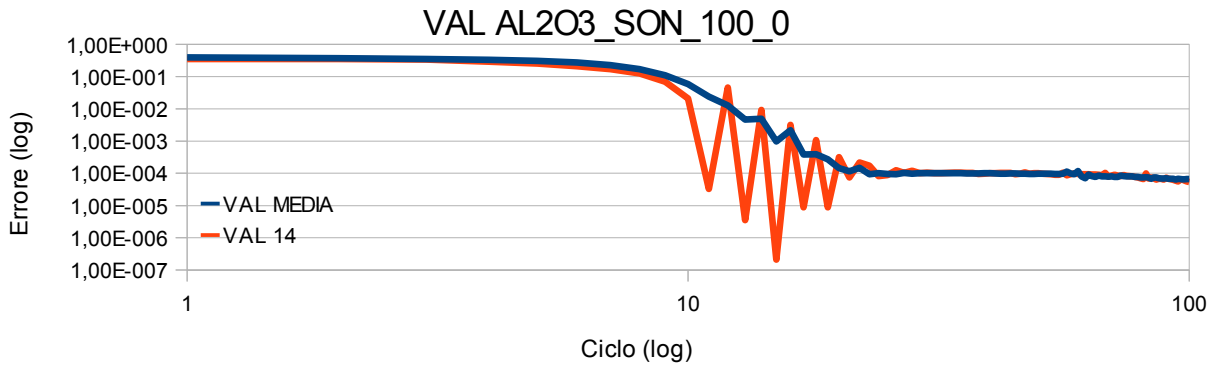

Grafico 22: Confronto dell'errore di predizione tra il valore medio di tutti i trajectory e trajectory 14, ottenuta con 100 cicli di learning, con training set composto da sonicated con 0 neuroni nel secondo strato nascosto.

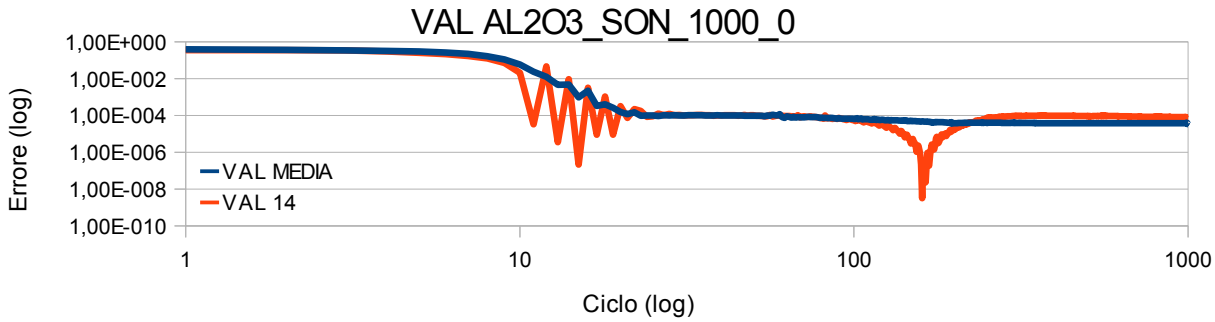

Grafico 23: Confronto dell'errore di predizione tra il valore medio di tutti i trajectory e trajectory 14, ottenuta con 1000 cicli di learning, con training set composto da sonicated con 0 neuroni nel secondo strato nascosto.

L'andamento dell'errore della media dei trajectory decresce all'aumentare del numero di cicli e l'instabilità che si riscontra a circa 20 cicli è meno evidente rispetto al singolo trajectory. Con un elevato numero di cicli si osserva solamente che nel grafico 21 i due errori di predizione sono distanti una unità di grandezza per poi sovrapporsi poco prima del centesimo ciclo.
Nel grafico 23 l'andamento dell'errore di predizione tra i valori medi di tutti i trajectory non presenta overfitting.
In generale, con un numero di cicli superiore a 100 l'errore di predizione delle due serie sono molto simili tra loro.

In tabella 8 sono riportati i valori di errore minimo assoluto e il rispettivo ciclo corrispondenti alla predizione e alla relazione ingresso-uscita.

Nanofluido	Training set	Learning Cycles	Neuroni Hidden Layer	Relazione ingresso-uscita		Predizione	
				Valore	Ciclo	Valore	Ciclo
Al2O3-Acqua	So+St	100	0	2,75E-005	99	7,03E-009	64
Al2O3-Acqua	So+St	100	1	2,10E-005	98	1,38E-008	43
Al2O3-Acqua	So+St	100	2	2,26E-005	100	1,06E-010	43
Al2O3-Acqua	So+St	100	3	3,19E-005	100	3,31E-006	19
Al2O3-Acqua	So	100	0	4,65E-005	100	2,13E-007	15
Al2O3-Acqua	So	100	1	1,67E-005	100	7,95E-008	63
Al2O3-Acqua	So	100	2	2,68E-005	100	5,58E-008	16
Al2O3-Acqua	So	100	3	2,29E-005	100	4,26E-009	66
Al2O3-Acqua	So	1000	0	1,36E-005	1000	3,15E-009	160
Al2O3-Acqua	So	1000	1	1,35E-005	999	7,95E-008	63
Al2O3-Acqua	So	1000	2	1,34E-005	1000	5,58E-008	16
Al2O3-Acqua	So	1000	3	1,37E-005	1000	4,26E-009	66
Al2O3-Acqua	So	5000	0	1,35E-005	4984	3,15E-009	160
Al2O3-Acqua	So	5000	1	1,34E-005	4986	7,95E-008	63
Al2O3-Acqua	So	5000	2	1,29E-005	5000	5,58E-008	16
Al2O3-Acqua	So	5000	3	1,35E-005	4992	4,26E-009	66
Al2O3-Acqua	So	100000	0	1,01E-005	99856	3,62E-014	84072

Tabella 8: Valori degli errori della relazione ingresso-uscita e di predizione. So = Sonicated, St = Stirred

L'errore di apprendimento e relazione ingresso-uscita assoluti, riferiti al singolo file trajectory in esame, sono stati trovati con una rete con zero neuroni nello strato nascosto e con 100000 cicli di apprendimento.
Escludendo questa rete che comporta elevati costi computazionali, si è ottenuto il minor errore di apprendimento utilizzando una rete che utilizza il training set di dati composto da sonicated e stirred, e due neuroni nell'hidden layer (ALL_100_2) con soli 43 cicli di apprendimento.
Il minor errore di relazione ingresso-uscita, come previsto, si ottiene con le reti che utilizzano un maggior numero di cicli di apprendimento.

4.2.2 Analisi dei risultati raccolti per TiO_2/Acqua

Considerando il nanofluido TiO_2/Acqua, vengono ora presi in considerazione sei diverse combinazioni che si differenziano per training set (TiO2_Therm_Cond, TiO2 Therm_Cond_sonicated e TiO2_Therm_Cond_stirred) , numero di cicli di apprendimento (100, 1000 e 5000) numero di neuroni nel secondo hidden layer (0 e 3).
Il confronto tra la conduttività termica misurata sperimentalmente e quella invece predetta dalla rete neurale viene effettuata tramite sei grafici.
Nei diagrammi sono presenti due curve che rappresentano l'output misurato e l'output previsto. Le linee grafiche sono costruite dai valori forniti dal file *previsione* generato dal programma e raggruppati per concentrazione volumetrica della particella (%vol).

Csv	Cicli di apprendimento	Second hidden layer	Nome Grafico
TiO2_Therm_Cond	100	0	TiO2_All_100_0
TiO2_Therm_Cond_sonicated	100	0	TiO2_Son_100_0
TiO2_Therm_Cond_sonicated	1000	0	TiO2_Son_1000_0
TiO2_Therm_Cond_sonicated	1000	3	TiO2_Son_1000_3
TiO2_Therm_Cond_sonicated	5000	0	TiO2_Son_5000_0
TiO2_Therm_Cond_stirred	1000	0	TiO2_Sti_1000_0

Tabella 9: Struttura e csv utilizzati per il confronto dei dati di output predetti e misurati

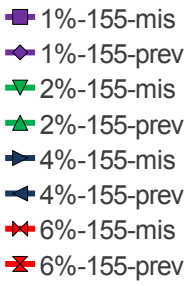

Figura 28: Legenda: concentrazione in volume - dimensione nanoparticelle - valore misurato o predetto

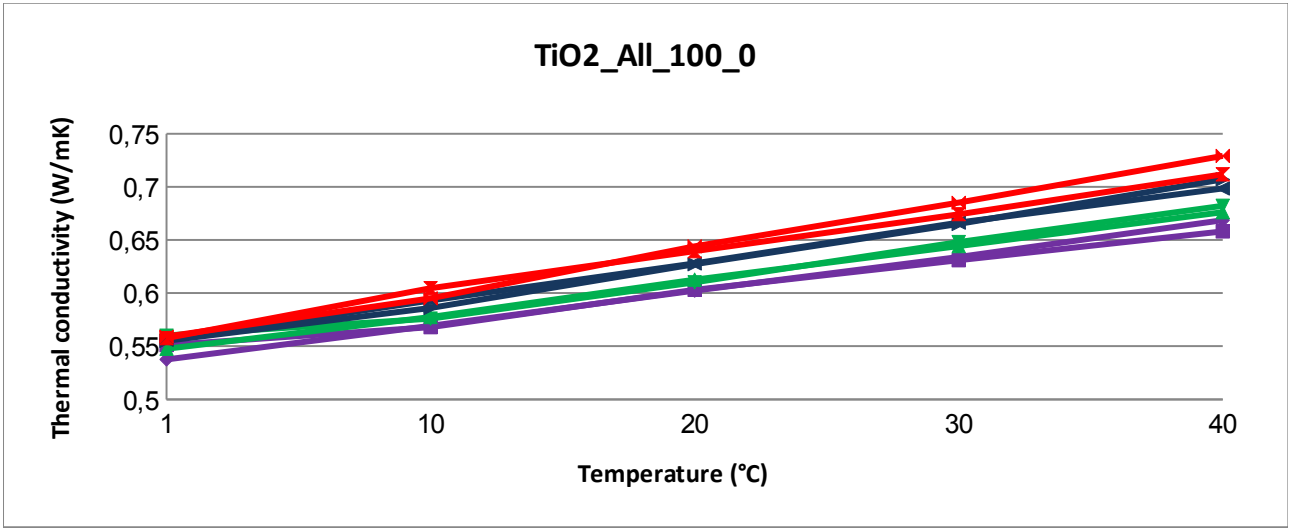

Grafico 24: Nanofluido: TiO2, data set: sonicated e stirred, cicli di apprendimento: 100, neuroni secondo strato nascosto: 0

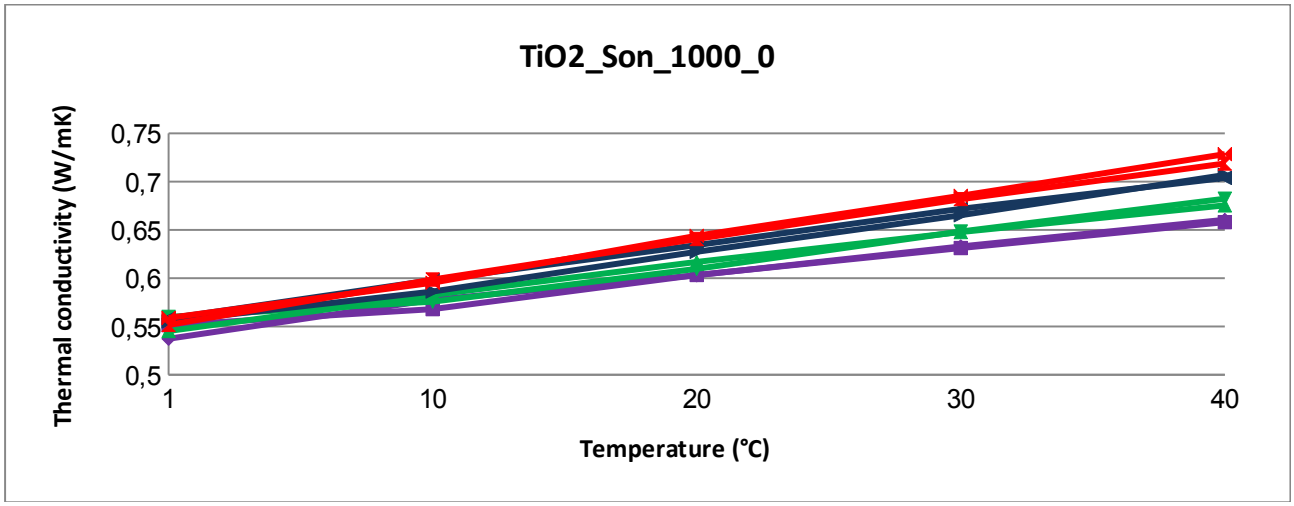

Grafico 25: Nanofluido: TiO2, data set: sonicated, cicli di apprendimento: 1000, neuroni secondo strato nascosto: 0

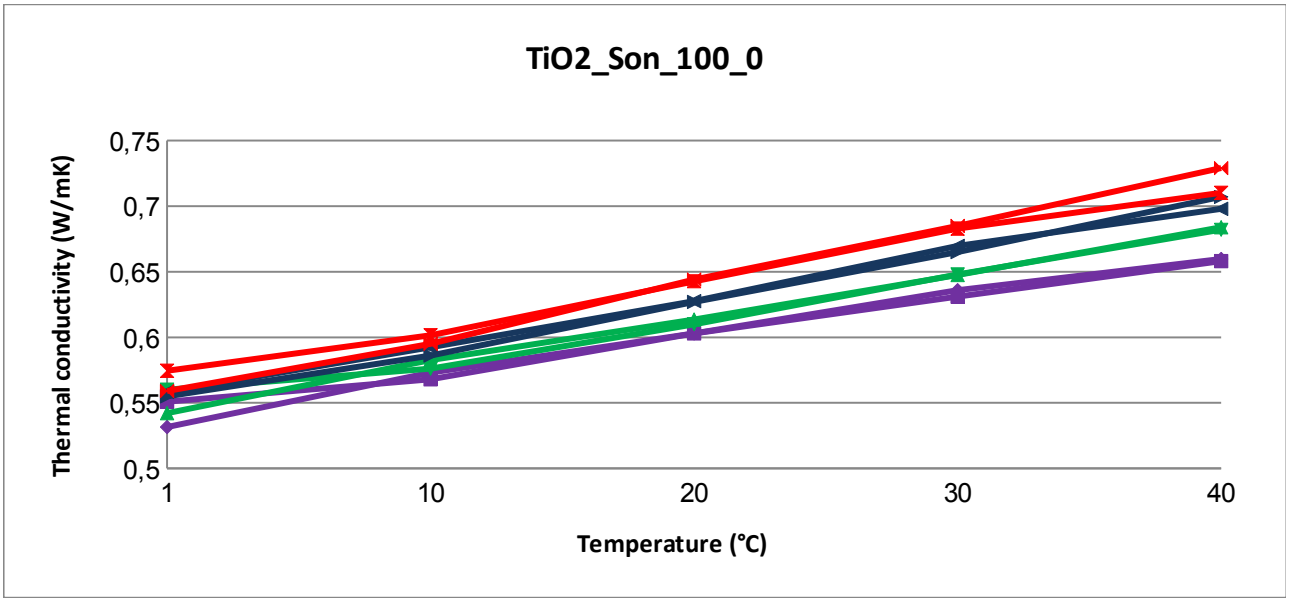

Grafico 26: Nanofluido: TiO2, data set: sonicated, cicli di apprendimento: 100, neuroni secondo strato nascosto: 0

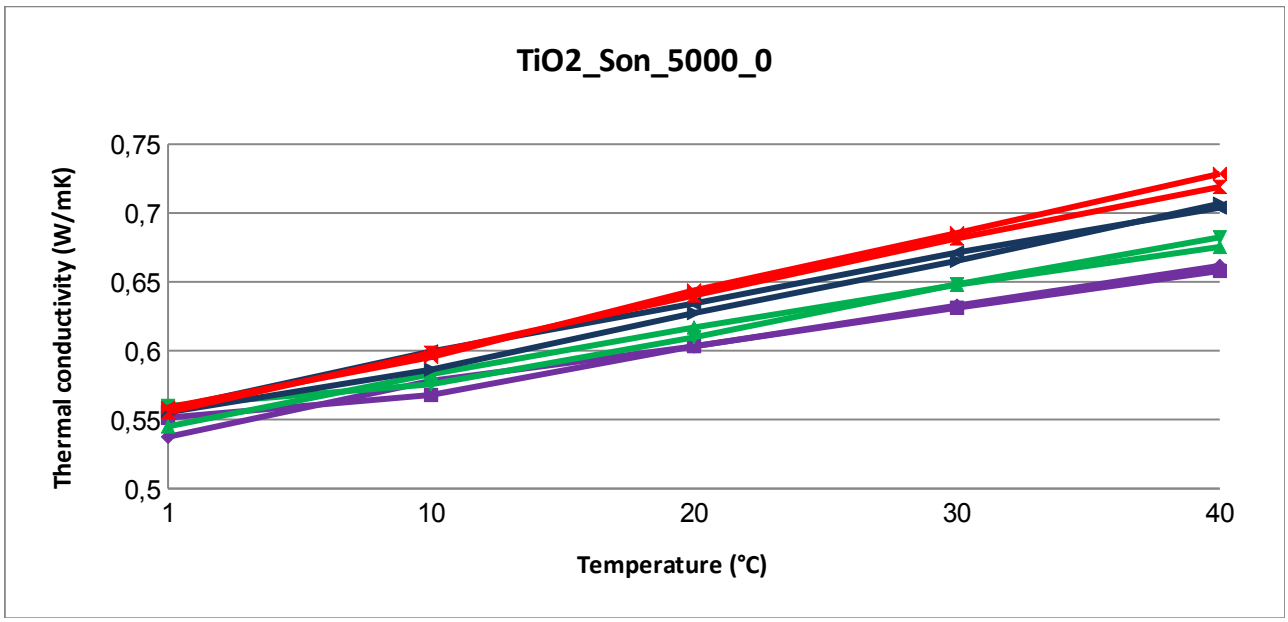

Grafico 27: Nanofluido: TiO2, data set: sonicated, cicli di apprendimento: 5000, neuroni secondo strato nascosto: 0

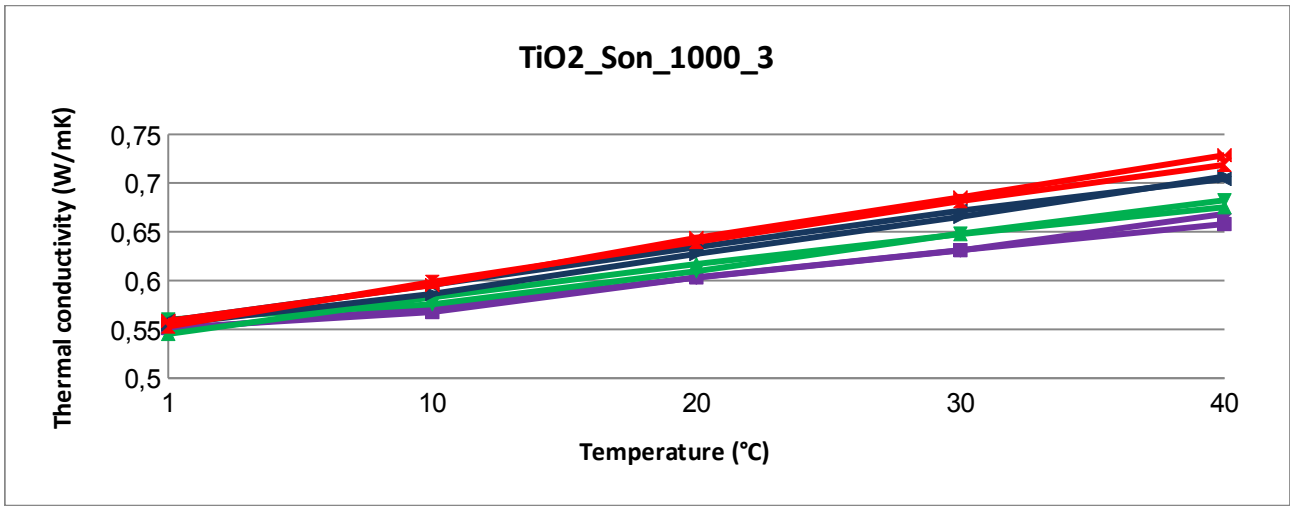

Grafico 28: Nanofluido: TiO2, data set: sonicated, cicli di apprendimento: 1000, neuroni secondo strato nascosto: 3

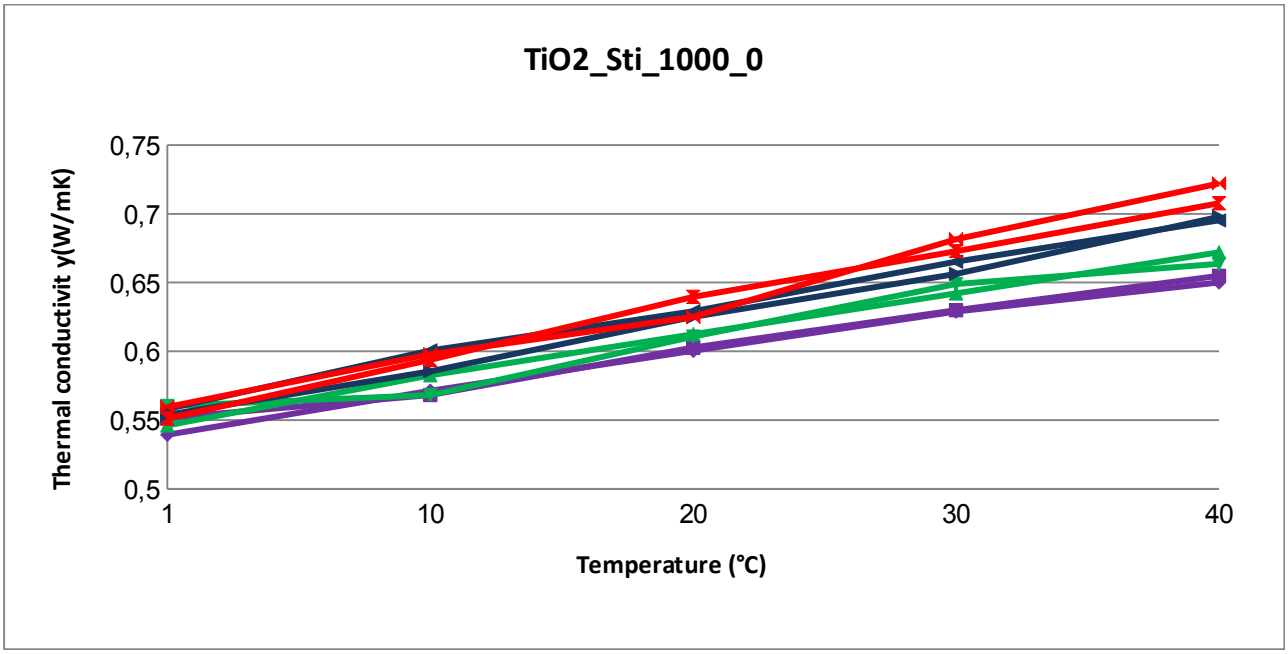

Grafico 29: Nanofluido: TiO2, data set: stirred, cicli di apprendimento: 1000, neuroni secondo strato nascosto: 0

Dai grafici 24, 25, 26, 27, 28 e 29 emerge che la conduttività termica predetta dalla rete neurale si avvicina molto ai valori misurati sperimentalmente. In tutti e sei i casi solamente per le predizioni effettuate con il 4% di concentrazione volumetrica e a basse temperature la differenza tra il valore predetto e quello misurato è maggiore rispetto alle altre predizioni restando comunque entro un margine tollerabile.

Nanofluido	Training set	Learning Cycles	Neuroni Hidden Layer	Relazione ingresso-uscita		Predizione	
				Valore	Ciclo	Valore	Ciclo
TiO2-Acqua	So+St	100	0	4,17E-005	100	5,26E-007	16
TiO2-Acqua	So+St	100	1	4,26E-005	100	3,37E-005	11
TiO2-Acqua	So+St	100	2	5,35E-005	99	8,72E-006	13
TiO2-Acqua	So+St	100	3	4,11E-005	100	3,13E-006	10
TiO2-Acqua	So	100	0	5,77E-005	100	1,00E-005	15
TiO2-Acqua	So	100	1	3,00E-005	100	1,61E-006	13
TiO2-Acqua	So	100	2	3,07E-005	100	4,39E-007	17
TiO2-Acqua	So	100	3	3,06E-005	100	1,15E-004	23
TiO2-Acqua	So	1000	0	2,58E-005	1000	1,00E-005	15
TiO2-Acqua	So	1000	1	2,58E-005	1000	1,61E-006	13
TiO2-Acqua	So	1000	2	2,66E-005	1000	4,39E-007	17
TiO2-Acqua	So	1000	3	2,59E-005	1000	6,84E-005	308
TiO2-Acqua	So	5000	0	2,58E-005	4993	1,00E-005	15
TiO2-Acqua	So	5000	1	2,56E-005	4994	1,61E-006	13
TiO2-Acqua	So	5000	2	2,57E-005	4999	4,39E-007	17
TiO2-Acqua	So	5000	3	2,52E-005	5000	6,84E-005	308

Tabella 10: Valori degli errori della relazione ingresso-uscita e di predizione. So = Sonicated, St = Stirred

Tutti i dati presenti in Tabella 10 rappresentano gli errori della relazione ingresso-uscita e gli errori di predizione per l'ultimo file trajectory di ogni struttura di rete utilizzata.

L'errore di training in tutti i casi in esame decresce all'aumentare del numero di cicli.

L'errore di predizione minimo assoluto (4.39E-007), invece, si trova a 17 cicli e, tra tutte le combinazioni analizzate, è generato dalla rete composta da 2 neuroni nel secondo strato nascosto, utilizzando 100 cicli di addestramento e un training set costituito da sonicated.

CONCLUSIONI

Il lavoro sviluppato in questa tesi si è focalizzato sullo sviluppo di una rete neurale adatta alla predizione della conduttività termica di nanofluidi e l'analisi delle capacità di predizione dei sistemi a reti neurali attraverso il metodo di cross-validation.
Questa parte di lavoro succede alla fase di misurazione dei dati di ingresso e precede la fase di previsione per valori alla rete sconosciuti.
L'utilizzo delle reti neurali risulta essere una buona soluzione in tutti quei casi in cui conoscere il modello fisico del fenomeno descritto risulta essere troppo complicato. Il campo di applicazione delle reti neurali è molto vasto, il loro utilizzo comporta però una amplia raccolta di dati necessaria per addestrare le reti.
Da questa tesi, partendo dallo studio del codice sorgente del programma nanonetworks, cercando di capire l'importanza della struttura della rete neurale, e tramite l'analisi dei risultati ottenuti si è cercato di capire quale fosse la rete neurale più adatta, in grado di fornire una accurata approssimazione del risultato di output a partire dai dati forniti come input.
I dai sperimentali, che sono la base di partenza per questo lavoro, sono stati utilizzati per istruire la rete neurale e creare una relazione tra le variabili di ingresso (concentrazione di volume della particella e temperatura) e le variabili di uscita (conduttività termica).
La struttura della rete neurale è stata fatta variare modificando il numero di neuroni del secondo strato nascosto e utilizzando un numero di cicli di addestramento diverso.
Attraverso i file di output, generati dall'esecuzione del programma, si può effettuare una prima analisi della scelta della struttura della rete. Vengono generati gli errori di training che indicano quanto il modello è capace di rappresentare la relazione input-output tra i dati forniti e gli errori di validation che consentono di capire se i valori predetti si avvicinano ai valori reali.
I grafici che rappresentano l'andamento della conduttività termica in funzione della temperatura dei nanofluidi Al_2O_3 e TiO_2 mostrano che la scelta dell'utilizzo della rete neurale per effettuare la previsione è una ottima soluzione. In tutti i grafici, al variare della struttura e utilizzando un numero di cicli di addestramento elevato, i valori predetti si avvicinano di molto ai valori reali.
Da questo lavoro si può concludere che l'impiego delle reti neurali per predire la conduttività termica dei nanofluidi è una valida alternativa ai metodi analitici, di correlazione e di simulazione.

BIBLIOGRAFIA

1: Stephen U.-S. Choi, NANOFLUID TECHNOLOGY: CURRENT STATUS AND FUTURE RESEARCH, 1998
2: Xiang-Qi Wang and Arun S. Mujumdar, A REVIEW ON NANOFLUIDS - PART I:THEORETICAL AND NUMERICALINVESTIGATIONS, 2008
3: S. Senthilraja, M. Karthikeyan and R.Gangadevi, Nanofluid applications in future automobiles: comprehensive review of existing data, 2010)
4: Hongbin Ma, Future Computer Chips Could Be CooledWith Nanofluid, 2006
5: Oronzio Manca, Yogesh Jaluria, and Dimos Poulikakos, Heat Transfer in Nanofluids, 2010
6: W. Yu, D.M. France, S.U.S. Choi, and J.L. Routbort, Review and Assessment of Nanofluid Technology for Transportation and Other Applications, 2007
7: M.Reza Azizian, Thermo physical Properties of Nanofluid, 2008
8: M. J. Assael, I. N. Metaxa, K. Kakosimos, D. Konstadinou, Thermal conductivity of nanofluids – Experimental and Theoretical,
9: Xiang-Qi Wang, Arun S. Mujumdar, Heat transfer characteristics of nanofluids: a review, 2006
10: Roberto Marmo, Intelligenza Artificiale - Reti Neurali,
11: Marco Botta, Introduzione alle Reti Neurali, 2002
12: Gianluca Baldassarre, Introduzione alle Reti Neurali,
13: Fabio Roli, Reti Neurali, 2009
14: Giovanni A. Longo, Claudio Zilio, Experimental measurement of thermophysical properties of oxide–waternano-fluids down to ice-point, 2011

SITOGRAFIA

{ 1 } http://hebb.mit.edu/courses/9.641/lectures/lecture16.pdf
{ 2 } http://www.princeton.edu/Eobdowns/bmhandouts.pdf
{ 3 } http://shark-project.sourceforge.net

RINGRAZIAMENTI

Ringrazio i miei familiari tutti che hanno condiviso le mie scelte.
Ringrazio la professoressa Monica Reggiani per avermi fatto scoprire nuovi mondi.
Ringrazio Elena Ceseracciu per il suo fondamentale aiuto fornito in questa tesi e la sua pazienza.
Ringrazio gli amici dell'università.
Ringrazio gli amici di sempre.
Ringrazio le persone che più di tutti mi sono state vicino in particolare Christian, Kry, Sara e Valentina .