

UNIVERSITÀ DI PISA  
FACOLTÀ DI SCIENZE MATEMATICHE FISICHE NATURALI  
CORSO DI LAUREA MAGISTRALE IN INFORMATICA

TESI DI LAUREA

**Un Sistema Adattivo di  
Reservoir Computing per Grafi  
Applicato in Tossicologia**

CANDIDATO

Andrea Zanelli

RELATORI

Alessio Micheli

Claudio Gallicchio

CONTRORELATORE

Dino Pedreschi

Anno Accademico 2011/2012



# Sommario

Il Reservoir Computing (RC) è un paradigma per Reti Neurali Ricorrenti (RNNs) in cui soltanto uno strato di output (il *readout*) è soggetto a training, mentre un insieme di unità ricorrenti (il *reservoir*) viene inizializzato in maniera opportuna e lasciato fissato. Semplicità ed efficienza di questo paradigma facilitano l'applicazione pratica delle RNNs, ottenendo risultati comparabili con quelli di sistemi più avanzati.

Le RNNs sono studiate per elaborare sequenze di elementi, mentre in molti ambiti è utile poter trattare strutture dati più complesse, come alberi o grafi. Per questo motivo di recente sono state introdotte le Graph Echo State Networks (GraphESNs) che estendono l'approccio del RC al dominio dei grafi e le GraphESN-NG che introducono una State Mapping Function (SMF) adattiva per migliorare le GraphESNs nei task di regressione e classificazione.

In questa tesi viene proposto un sistema basato su nuovi modelli ispirati alla linea iniziata con le GraphESN-NG, in cui particolare attenzione è posta alla valutazione qualitativa delle predizioni. Con questo scopo sono stati incrementati gli aspetti adattivi della SMF utilizzando Self-Organizing Map (SOM) supervisionata per ripartire lo spazio degli stati del reservoir e renderlo visualizzabile, mentre il training del readout viene fatto attraverso Elastic Net (una recente tecnica di regolarizzazione che combina Ridge Regression e Lasso) che permette di coniugare regolarizzazione e pruning, mettendo in evidenza le features (gli stati associati ai vertici dei grafi) più influenti per il problema in esame.

Questo sistema trova un'importante applicazione pratica nella tossicologia computazionale. In quest'ambito, infatti, oltre ad una risposta quantitativa, è necessario fornire informazioni qualitative di supporto alla predizione del modello, in modo da permettere ad un utente esperto del campo di valutarne l'affidabilità. Il lavoro proposto è stato perciò sperimentato in problemi di tossicologia computazionale ed è stato indirizzato ad arricchire le informazioni predittive con aspetti qualitativi propri del domino delle strutture molecolari.



# Indice

<b>1</b>	<b>Introduzione</b>	<b>7</b>
<b>2</b>	<b>Argomenti Preliminari</b>	<b>13</b>
2.1	Apprendimento Automatico . . . . .	13
2.1.1	Domini Strutturati . . . . .	13
2.1.2	Trasduzioni Strutturali . . . . .	18
2.1.3	Reti Neurali Ricorrenti . . . . .	19
2.1.4	Reti Neurali Ricorsive . . . . .	22
2.1.5	Kernel per Domini Strutturati . . . . .	23
2.1.6	Altri Metodi per Dati Strutturati . . . . .	24
2.1.7	Reservoir Computing . . . . .	25
2.1.8	Reservoir Computing per Domini Strutturati . . . . .	30
2.1.9	Regolarizzazione nel Reservoir Computing . . . . .	37
2.1.10	Self-Organizing Map . . . . .	41
2.2	Tossicologia Computazionale . . . . .	45
2.2.1	Quantitative Structure Activity Relationship . . . . .	47
2.2.2	Structural Alerts . . . . .	47
<b>3</b>	<b>Descrizione del Sistema Proposto</b>	<b>49</b>
3.1	State Mapping Function . . . . .	53
3.1.1	Training Supervisionato della SOM . . . . .	54
3.1.2	Disposizione delle Unità della SOM . . . . .	56
3.1.3	Funzione di Aggregazione . . . . .	58
3.2	Training del Readout . . . . .	60
3.2.1	Vantaggi del Pruning . . . . .	61
3.3	Classificazione Multi-classe e Regressione . . . . .	62
3.4	Costi Computazionali . . . . .	62
3.5	Un Modello di Confronto . . . . .	64
3.5.1	Metodo a Frammenti . . . . .	65
3.5.2	Analogie e Differenze con GraphESN-SOM . . . . .	68

<b>4 Risultati Sperimentali</b>	<b>71</b>
4.1 Configurazioni di Base . . . . .	71
4.2 SOM Supervisionata . . . . .	73
4.2.1 Risultati . . . . .	73
4.2.2 Conclusioni . . . . .	81
4.3 Valori Binari e Valori Continui . . . . .	82
4.3.1 Risultati . . . . .	83
4.3.2 Conclusioni . . . . .	85
4.4 Pruning e Regolarizzazione . . . . .	85
4.4.1 Risultati . . . . .	86
4.4.2 Conclusioni . . . . .	87
4.5 Confronto con Metodo a Frammenti . . . . .	88
4.5.1 Risultati . . . . .	88
4.5.2 Conclusioni . . . . .	90
4.6 Performance su Datasets di Tossicologia . . . . .	90
4.6.1 Risultati . . . . .	90
4.6.2 Conclusioni . . . . .	92
4.7 Analisi della Predizione . . . . .	93
<b>5 Conclusioni</b>	<b>101</b>
<b>A Datasets</b>	<b>105</b>
A.1 Bursi . . . . .	105
A.2 ISSCAN . . . . .	105
A.3 Predictive Toxicology Challenge (PTC) . . . . .	106
<b>Bibliografia</b>	<b>107</b>

# Capitolo 1

## Introduzione

I principali modelli di apprendimento automatico [1, 2] riguardano il trattamento di dati vettoriali a dimensione fissata. In molti ambiti però i dati sono rappresentati attraverso strutture complesse, come liste, alberi o grafi, le quali permettono di rappresentare anche relazioni tra le varie componenti dei dati stessi. Un esempio tra tutti è in ambito chimico dove una molecola è tipicamente rappresentata come un grafo, dove i vertici sono gli atomi della molecola, mentre gli archi rappresentano i legami chimici tra gli atomi stessi. Per gestire questo tipo di dati, chiamati *dati strutturati*, è necessario ricorrere a tecniche o metodi specifici. Una soluzione spesso utilizzata consiste nel trasformare un dato strutturato in una dato a dimensione fissata, tramite un vettore di descrittori che ne riassume proprietà e caratteristiche ritenute importanti e significative per il problema da affrontare. Questa soluzione presenta però diversi svantaggi:

- Il modello che si crea in questo modo manca di generalità: per ogni specifico problema può essere necessario scegliere un insieme di descrittori differenti poiché, a seconda del problema, alcune caratteristiche strutturali possono essere più significative di altre.
- In genere è richiesto il supporto di un esperto del dominio applicativo per la costruzione e la messa a punto del modello stesso, i.e. per la scelta dell'insieme di descrittori più adatti.
- Possono andare perse importanti informazioni sulle relazioni tra le componenti del dato legate alla struttura con cui è rappresentato.

Per questi motivi sono stati studiati, e sono oggetto di studi recenti, vari metodi di apprendimento automatico che permettono di elaborare direttamente dati strutturati.

Il lavoro svolto in questa tesi si concentra su modelli di Reti Neurali. Una Rete Neurale (NN) [3, 4] (o Rete Neurale Artificiale (ANN) per distinguerla da quella biologica) è un modello di apprendimento automatico di ispirazione

biologica composta da una rete di unità interconnesse tra loro. Le reti neurali costituiscono uno dei modelli più diffusi ed utilizzati, supportate da diversi risultati teorici [5, 6, 7], che ne garantiscono capacità computazionali, ed applicate con successo in vari problemi [8], dal riconoscimento di pattern (come il riconoscimento di caratteri manoscritti o il riconoscimento vocale) alla classificazione di composti chimici. Le Reti Neurali Ricorrenti (RNNs) [3, 9] e le Reti Neurali Ricorsive (RecNNs) [10, 11] sono classi di reti neurali che permettono di trattare dati strutturati. Le RNNs sono caratterizzate dalla presenza di cicli nelle connessioni delle unità. Questa caratteristica permette loro di elaborare sequenze di elementi, mantenendo in uno stato interno una trasformazione non lineare della storia degli elementi passati (già elaborati). Le RecNNs sono una generalizzazione delle RNNs per l’elaborazione di strutture gerarchiche (con un ordinamento topologico dei vertici) come alberi o grafi diretti aciclici. Le principali procedure di training per RNNs e RecNNs sono basate su metodi a discesa del gradiente con l’obiettivo di ridurre iterativamente l’errore di training.

Con il Reservoir Computing (RC) [12, 13, 14] è stato introdotto un paradigma per l’implementazione di RNN che propone un’alternativa all’utilizzo di algoritmi basati su discesa del gradiente, rinunciando ad alcune proprietà teoriche a favore di una maggiore efficienza del processo di training. Il RC è caratterizzato da una separazione concettuale della rete in uno strato di unità ricorrenti, chiamato *reservoir*, ed uno strato di output composto da unità non ricorrenti, il *readout*. Il punto di forza, in termini di efficienza, sta nel fatto che il reservoir non è allenato, ma è lasciato fissato dopo un’opportuna inizializzazione [15], mentre l’unica parte soggetta a training è il readout lineare. Le Echo State Networks (ESNs) [16, 17] sono uno dei principali modelli di RC e, nonostante la loro semplicità, possono raggiungere performance eccellenti e in alcuni compiti hanno portato a risultati anche superiori allo stato dell’arte [12, 17]. Di recente sono state introdotte le Graph Echo State Networks [18] che estendono le ESNs all’elaborazione di dati strutturati conservandone i punti di forza di semplicità ed efficienza dell’approccio. Inoltre, sotto un’opportuna inizializzazione del reservoir, le GraphESNs possono trattare direttamente strutture sia cicliche che acicliche e sia grafi diretti che indiretti.

Le GraphESNs producono, in modo naturale, un output in corrispondenza di ogni nodo di una struttura di input, implementando una *trasduzione struttura-struttura* [11, 18]. Molte applicazioni però riguardano problemi di classificazione o regressione in cui una struttura di input (con dimensioni e topologia variabili) dev’essere mappata in un singolo valore di output. In questo caso si parla di *trasduzione struttura-elemento*. Per esempio, nell’ambito della tossicologia il problema consiste nel classificare un composto chimico come tossico o non tossico, restituendo in output un valore che rappresenta una delle due classi, dato in input un grafo che rappresenta il composto. Per produrre un singolo output per una qualunque struttura di

input viene utilizzata una funzione detta State Mapping Function (SMF) [19, 18] che mappa l’insieme di stati prodotti dal reservoir per ogni vertice della struttura di input in un unico stato a dimensione fissata, da cui il readout ricava la risposta finale. Le SMFs classiche consistono nel prendere semplicemente lo stato di un nodo radice oppure di utilizzare metriche fissate per combinare l’insieme di stati, ad esempio prendendone la media dei valori. La prima soluzione risulta adatta solamente nel caso di strutture gerarchiche [10], dove un nodo radice è definito a priori, mentre la seconda non tiene conto che ci possono essere stati più influenti di altri per il problema, a cui potrebbe essere data una maggiore importanza. L’implementazione di SMFs avanzate costituisce un punto determinante per migliorare le prestazioni nell’ambito del RC per domini strutturati, e per questo è oggetto di recentissimi studi [20, 21]. Nel modello GraphESN-NG [21] è stata realizzata una SMF basata su una ripartizione dello spazio degli stati del reservoir guardando ai dati di training, che porta ad un incremento delle prestazioni rispetto a GraphESN la quale utilizza la media degli stati.

In questo contesto si colloca il lavoro svolto nella tesi. L’obiettivo è di realizzare un sistema basato su modelli di Reservoir Computing per grafi, che implementano trasduzioni struttura-elemento, estendendo GraphESN attraverso una nuova SMF in grado di fornire elementi per un’analisi qualitativa della risposta. Quest’ultimo punto infatti manca in GraphESN e negli altri modelli basati su essa. Difatti, nonostante tali modelli siano in grado di generare risultati molto accurati, è quasi impossibile spiegare come e perché tale risultato è stato generato, per cui risulta molto difficile interpretare la risposta restituita dal sistema, quindi stabilirne un grado di attendibilità. In alcuni ambiti questa è però una cosa indispensabile. In tossicologia si utilizzano sistemi di apprendimento automatico per valutare l’impatto e il grado di pericolosità che un composto chimico può avere sull’ambiente e sulla salute degli esseri umani, una questione estremamente sensibile e di indiscutibile importanza. Tali sistemi, che devono essere in grado di trattare dati strutturati con cui vengono rappresentati i composti chimici, vengono utilizzati come supporto nell’analisi di utenti esperti. A questo scopo è indispensabile che il sistema utilizzato fornisca informazioni qualitative in supporto ad una risposta quantitativa, che permettano all’utente esperto di valutare l’affidabilità della predizione. Per questi motivi il lavoro della tesi è stato indirizzato ad una sperimentazione specifica in ambito tossicologico. Inoltre, sempre in ambito tossicologico, sono recenti e molto importanti alcune posizioni dell’Unione Europea che incentivano l’utilizzo di modelli *in silico* (di cui fanno parte i modelli predittivi di apprendimento automatico) per valutazioni di nuovi composti chimici, alternativi alla sperimentazione *in vivo* (i.e. su animali): dal 2013 sono proibite, a livello europeo, le sperimentazioni di prodotti cosmetici su animali<sup>1</sup>, mentre il regolamento

---

<sup>1</sup><http://ec.europa.eu/consumers/sectors/cosmetics/documents/directive/>

REACH (Registration, Evaluation and Authorisation of CHemicals) del 2007 promuove l’innovazione nella valutazione di sostanze chimiche e prevede l’utilizzo di metodi alternativi ai metodi *in vivo*<sup>2</sup>. Tutto questo ha portato naturalmente ad un crescente interesse verso sistemi predittivi per la valutazione di composti chimici. Il regolamento REACH, in particolare, definisce un insieme di regole per la validazione di modelli predittivi, riferendosi ai principi definiti dall’OECD (Organisation for Economic Co-operation and Development), riassumibili in cinque caratteristiche che un modello deve avere [22]: (1) un *endpoint* definito (e.g. classificazione del composto chimico come tossico o non tossico), (2) un algoritmo non ambiguo, (3) un dominio di applicabilità definito, (4) una appropriata valutazione del modello tramite misure di *fitting* del training set, robustezza e capacità di generalizzazione e (5) se possibile, una interpretazione meccanicistica. Il sistema proposto in questa tesi è stato quindi orientato anche alla realizzazione di modelli validi seguendo tali principi.

Per fornire elementi di analisi della risposta vengono impiegate diverse tecniche. Viene introdotta una nuova SMF che permette di ottenere, in modo congiunto, due importanti punti: una rappresentazione grafica di un modello allenato che ne sintetizza il funzionamento, ed un sistema che associa un singolo peso, che guida la predizione, ad ogni vertice di un grafo. Il secondo punto permette di visualizzare sul grafo stesso quali sono le componenti che di più contribuiscono alla risposta finale. Attraverso un pruning delle connessioni del readout si ottiene una semplificazione dell’analisi della predizione riducendo il numero di elementi che determinano la risposta del sistema. Per la realizzazione della SMF viene utilizzata una Self-Organizing Map (SOM) [23], allenata in modo supervisionato, per ripartire lo spazio degli stati del reservoir, continuando nella linea iniziata con GraphESN-NG. La SOM ha la notevole caratteristica di poter essere facilmente visualizzata in una mappa bi-dimensionale, permettendo di rappresentare graficamente il modello allenato. D’altro lato, il training supervisionato della SOM permette di costruire una ripartizione significativa rispetto al problema e non solo basata sulla sintassi dei grafi. Questo porta, inoltre, ad avere una procedura particolarmente innovativa, per gli aspetti supervisionati, nell’ambito delle SMFs. Il pruning è fatto utilizzando Elastic Net [24] per il training del readout. Tale tecnica è una combinazione dei metodi di regolarizzazione Ridge Regression [25] e Lasso [26] e permette di coniugare *pruning* (annullamento dei pesi) e *shrinking* (restringimento dei pesi) in un’unica procedura di training.

La tesi è così suddivisa: nel capitolo 2 vengono introdotte le nozioni e i concetti fondamentali che saranno utilizzati nel seguito, descrivendo anche altri metodi per dati strutturati. Nel capitolo 3 è descritto il sistema proposto in questa tesi, presentandone e discutendone i vari aspetti. Nel

---

<sup>2</sup>[http://ec.europa.eu/environment/chemicals/reach/reach\\_intro.htm](http://ec.europa.eu/environment/chemicals/reach/reach_intro.htm)

capitolo 4 sono riportati i risultati di una serie di prove sperimentalni su problemi di tossicologia che valutano le specifiche componenti del sistema e le performance globali confrontate con risultati allo stato dell'arte. In questo capitolo viene inoltre mostrato il funzionamento del sistema attraverso un caso di esempio. Infine il capitolo 5 trae le conclusioni di questo lavoro e suggerisce alcuni possibili sviluppi futuri.



# Capitolo 2

# Argomenti Preliminari

In questo capitolo viene fatta un'introduzione sintetica degli argomenti di base utilizzati nei capitoli successivi.

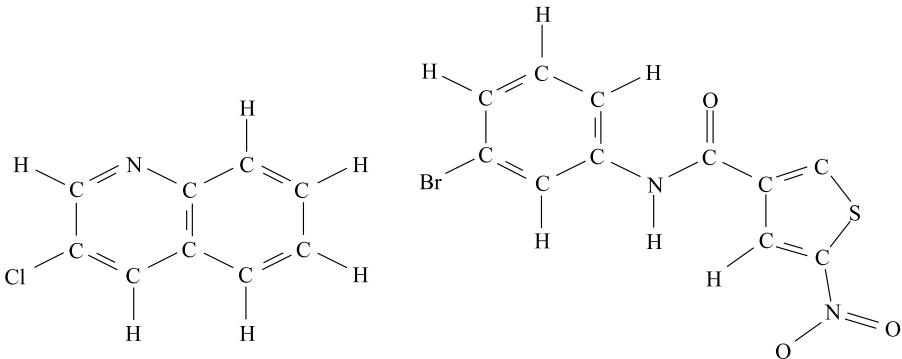
## 2.1 Apprendimento Automatico

In questa sezione vengono introdotti i concetti base di apprendimento automatico, le notazioni e le terminologie utilizzati nel resto della tesi.

### 2.1.1 Domini Strutturati

In molti ambiti applicativi [11, 10] i dati sono rappresentati in strutture complesse, come sequenze (o liste), alberi o grafi. Tali strutture permettono di rappresentare le relazioni tra le varie componenti dei dati stessi. Un esempio tra tutti è nel campo della chimica [27] (campo in cui è stato sperimentato il lavoro di questa tesi) nel quale i composti sono comunemente rappresentati come grafi indiretti [28]: i nodi del grafo sono gli atomi del composto, mentre gli archi rappresentano i legami tra atomi (Figura 2.1).

Il trattamento di strutture complesse, riconoscerle o classificarle in modo automatico, può essere fondamentale per diverse applicazioni [10]. Per esempio si hanno applicazioni in biologia e in chimica per la classificazione di composti chimici, per analisi del DNA, per analisi di relazioni quantitative struttura-attività (QSAR) o relazioni quantitative struttura-proprietà (QSPR), nell'elaborazione del linguaggio naturale per supporto al parsing, per la disambiguazione semantica o per la ricerca di patterns o strutture nel testo, ma anche nel ragionamento automatico (manipolazione dei termini logici, supporto alla dimostrazione automatica di teoremi), nel ragionamento spaziale e geometrico (robotica, rappresentazione di oggetti nello spazio, animazione di figure), ed in ogni altra applicazione dove vi è la necessità di utilizzare o manipolare questo tipo di strutture dati.



**Figura 2.1.** In chimica i composti vengono comunemente rappresentati con grafi indiretti.

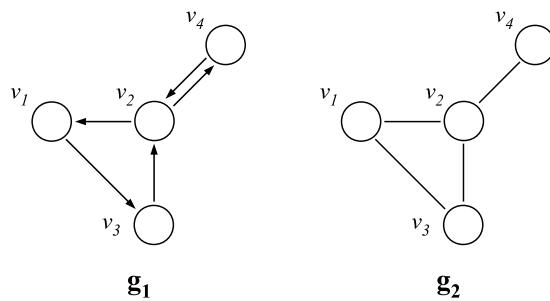
Nelle sezioni successive verranno descritti alcuni modelli di apprendimento automatico in grado di gestire dati strutturati, come la classe delle Reti Neurali Ricorrenti o le Reti Neurali Ricorsive. Qui sotto viene formalizzato il dominio dei grafi (diretti e indiretti) etichettati. Le altre strutture tipiche (sequenze e alberi) possono essere facilmente ricondotte a casi particolari di grafi diretti. Nel resto della tesi parlando di strutture o di domini strutturati faremo riferimento a questo particolare dominio.

## Il Dominio dei Grafi

Definiamo l'insieme dei grafi con vertici etichettati in uno spazio vettoriale. Un *grafo*  $\mathbf{g}$  è una coppia  $(V(\mathbf{g}), E(\mathbf{g}))$ , dove:

- $V(\mathbf{g})$  denota l'insieme dei *vertici* di  $\mathbf{g}$ ,
- $E(\mathbf{g}) = \{(u, v) : u, v \in V(\mathbf{g})\}$  denota l'insieme degli *archi* di  $\mathbf{g}$ .

In Figura 2.2 sono raffigurati un grafo diretto e un grafo indiretto. In un



**Figura 2.2.** Rappresentazione grafica di un grafo diretto  $\mathbf{g}_1$  e di un grafo indiretto  $\mathbf{g}_2$ . Entrambi i grafi hanno 4 vertici. Differente è il numero di archi:  $\mathbf{g}_1$  ne ha 5, mentre  $\mathbf{g}_2$  ha 4 archi.

grafo *diretto*  $\mathbf{g}$  ogni arco  $(u, v) \in E(\mathbf{g})$  ha una *direzione* da  $u$  a  $v$ , *uscente* dal vertice  $u$  ed *entrante* nel vertice  $v$ . Dato un vertice  $v \in V(\mathbf{g})$  di un grafo diretto  $\mathbf{g}$  denotiamo con:

- $\mathcal{P}(v) = \{u \in V(\mathbf{g}) : (u, v) \in E(\mathbf{g})\}$  l'insieme dei *predecessori* di  $v$ ,
- $\mathcal{S}(v) = \{u \in V(\mathbf{g}) : (v, u) \in E(\mathbf{g})\}$  l'insieme dei *successori* di  $v$ ,
- $\mathcal{N}(v) = \mathcal{P}(v) \cup \mathcal{S}(v)$  l'insieme dei *vicini* di  $v$ .

In un grafo *indiretto*  $\mathbf{g}$  l'insieme degli archi  $E(\mathbf{g})$  è composto da coppie non ordinate di vertici, ovvero gli archi non hanno una direzione. In questo caso se  $(u, v) \in E(\mathbf{g})$  diremo che i vertici  $v$  ed  $u$  sono *adiacenti* l'uno all'altro. L'insieme dei vicini di un vertice  $v \in V(\mathbf{g})$  per un grafo indiretto  $\mathbf{g}$  consiste nell'insieme di vertici adiacenti a  $v$ :

- $\mathcal{N}(v) = \{u \in V(\mathbf{g}) : \exists(u, v) \in E(\mathbf{g})\}$

Dato un vertice  $v$  di un grafo diretto  $\mathbf{g}$  si definiscono:

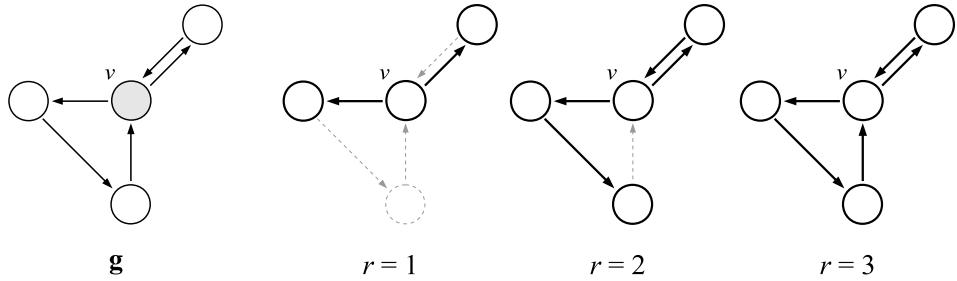
- $\text{in-degree}(v) = |\mathcal{P}(v)|$  il *grado entrante* del vertice  $v$ ,
- $\text{out-degree}(v) = |\mathcal{S}(v)|$  il *grado uscente* del vertice  $v$ ,
- $\text{degree}(v) = |\mathcal{N}(v)|$  il *grado* del vertice  $v$ .

Per un grafo indiretto  $\mathbf{g}$  è definito solamente il *grado* di un vertice  $v$  ( $\text{degree}(v)$ ) come la cardinalità dell'insieme  $\mathcal{N}(v)$ . Il *grado entrante* (in-degree), il *grado uscente* (out-degree) di un grafo diretto  $\mathbf{g}$  sono rispettivamente definiti come il massimo grado entrante e il massimo grado uscente dei vertici di  $\mathbf{g}$ . Il *grado* di un grafo  $\mathbf{g}$  è il massimo grado dei vertici di  $\mathbf{g}$ .

Dato un grafo  $\mathbf{g}$ , un *cammino* di lunghezza  $L$  da un vertice  $v$  ad un vertice  $u$  è una sequenza di vertici  $\langle v_0, v_1, \dots, v_L \rangle$ , dove:

- $v_i \in V(\mathbf{g}) \quad \forall i = 0, \dots, L$
- $(v_i, v_{i+1}) \in E(\mathbf{g}) \quad \forall i = 0, \dots, L - 1$
- $v_0 = v$  e  $v_L = u$

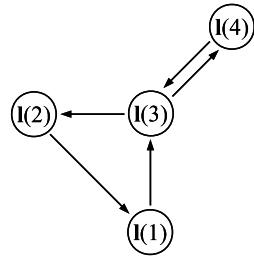
Un grafo diretto  $\mathbf{g}$  è detto *ciclico* se esiste un cammino  $\langle v_0, v_1, \dots, v_L \rangle$  in  $\mathbf{g}$  tale che  $v_0 = v_L$  e  $L > 0$ . In modo analogo, un grafo indiretto è detto *ciclico* se esiste un cammino  $\langle v_0, v_1, \dots, v_L \rangle$  in  $\mathbf{g}$  tale che  $v_0 = v_L$ ,  $L > 0$  e i vertici del cammino sono tutti distinti. Un grafo è detto *acyclico* se non è ciclico. Dato un vertice  $v$  di un grafo  $\mathbf{g}$ , si definisce l'*intorno di raggio*  $r$  del vertice  $v$  come il sottografo di  $\mathbf{g}$  ottenuto dall'insieme di tutti i vertici raggiungibili con un cammino di lunghezza  $L \leq r$ , a partire da  $v$ , e di tutti gli archi che fanno parte di un tale cammino. In Figura 2.3 è rappresentato un esempio di intorni di vario raggio per un vertice  $v$  nel caso di un grafo diretto.



**Figura 2.3.** Sulla sinistra, un grafo diretto  $\mathbf{g}$  con evidenziato (in grigio) un vertice  $v$ . Andando verso destra, con una linea più marcata sono rappresentati gli intorni di raggio 1, 2 e 3 del vertice  $v$ . Da notare che per  $r = 3$  l'intorno del vertice  $v$  coincide con l'intero grafo  $\mathbf{g}$ .

Siano  $N, N_U, N_R, N_Y \in \mathbb{N}$ , ad ogni vertice  $v \in V(\mathbf{g})$  di un grafo  $\mathbf{g}$  è associato un elemento in uno spazio vettoriale, detto *etichetta* del vertice  $v$ , che a seconda dei casi denotiamo con:

- $\mathbf{l}(v) \in \mathbb{R}^N$  nel caso generico,
- $\mathbf{u}(v) \in \mathbb{R}^{N_U}$ , in questo caso  $\mathbf{g}$  è detto *grafo di input*,
- $\mathbf{x}(v) \in \mathbb{R}^{N_R}$ , in questo caso  $\mathbf{g}$  è detto *grafo degli stati*,
- $\mathbf{y}(v) \in \mathbb{R}^{N_Y}$ , in questo caso  $\mathbf{g}$  è detto *grafo di output*.



**Figura 2.4.** Un esempio di grafo diretto ciclico di grado 2 con 4 vertici e 5 archi.

Con  $(\mathbb{R}^N)^\#$  si denota l'*insieme dei grafi etichettati* in uno spazio vettoriale  $\mathbb{R}^N$ , mentre con  $(\mathbb{R}^N)^{\#k}$  si denota l'*insieme dei grafi etichettati con grado al più  $k$* .

Due grafi  $\mathbf{g}_1 = (V(\mathbf{g}_1), E(\mathbf{g}_1))$  e  $\mathbf{g}_2 = (V(\mathbf{g}_2), E(\mathbf{g}_2))$  sono detti *isomorfi* se esiste una biiezione  $f: V(\mathbf{g}_1) \rightarrow V(\mathbf{g}_2)$  tale che  $(v, u) \in E(\mathbf{g}_1)$  se e solo se  $(f(v), f(u)) \in E(\mathbf{g}_2)$ . Informalmente, due grafi sono isomorfi se hanno la stessa struttura (stessi insiemi di vertici ed archi), ma possono avere etichette differenti su ogni vertice.

## Sequenze

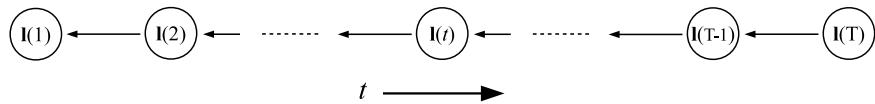
Definiamo una *sequenza*  $\mathbf{s}$  come un grafo diretto aciclico in cui:

- se  $|V(\mathbf{s})| = 0$  allora  $\mathbf{s}$  è detta *sequenza vuota*,
- se  $|V(\mathbf{s})| = 1$  allora  $\mathbf{s}$  ha grado entrante e grado uscente 0 ed è una sequenza di *lunghezza* 1,
- se  $|V(\mathbf{s})| > 1$  allora diremo che  $\mathbf{s}$  ha *lunghezza*  $L = |V(\mathbf{s})|$  e inoltre:
  - esiste uno ed un solo  $v_I \in V(\mathbf{s})$  tale che  $\text{in-degree}(v_I) = 1$  e  $\text{out-degree}(v_I) = 0$ ,  $v_I$  è detto *elemento iniziale* (o *primo elemento*) della sequenza,
  - esiste uno ed un solo  $v_E \in V(\mathbf{s})$  tale che  $\text{in-degree}(v_E) = 0$  e  $\text{out-degree}(v_E) = 1$ ,  $v_E$  è detto *elemento finale* (o *ultimo elemento*) della sequenza,
  - per ogni altro  $v \in V(\mathbf{s})$ ,  $v \neq v_I$ ,  $v \neq v_E$  si ha  $\text{in-degree}(v) = 1$  e  $\text{out-degree}(v) = 1$ .

Data una sequenza  $\mathbf{s}$  introduciamo alcune notazioni aggiuntive:

- $\mathbf{s} = []$  denota la sequenza vuota,
- $\mathbf{s} = [l(1), l(2), \dots, l(L)]$  denota una sequenza di lunghezza  $L$ , dove:
  - $l(1)$  è l'elemento iniziale,
  - $l(L)$  è l'elemento finale,
  - $(v+1, v) \in E(\mathbf{s}) \quad \forall v = 1, \dots, L-1$ .
- $(\mathbb{R}^N)^L$  denota l'insieme di tutte le possibili sequenze di lunghezza  $L$  con elementi in  $\mathbb{R}^N$ .

Nel caso sia coinvolta una dimensione temporale, un *time step*  $t \in \mathbb{N}$  può essere associato ad ogni elemento della sequenza. In questo caso parleremo di *sequenza temporale* (o *serie temporale*), l'elemento iniziale della sequenza sarà l'*elemento più vecchio* mentre l'elemento finale sarà l'*elemento più recente*. In Figura 2.5 è riportato un esempio di sequenza temporale.



**Figura 2.5.** Un esempio di sequenza temporale. Da notare che in questo caso l'ordine temporale è opposto all'ordine topologico della struttura.

### 2.1.2 Trasduzioni Strutturali

Nell'elaborazione di dati strutturati siamo interessati a calcolare *trasduzioni strutturali* [11], cioè funzioni che mappano elementi di un dominio strutturato di input in uno spazio strutturato o in uno spazio vettoriale di output. Le definizioni di trasduzioni strutturali (e domini strutturati nella sezione precedente) saranno utili nel seguito per formalizzare i modelli di apprendimento automatico per domini strutturati. Lo scopo di tali modelli infatti è apprendere, in modo automatico, trasduzioni strutturali da un insieme di esempi.

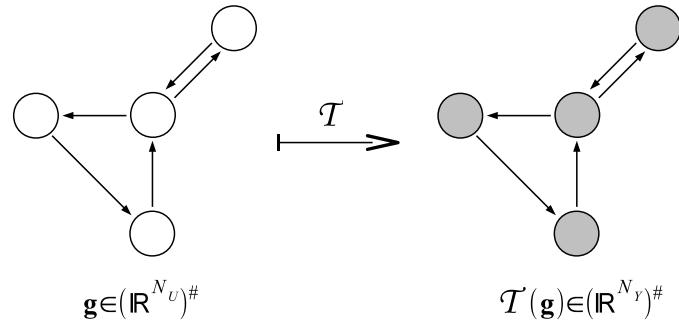
Siano  $N_U, N_Y \in \mathbb{N}$ , allora chiameremo gli insiemi  $(\mathbb{R}^{N_U})^\#$  ed  $(\mathbb{R}^{N_Y})^\#$  rispettivamente *spazio delle strutture di input* e *spazio delle strutture di output*. Distinguiamo due classi di trasduzioni strutturali: trasduzioni *struttura-struttura* e trasduzioni *struttura-elemento*. Una trasduzione struttura-struttura  $\mathcal{T}$  è una funzione che mappa una struttura di input in una struttura isomorfa di output:

$$\mathcal{T}: (\mathbb{R}^{N_U})^\# \rightarrow (\mathbb{R}^{N_Y})^\# \quad (2.1)$$

mentre una trasduzione struttura-elemento  $\mathcal{T}$  è una funzione che mappa una struttura di input in un singolo elemento vettoriale di output:

$$\mathcal{T}: (\mathbb{R}^{N_U})^\# \rightarrow \mathbb{R}^{N_Y} \quad (2.2)$$

In Figura 2.6 e 2.7 sono illustrate rispettivamente una trasduzione struttura-struttura e una trasduzione struttura-elemento. Una trasduzione strutturale

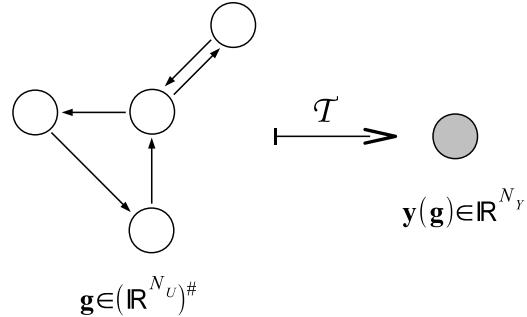


**Figura 2.6.** Una trasduzione struttura-struttura  $\mathcal{T}$  che mappa un grafo diretto di input  $g \in (\mathbb{R}^{N_U})^\#$  in un grafo di output isomorfo  $\mathcal{T}(g) \in (\mathbb{R}^{N_Y})^\#$ .

$\mathcal{T}$  è detta *adattiva* se viene appresa da un insieme di dati osservati, mentre è detta *fissata* se è definita a priori.

Una trasduzione struttura-struttura si può decomporre in due funzioni:

$$\mathcal{T} = \mathcal{T}_{\text{out}} \circ \mathcal{T}_{\text{enc}} \quad (2.3)$$



**Figura 2.7.** Una trasduzione struttura-elemento  $\mathcal{T}$  che mappa un grafo diretto di input  $\mathbf{g} \in (\mathbb{R}^{N_U})^\#$  in un singolo elemento (vettoriale) in output  $\mathbf{y}(\mathbf{g}) \in \mathbb{R}^{N_Y}$ .

dove

- $\mathcal{T}_{\text{enc}}: (\mathbb{R}^{N_U})^\# \rightarrow (\mathbb{R}^{N_R})^\#$ ,  $N_R \in \mathbb{N}$ , è detta *funzione di encoding* (o *di codifica*).
- $\mathcal{T}_{\text{out}}: (\mathbb{R}^{N_R})^\# \rightarrow (\mathbb{R}^{N_Y})^\#$  è detta *funzione di output*.

La funzione di encoding  $\mathcal{T}_{\text{enc}}$  mappa una struttura di input in una struttura isomorfa di uno spazio strutturato  $(\mathbb{R}^{N_R})^\#$ , detto *spazio strutturato degli stati* (oppure *delle features*), mentre la funzione di output  $\mathcal{T}_{\text{out}}$  mappa una struttura dello spazio strutturato degli stati in una struttura isomorfa nello spazio delle strutture di output  $(\mathbb{R}^{N_Y})^\#$ . Nel caso di trasduzione struttura-elemento una funzione  $\mathcal{X}$  è necessaria per ottenere una rappresentazione a dimensione fissata dell'intera struttura restituita dalla funzione di encoding:

$$\mathcal{T} = \mathcal{T}_{\text{out}} \circ \mathcal{X} \circ \mathcal{T}_{\text{enc}} \quad (2.4)$$

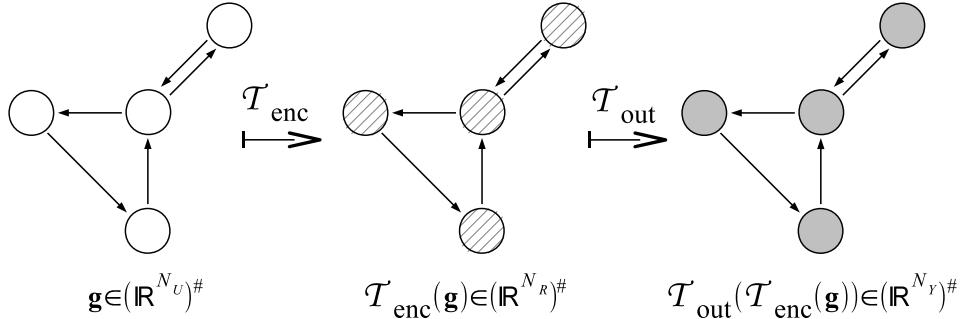
dove

- $\mathcal{T}_{\text{enc}}: (\mathbb{R}^{N_U})^\# \rightarrow (\mathbb{R}^{N_R})^\#$  è la *funzione di encoding*.
- $\mathcal{X}: (\mathbb{R}^{N_R})^\# \rightarrow \mathbb{R}^{N_S}$ ,  $N_S \in \mathbb{N}$ , è detta *state mapping function* (SMF).
- $\mathcal{T}_{\text{out}}: \mathbb{R}^{N_S} \rightarrow \mathbb{R}^{N_Y}$  è la *funzione di output*.

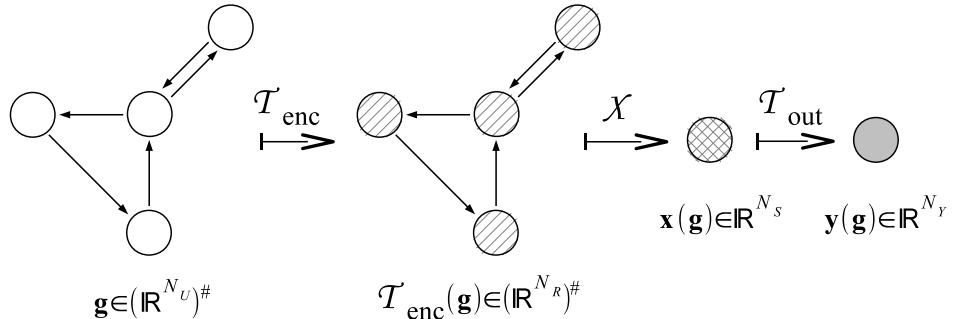
In questo caso l'insieme  $\mathbb{R}^{N_S}$  è detto *spazio degli stati* e la funzione di output si riduce ad una funzione che mappa un elemento vettoriale dello spazio degli stati in un elemento vettoriale di output. In Figura 2.8 e 2.9 sono rappresentate graficamente le decomposizioni delle trasduzioni strutturali.

### 2.1.3 Reti Neurali Ricorrenti

Le Reti Neurali Ricorrenti (RNNs) [3, 9, 12] sono una classe di modelli computazionali, di ispirazione biologica, che permettono di implementare



**Figura 2.8.** Nel caso di trasduzione struttura-struttura la funzione di encoding  $T_{\text{enc}}$  mappa il grafo di input  $\mathbf{g} \in (\mathbb{R}^{N_U})^\#$  in un grafo degli stati (isomorfo)  $T_{\text{enc}}(\mathbf{g}) \in (\mathbb{R}^{N_R})^\#$ , mentre la funzione di output  $T_{\text{out}}$  prende il grafo degli stati e lo mappa in un grafo di output.



**Figura 2.9.** Nel caso di trasduzione struttura-elemento una State Mapping Function  $X$  serve a mappare l'intero grafo degli stati  $T_{\text{enc}}(\mathbf{g}) \in (\mathbb{R}^{N_R})^\#$  in una rappresentazione a dimensione fissata  $\mathbf{x}(\mathbf{g}) \in \mathbb{R}^{N_S}$ , mentre la funzione di output mappa il singolo elemento in un elemento di output  $\mathbf{y}(\mathbf{g}) \in \mathbb{R}^{N_Y}$ .

trasduzioni adattive sul dominio delle sequenze. Una RNN è composta da una *rete* di *unità* interconnesse tra loro. Ogni unità riceve un insieme di *connessioni* di input e possiede un output sul quale invia il suo segnale di *attivazione*. Ad ogni connessione di input è associato un *peso*, ed ogni unità ha un peso aggiuntivo detto *bias*. L'attivazione di un'unità è calcolata in funzione dei segnali trasmessi nelle connessioni di input, pesati con i pesi associati alle connessioni, e del valore del bias. Le connessioni propagano l'attivazione di una unità dal suo output all'input di un'altra unità oppure in uscita alla rete determinando, in questo caso, un segnale di output. Le RNNs si differenziano dalle più utilizzate Reti Feedforward (FNN) per la presenza di cicli nella topologia della rete. La presenza di cicli ha un profondo impatto sul modello:

- Una RNN può produrre un segnale di output anche in assenza di un

segna di input, auto-alimentando la dinamica della rete attraverso le connessioni cicliche. Matematicamente, questo rende una RNN un *sistema dinamico*, mentre le FNN sono *funzioni*.

- Se riceve un segnale di input, mantiene in uno stato interno una trasformazione non lineare della storia degli elementi passati (già elaborati). In altre parole ha una *memoria dinamica* interna ed è in grado di elaborare informazioni temporali contestuali.

Nella sua forma più semplice una RNN è composta da:

- Uno strato di input di  $N_U$  unità.
- Uno strato nascosto di  $N_R$  unità *ricorrenti* (i.e. con connessioni cicliche).
- Uno strato di output di  $N_Y$  unità non ricorrenti.

Data una sequenza temporale di input  $\mathbf{s}$ , al tempo  $t$  l'elemento  $\mathbf{u}(t) \in \mathbb{R}^{N_U}$  di  $\mathbf{s}$  è dato in input alla RNN. Lo strato nascosto calcola una *funzione di transizione di stato*  $\tau: \mathbb{R}^{N_U} \times \mathbb{R}^{N_R} \rightarrow \mathbb{R}^{N_R}$  che produce lo stato  $\mathbf{x}(t) \in \mathbb{R}^{N_R}$  della rete al tempo  $t$ :

$$\mathbf{x}(t) = f(\mathbf{W}_{\text{in}}\mathbf{u}(t) + \hat{\mathbf{W}}\mathbf{x}(t-1)) \quad (2.5)$$

dove  $f$  è la *funzione di attivazione* (tipicamente una funzione non lineare di tipo sigmoidale) delle unità nascoste applicata elemento ad elemento, la matrice  $\mathbf{W}_{\text{in}} \in \mathbb{R}^{N_R \times N_U}$  contiene i pesi delle connessioni tra le unità dello strato di input e le unità dello strato nascosto (compreso il peso di bias delle unità nascoste, con un input fisso a  $+1$  in corrispondenza di tale peso) e la matrice  $\hat{\mathbf{W}} \in \mathbb{R}^{N_R \times N_R}$  contiene i pesi delle connessioni ricorrenti tra le unità dello strato nascosto. Lo strato di output calcola la funzione di output  $g_{\text{out}}: \mathbb{R}^{N_R} \rightarrow \mathbb{R}^{N_Y}$ :

$$\mathbf{y}(t) = g_{\text{out}}(\mathbf{W}_{\text{out}}\mathbf{x}(t)) \quad (2.6)$$

dove  $\mathbf{y}(t) \in \mathbb{R}^{N_Y}$  è l'output della rete al tempo  $t$  e  $\mathbf{W}_{\text{out}} \in \mathbb{R}^{N_Y \times N_R}$  contiene i pesi delle connessioni tra le unità dello strato nascosto e le unità dello strato di output (più il bias delle unità di output).

Le principali procedure di training per RNNs sono basate su metodi a discesa del gradiente con i quali si cerca di minimizzare iterativamente l'errore di training, modificando i pesi delle unità della rete. Gli algoritmi più comuni di questo tipo sono Back Propagation Through Time (BPTT) [29] e Real-Time Recurrent Learning (RTRL) [30]. Questi algoritmi di training sono però caratterizzati da alti costi computazionali [31]. Ad esempio, l'algoritmo RTRL ha costo  $\mathcal{O}(N_R^4)$  mentre BPTT riduce il costo a  $\mathcal{O}(N_R^2)$  a scapito però di una elevata occupazione di memoria.

### 2.1.4 Reti Neurali Ricorsive

Le Reti Neurali Ricorsive (RecNNs) [11, 10] generalizzano le RNNs permettendo di calcolare trasduzioni strutturali adattive su strutture gerarchiche (con un ordinamento topologico dei vertici), come alberi o grafi diretti aciclici. Nella sua forma più semplice una RecNN è composta da uno strato di input di  $N_U$  unità, uno strato nascosto di  $N_R$  unità *ricorsive* e uno strato di output di  $N_Y$  unità non ricorsive. Dato un albero di input  $k$ -ario  $\mathbf{t}$  (i.e. un albero con nodi di grado al più  $k$ ), lo strato nascosto implementa la funzione di transizione di stato  $\tau: \mathbb{R}^{N_U} \times \mathbb{R}^{kN_R} \rightarrow \mathbb{R}^{N_R}$ :

$$\begin{aligned}\mathbf{x}(n) &= \tau(\mathbf{u}(n), \mathbf{x}(\text{ch}_1(n)), \dots, \mathbf{x}(\text{ch}_k(n))) \\ &= f(\mathbf{W}_{\text{in}} \mathbf{u}(n) + \sum_{i=1}^k \hat{\mathbf{W}} \mathbf{x}(\text{ch}_i(n)))\end{aligned}\tag{2.7}$$

dove  $\mathbf{u}(n) \in \mathbb{R}^{N_U}$  è l'etichetta associata ad un nodo  $n$  dell'albero  $\mathbf{t}$ ,  $\mathbf{x}(n) \in \mathbb{R}^{N_R}$  è lo stato calcolato per il nodo  $n$ ,  $\text{ch}_i(n)$  è l' $i$ -esimo nodo figlio del nodo  $n$  per  $i = 1, \dots, k$ , con  $\text{ch}_i(n) = \text{nil}$  se l' $i$ -esimo figlio non esiste (i.e. il nodo  $n$  ha meno di  $k$  figli) e  $\mathbf{x}(\text{nil}) = \mathbf{0} \in \mathbb{R}^{N_R}$ ,  $\hat{\mathbf{W}}_i \in \mathbb{R}^{N_R \times N_R}$  è la matrice dei pesi delle connessioni ricorsive e la funzione  $f$  è la funzione di attivazione delle unità nascoste applicata elemento ad elemento. Lo strato di output calcola la funzione di output  $g_{\text{out}}: \mathbb{R}^{N_R} \rightarrow \mathbb{R}^{N_Y}$ :

$$\mathbf{y}(n) = g_{\text{out}}(\mathbf{W}_{\text{out}} \mathbf{x}(n))\tag{2.8}$$

dove  $\mathbf{y}(n) \in \mathbb{R}^{N_Y}$  è l'output corrispondente al nodo  $n$ ,  $\mathbf{W}_{\text{out}} \in \mathbb{R}^{N_Y \times N_R}$  è la matrice dei pesi delle connessioni tra le unità dello strato nascosto e le unità dello strato di output e  $g_{\text{out}}$  è la funzione di attivazione delle unità di output. Definita una numerazione dei nodi dell'albero  $\mathbf{t}$  in accordo ad una visita in profondità in ordine posticipato a partire dal nodo radice  $n_r$ , il *processo di encoding* è implementato calcolando gli stati  $\mathbf{x}(n)$  per ogni nodo  $n$  in accordo alla funzione di transizione di stato  $\tau$  dell'equazione 2.7, seguendo la numerazione dei vertici. Per ogni stato  $\mathbf{x}(n)$  si calcola l'output  $\mathbf{y}(n)$  del nodo  $n$  attraverso l'equazione 2.8. Questo permette di calcolare una trasduzione struttura-struttura ottenendo un albero di output isomorfo all'albero di input. Nel caso di trasduzione struttura-elemento la funzione di output viene applicata solamente al nodo radice dell'albero. Ciò equivale ad utilizzare una SMF che seleziona e restituisce lo stato  $\mathbf{x}(n_r)$  del nodo radice. Da notare che il funzionamento illustrato qui, ristretto al caso di alberi (per maggiore chiarezza di esposizione), può essere banalmente esteso ad ogni struttura gerarchica, come grafi diretti aciclici, con nodi di grado massimo  $k$ . Infatti anche su tali strutture si può definire un nodo radice ed una numerazione dei vertici come descritta in precedenza, grazie alla quale si può calcolare il processo di encoding ed ottenere l'output per l'intera struttura o solamente in

corrispondenza del nodo radice. Per evitare condizioni cicliche nel processo di encoding si ricorre all’assunzione di *causalità* [10]. Cioè si assume che lo stato corrispondente ad un nodo dipenda solamente dall’etichetta del nodo stesso e dagli stati corrispondenti ai nodi discendenti da esso. Questo ha implicazioni sul dominio di strutture trattabili dalle RecNNs. Le RecNNs risultano infatti inappropriate all’elaborazione di strutture cicliche o con archi indiretti. In questi casi non si ha, in generale, garantita la convergenza del processo di encoding [10]. Per superare questa limitazione sono stati proposti vari approcci. Nelle Contextual Recursive Cascade Correlation (CRCC) [32, 33, 7] viene utilizzato un approccio costruttivo per rimuovere il vincolo di causalità. Con il modello Neural Network for Graphs (NN4G) [34] si possono trattare direttamente grafi e strutture cicliche grazie ad un processo di encoding non ricorsivo. Infine, nelle Graph Neural Network (GNN) e nelle Graph Echo State Network (GraphESN) [18] si fa affidamento ad una configurazione contrattiva della funzione di transizione di stato che permette di garantire la stabilità del processo di encoding anche in presenza di strutture cicliche. Diverso è il modo in cui si ottiene la contrattività in questi due ultimi metodi. In GGN è il risultato di un processo iterativo di apprendimento, mentre in GraphESN si ricorre ad una opportuna inizializzazione dei pesi delle unità ricorsive, lasciati poi fissati in accordo ai principi del Reservoir Computing (come si vede meglio nella sezione 2.1.8).

Le RecNNs classiche sono applicate con successo in vari domini applicativi, per esempio in chemoinformatica [35, 36], nel processo del linguaggio naturale [37, 38] e nell’analisi di immagini [39]. Per il training vengono tipicamente utilizzati algoritmi basati su discesa del gradiente analoghi a quelli utilizzati nelle RNNs, come Back Propagation Through Time e Real-Time Recurrent Learning [10]. Rimangono però gli svantaggi computazionali di tali algoritmi che nel caso di RecNNs possono risultare addirittura amplificati.

### 2.1.5 Kernel per Domini Strutturati

I *metodi a kernel* sono una classe di modelli di apprendimento automatico in grado di trattare anche dati strutturati [40, 41, 42]. Siano  $\mathbf{x}, \mathbf{x}'$  due oggetti di input (per esempio due grafi) appartenenti ad uno spazio di input  $\mathcal{U}$ , l’idea di base dei metodi a kernel è di costruire una *funzione di kernel* (o semplicemente *kernel*)  $k(\mathbf{x}, \mathbf{x}')$  che restituisce una misura di similarità tra  $\mathbf{x}$  ed  $\mathbf{x}'$ . Più precisamente, una funzione di kernel  $k$  è una funzione simmetrica che definisce un prodotto interno in uno *spazio delle features*  $\mathcal{H}$  per ogni  $\mathbf{x}, \mathbf{x}' \in \mathcal{U}$ :

$$k(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle \quad (2.9)$$

dove  $\phi$  è una mappatura (non lineare) dallo spazio di input  $\mathcal{U}$  allo spazio delle features  $\mathcal{H}$ . Un kernel può essere utilizzato per estendere l’applicabilità

di un qualunque metodo lineare (convesso) di apprendimento automatico, basato sul prodotto interno tra elementi di input, al trattamento di problemi non lineari [43]. Questo viene fatto sostituendo ogni occorrenza del prodotto interno con l'applicazione del kernel  $k$ . Un metodo ottenuto in questo modo fa parte della classe dei metodi a kernel ed è in grado di lavorare sul dominio di input  $\mathcal{U}$  su cui il kernel è definito. Quindi ogni metodo a kernel è in grado di trattare dati strutturati a patto di utilizzare e definire un'apposita funzione di kernel per tali dati.

Per la definizione di un kernel valido, cioè che corrisponda effettivamente ad un prodotto interno in uno spazio delle features  $\mathcal{H}$ , la funzione  $k: \mathcal{U} \times \mathcal{U} \rightarrow \mathbb{R}$  deve soddisfare le condizioni del teorema di Mercer [44] (i.e dev'essere un kernel definito positivo). Non è quindi necessario conoscere (e definire) lo spazio delle features  $\mathcal{H}$  che può essere ad altissima dimensionalità, anche a dimensione infinita, e non è neppure richiesta una rappresentazione esplicita della funzione di mappatura  $\phi$ . Un vantaggio dei metodi a kernel è che spesso è più semplice definire una misura di similarità tra gli elementi di input per un problema di classificazione o regressione, piuttosto che proiettarli in uno spazio di features e risolvere il problema su tale spazio (soprattutto se ha dimensioni elevate o addirittura infinite). D'altro lato, uno svantaggio di questi metodi è il fatto che per ogni problema può essere richiesta la definizione di un kernel specifico e adatto al problema in esame. La funzione di kernel dev'essere quindi fissata a priori e non viene appresa dai dati di training. Un altro aspetto critico dei metodi a kernel è il costo dell'algoritmo utilizzato per calcolare la funzione di kernel, la cui progettazione richiede quindi particolare attenzione.

Le Support Vector Machines (SVMs) [45, 46] rappresentano il metodo a kernel più famoso e maggiormente utilizzato. Sono applicate con successo in vari domini applicativi, dal riconoscimento di pattern [47] alla tossicologia computazionale [48].

Una classe particolarmente rilevante di kernel per domini strutturati è rappresentata dai kernel *convoluzionali* [49]. L'idea di base è che un kernel può essere costruito in termini di kernel definiti sulle componenti del dato strutturato. Il maggior vantaggio di questo tipo di kernel è il fatto che può essere applicato ad una vasta classe di strutture. Nel caso di grafi, i kernel Marginalized [50], Optimal Assignment ed Expected Match [51] sono esempi di kernels applicati nel campo della chemoinformatica dove una molecola è rappresentata come un grafo.

### 2.1.6 Altri Metodi per Dati Strutturati

Esistono molti altri metodi e paradigmi nell'ambito dell'apprendimento automatico che permettono di trattare dati strutturati, incluse strutture dati molecolari. Nella Programmazione Logica Induttiva (PLI) [52] vengono rappresentati un dominio e relazioni tra elementi in termini di predicati della

logica del prim'ordine, e si apprendono teorie logiche dai dati attraverso un processo induttivo (i.e. una ricerca nello spazio delle ipotesi). In [53] la PLI è applicata a problemi di tossicologia computazionale con composti chimici rappresentati come grafi. Un altro interessante approccio è dato dalla classe dei metodi evolutivi, in particolare dagli algoritmi genetici, utilizzati, per esempio, in [54] per la classificazione di segmenti di proteine. Infine, nell'ambito del Graph Mining [55] svariati metodi esistono per trovare ed estrarre informazioni utili da collezioni di dati strutturati (grafo in particolare). Ad esempio in [56] uno di questi metodi (Molfea) viene utilizzato per estrarre un insieme di frammenti molecolari, da un dataset di composti chimici, utili alla predizione di attività mutagene.

### 2.1.7 Reservoir Computing

Il Reservoir Computing (RC) [12, 14, 13] è un paradigma per RNNs che ne facilita l'utilizzo pratico grazie a caratteristiche come semplicità ed efficienza. Venne introdotto contemporaneamente (e indipendentemente) con i modelli Echo State Networks (ESNs) [16] e Liquid State Machines (LSMs) [57]. È caratterizzato da una separazione concettuale tra uno strato di unità ricorrenti, chiamato *reservoir*, ed uno strato di output composto da unità non ricorrenti, *il readout*. Il reservoir riceve il segnale di input e mantiene nel suo stato una trasformazione non lineare dei segnali di input passati. Il readout calcola l'output, per un dato segnale di input, come combinazione lineare dello stato generato dal reservoir. La separazione riguarda il modo in cui le due parti sono costruite ed allenate. I pesi associati alle unità del reservoir vengono inizializzati in modo casuale rispettando opportuni vincoli (per garantire caratteristiche come la stabilità della rete), dopodiché sono lasciati fissati, mentre i pesi delle unità del readout sono gli unici soggetti a training. Questa separazione permette di evitare i tipici problemi di efficienza e convergenza degli algoritmi di training, basati su discesa del gradiente, per le RNNs. Nel caso del RC il training infatti può essere ridotto ad un problema di regressione lineare per trovare i pesi delle unità del readout, risolvibile con tecniche efficienti ed efficaci.

In questa tesi l'attenzione va sulle ESNs, descritte di seguito, ma esistono molti altri approcci e modelli interessanti legati al RC (si può vedere in [12] per i più noti): per esempio le già citate LSMs che nascono dal campo delle neuroscienze e sono state introdotte per modellare il comportamento del cervello biologico, le Extreme Learning Machines [58] che applicano il principio del RC alle reti feedforward, oppure BackPropagation Decorrelation (BPDC) [59] che introduce un nuovo efficiente algoritmo di training per RNNs sfruttando questo paradigma.

## Echo State Networks

Le ESNs sono state introdotte come strumenti ingegneristici per applicazioni tecniche, ottenendo ottimi risultati nella predizione di serie temporali, anche superiori a metodi già esistenti [16, 60, 17]. Si basano sull'osservazione che si possono ottenere ottime prestazioni allenando solamente il readout, se la matrice dei pesi del reservoir possiede certe proprietà algebriche. Più in dettaglio, un ESN è una rete neurale ricorrente composta da (Figura 2.10):

- Uno strato di input di  $N_U$  unità.
- Uno strato nascosto di  $N_R$  unità ricorrenti non lineari (il reservoir).
- Uno strato di output di  $N_Y$  unità lineari (il readout).

con connessioni dalle unità di input alle unità del reservoir, dalle unità del reservoir alle unità del readout e con connessioni ricorrenti tra le unità del reservoir stesse<sup>1</sup>. I pesi delle connessioni sono raccolti nelle seguenti matrici:

- $\mathbf{W}_{\text{in}} \in \mathbb{R}^{N_R \times N_U}$  raccoglie i pesi delle connessioni dall'input al reservoir e viene detta *matrice dei pesi di input*.
- $\hat{\mathbf{W}} \in \mathbb{R}^{N_R \times N_R}$  raccoglie i pesi delle connessioni ricorrenti del reservoir e viene chiamata *matrice dei pesi del reservoir*.
- $\mathbf{W}_{\text{out}} \in \mathbb{R}^{N_Y \times N_R}$  raccoglie i pesi delle connessioni dal reservoir al readout e viene detta *matrice dei pesi del readout* (oppure *di output*).

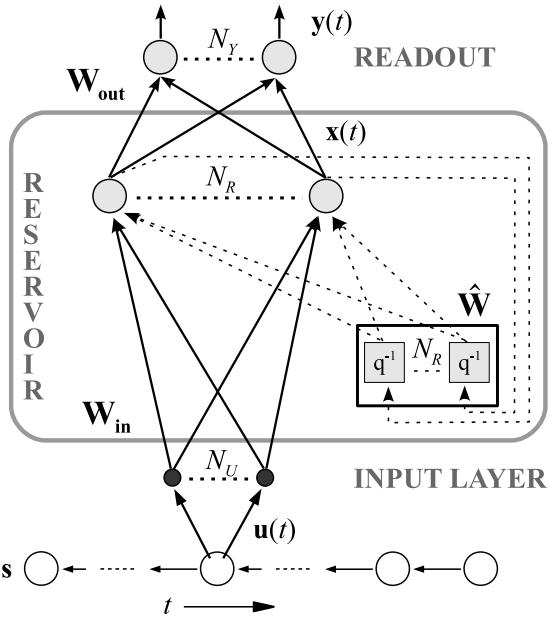
Per migliore chiarezza il bias viene omesso, qui e nelle equazioni successive. Si può infatti considerare come un input addizionale con valore fissato a +1 e una colonna dedicata nella matrice  $\mathbf{W}_{\text{in}}$ , per il bias nelle unità del reservoir, mentre si può considerare come elemento addizionale di valore fissato a +1, da concatenarsi allo stato del reservoir, ed una colonna extra nella matrice dei pesi  $\mathbf{W}_{\text{out}}$ , per il bias delle unità del readout.

Le ESNs sono applicate nell'elaborazione non lineare di serie temporali, in cui una sequenza temporale di input è trasformata in una serie temporale di output. Data quindi una sequenza temporale di input  $\mathbf{s} \in (\mathbb{R}^{N_U})^\#$ , al tempo  $t$  l'elemento  $\mathbf{u}(t) \in \mathbb{R}^{N_U}$  di  $\mathbf{s}$  è dato in input alla ESN. L'attivazione delle unità del reservoir per tale input crea lo *stato* della rete al tempo  $t$ , denotato con  $\mathbf{x}(t) \in \mathbb{R}^{N_R}$ , calcolato in modo ricorrente in accordo alla *funzione di transizione di stato*  $\tau: \mathbb{R}^{N_U} \times \mathbb{R}^{N_R} \rightarrow \mathbb{R}^{N_R}$ :

$$\begin{aligned} \tau(\mathbf{u}, \mathbf{x}) &= f(\mathbf{W}_{\text{in}}\mathbf{u} + \hat{\mathbf{W}}\mathbf{x}) \quad \text{per } \mathbf{u} \in \mathbb{R}^{N_U}, \mathbf{x} \in \mathbb{R}^{N_R} \\ \mathbf{x}(t) &= \tau(\mathbf{u}(t), \mathbf{x}(t-1)) \end{aligned} \tag{2.10}$$

---

<sup>1</sup>Nella loro forma più generale, le ESNs prevedono anche connessioni dalle unità di input direttamente a quelle del readout e connessioni ricorrenti dalle unità del readout alle unità del reservoir e tra le unità del readout stesse [16]. In questo contesto però è sufficiente riferirsi ad una forma semplificata, che permette una notazione più semplice.



**Figura 2.10.** Rappresentazione grafica di una Echo State Network.

dove  $f$  è la funzione di attivazione delle unità del reservoir applicata componente a componente. In questa tesi verrà presa la tangente iperbolica ( $\tanh$ ) che è la funzione di attivazione  $f$  tipicamente utilizzata nelle ESNs. Lo stato al tempo  $t = 0$ , cioè  $\mathbf{x}(0)$ , è chiamato *stato iniziale* della rete. Per ogni stato della rete  $\mathbf{x}(t)$ , al tempo  $t$ , il readout calcola l'output  $\mathbf{y}(t) \in \mathbb{R}^{N_Y}$  in accordo con la *funzione di output*  $g_{\text{out}}: \mathbb{R}^{N_R} \rightarrow \mathbb{R}^{N_Y}$ :

$$\mathbf{y}(t) = g_{\text{out}}(\mathbf{x}(t)) = \mathbf{W}_{\text{out}} \mathbf{x}(t) \quad (2.11)$$

**Echo State Property e Contrattività** Per il corretto funzionamento delle ESNs deve valere una proprietà riguardante il reservoir chiamata *Echo State Property* (ESP) [16]. Essenzialmente, l'effetto di uno stato passato  $\mathbf{x}(t)$  e del relativo input  $\mathbf{u}(t)$  su uno stato futuro  $\mathbf{x}(t+k)$  dovrebbe gradualmente svanire con il passare del tempo, ossia per  $k \rightarrow \infty$ , e non persistere o essere amplificato. Quindi la dipendenza con lo stato iniziale della rete dev'essere progressivamente persa con la lunghezza della sequenza che va all'infinito. In questo modo ogni stato passato prodotto dal reservoir è un *echo* degli input precedenti (da qui il nome della proprietà e del modello). Una definizione formale della ESP la si può trovare in [16] dove sono dimostrate anche una condizione necessaria ed una condizione sufficiente, entrambe riguardo proprietà algebriche della matrice dei pesi del reservoir  $\hat{\mathbf{W}}$ , riassunte di seguito. Assumiamo che le unità del reservoir abbiano la tangente iperbolica ( $\tanh$ ) come funzione di attivazione. Una condizione necessaria per la ESP è

che il raggio spettrale della matrice  $\hat{\mathbf{W}}$  sia minore o uguale a uno:

$$\rho(\hat{\mathbf{W}}) \leq 1 \quad (2.12)$$

Una condizione sufficiente è che la norma euclidea della matrice  $\hat{\mathbf{W}}$  sia minore di uno:

$$\|\hat{\mathbf{W}}\|_2 < 1 \quad (2.13)$$

Un'altra importante proprietà è la contrattività della funzione di transizione di stato  $\tau$  [15]. La funzione  $\tau$  è detta *contrattiva* se soddisfa la seguente condizione:

$$\begin{aligned} \exists C \in \mathbb{R}, 0 \leq C < 1, \text{ t.c. } \forall \mathbf{u} \in \mathbb{R}^{N_U}, \forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^{N_R} : \\ \|\tau(\mathbf{u}, \mathbf{x}) - \tau(\mathbf{u}, \mathbf{x}')\| \leq C \|\mathbf{x} - \mathbf{x}'\| \end{aligned} \quad (2.14)$$

per una qualche norma vettoriale  $\|\cdot\|$ . Si può dimostrare [15] che la contrattività della funzione di transizione di stato implica l'ESP del reservoir. Sotto l'assunzione di utilizzo della tangente iperbolica come funzione di attivazione, la contrattività in norma euclidea è garantita se  $\|\hat{\mathbf{W}}\|_2 < 1$  [15]. Questo porta anche alla condizione sufficiente per l'ESP dell'equazione 2.13.

La contrattività della funzione  $\tau$  serve a garantire, oltre alla ESP, la stabilità nelle dinamiche degli stati della rete, indipendentemente dagli altri fattori di inizializzazione. Una conseguenza di cui si deve tenere conto è che la contrattività porta le ESNs ad avere una organizzazione Markoviana degli stati prodotti dal reservoir [15, 61, 62], per la quale vengono generati stati simili per sequenze di input che condividono suffissi simili. Per questo motivo, sotto le condizioni di contrattività, è bene sottolineare che le ESNs non sono adatte a certe classi di problemi che non seguono l'assunzione di Markovianità, come mostrato sperimentalmente in [15] e [19]. D'altro lato, il reservoir di una ESN è in grado di differenziare sequenze di input in base al suffisso, in modo Markoviano, anche in assenza di training della funzione di transizione di stato. Questo le rende uno strumento in grado di trattare problemi di natura Markoviana in modo naturale e molto più efficiente di altri modelli per sequenze (come le RNN classiche in cui le connessioni ricorrenti devono essere allenate).

**Costruzione e Training** In genere il reservoir delle ESNs è composto da una grande quantità di unità (da decine a migliaia di unità) connesse in modo sparso e casuale. Questo viene fatto per cercare di produrre un insieme di dinamiche degli stati più ricco e diversificato possibile, in modo tale che il readout possa trarne maggiore vantaggio. Per costruire una ESN valida (tale che valga l'ESP) si genera in modo casuale la matrice dei pesi del reservoir  $\hat{\mathbf{W}}$  e la si riscalda in modo tale da soddisfare la condizione sufficiente dell'equazione 2.13, dividendo ogni elemento della matrice di

un fattore maggiore della sua norma euclidea. Come detto in precedenza, questo assicura anche la contrattività della funzione di transizione di stato. Come per  $\hat{\mathbf{W}}$ , anche la matrice dei pesi di input  $\mathbf{W}_{\text{in}}$  viene inizializzata in modo casuale (ma non riscalata), in genere con valori piccoli (minori di 1). La grande differenza rispetto alle RNNs classiche è che solamente le connessioni dal reservoir al readout sono soggette a training. Perciò i pesi delle altre connessioni, nelle matrici  $\hat{\mathbf{W}}$  e  $\mathbf{W}_{\text{in}}$ , sono lasciati fissati dopo l'inizializzazione.

Il training diventa così una procedura molto semplice. Dato un training set  $\{\langle \mathbf{u}(t), \mathbf{y}_{\text{target}}(t) \rangle \mid \mathbf{u}(t) \in \mathbb{R}^{N_U}, \mathbf{y}_{\text{target}}(t) \in \mathbb{R}^{N_Y}\}_{t=1}^T$  contenente  $T$  elementi di input  $\mathbf{u}(t)$  (i.e. tutti gli elementi di una sequenza temporale di input, di lunghezza  $T$ , ordinati dal più vecchio al più recente) ed altrettanti elementi di output  $\mathbf{y}_{\text{target}}(t)$  (i.e. gli elementi della sequenza target da cui apprendere), sia  $\mathbf{X} \in \mathbb{R}^{N_R \times T}$  la matrice le cui colonne sono gli stati prodotti dal reservoir  $\mathbf{x}(1), \dots, \mathbf{x}(T)$  corrispondenti agli input del training set  $\mathbf{u}(1), \dots, \mathbf{u}(T)$ , e sia  $\mathbf{Y}_{\text{target}} \in \mathbb{R}^{N_Y \times T}$  la matrice le cui colonne sono gli output nel training set  $\mathbf{y}_{\text{target}}(1), \dots, \mathbf{y}_{\text{target}}(T)$ . L'obiettivo della procedura di training è trovare valori dei pesi in  $\mathbf{W}_{\text{out}} \in \mathbb{R}^{N_Y \times N_R}$  che minimizzano l'errore quadratico:

$$\|\mathbf{W}_{\text{out}} \mathbf{X} - \mathbf{Y}_{\text{target}}\|_2^2 \quad (2.15)$$

Questo è un semplice problema di regressione lineare che può essere risolto in modo diretto utilizzando la pseudo inversa di Moore-Penrose della matrice  $\mathbf{X}$ :

$$\mathbf{W}_{\text{out}} = \mathbf{Y}_{\text{target}} \mathbf{X}^+ \quad (2.16)$$

Per mantenere bassa la dimensione dei pesi di  $\mathbf{W}_{\text{out}}$ , evitando così problemi di overfitting o di sensibilità al rumore, si possono utilizzare metodi di regolarizzazione. Un metodo tipicamente utilizzato è Ridge Regression:

$$\mathbf{W}_{\text{out}} = \mathbf{Y}_{\text{target}} \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})^{-1} \quad (2.17)$$

dove il parametro  $\lambda$  controlla il grado di regolarizzazione (più grande viene preso, minori saranno i pesi in  $\mathbf{W}_{\text{out}}$ ). Altri interessanti metodi di regolarizzazione per il RC sono descritti nella sezione 2.1.9.

Dato che, per la ESP, dopo l'elaborazione di un numero sufficientemente grande di elementi di input, la dipendenza dallo stato iniziale svanisce, tipicamente un numero  $t_w$  di stati  $\mathbf{x}(t)$ , a partire dall'inizio del training, vengono scartati siccome sono condizionati dallo stato iniziale. Solamente i rimanenti stati  $\mathbf{x}(t_w + 1), \dots, \mathbf{x}(T)$  vengono utilizzati per il training. Questo intervallo di tempo in cui gli stati vengono esclusi dal training è conosciuto come *washout* (lavaggio) della rete o transiente.

### 2.1.8 Reservoir Computing per Domini Strutturati

Data la semplicità e l'efficienza dell'approccio del Reservoir Computing e i risultati promettenti ottenuti dalle ESNs è di sicuro interesse portare questo approccio ai domini strutturati, per sfrutarne le caratteristiche migliori anche nel trattamento di dati complessi. Con questo scopo, prima con le Tree Echo State Networks [19], successivamente con le Graph Echo State Networks [18], si è esteso l'approccio del RC ai domini strutturati. Nel seguito vengono descritte le GraphESNs che estendono le ESNs al dominio dei grafi, e le GraphESNs-NG che aggiungono una SMF adattiva alle GraphESNs per ottenere migliori prestazioni nel caso di trasduzioni struttura-elemento. Il sistema proposto in questa tesi (descritto nel capitolo 3) è composto da modelli basati su GraphESN-NG.

#### Graph Echo State Networks

Le Graph Echo State Networks (GraphESNs) [18] sono reti neurali ricorsive, ispirate dall'approccio delle ESNs, per l'elaborazione di grafi. Come le ESNs sono suddivise in un reservoir di unità nascoste non lineari ed un readout di unità di output lineari, il primo fissato dopo una opportuna inizializzazione, mentre il secondo è l'unica parte ad essere allenata. Questo porta ad un procedura di training efficiente, che invece può essere un problema per le RecNNs standard. La contrattività della funzione di transizione di stato gioca un ruolo fondamentale in questo modello, infatti permette di avere stabilità nel calcolo degli stati anche in caso di cicli nella struttura di input, permettendo di elaborare grafi di ogni tipo, diretti ed indiretti, ciclici e aciclici.

Le GraphESNs generali sono architetturalmente molto simili alle ESNs (Figura 2.11). La maggiore differenza è nell'utilizzo delle più generali unità ricorsive invece che semplici unità ricorrenti, generalizzando il reservoir dal trattamento di sequenze al trattamento di strutture. Dato un insieme di grafi di input  $(\mathbb{R}^{N_U})^{\#}$ , ricordando che denotiamo con  $k$  il grado massimo che un vertice può avere su tale insieme (sezione 2.1.1), l'utilizzo di unità ricorsive implica l'introduzione di  $k$  matrici dei pesi del reservoir:

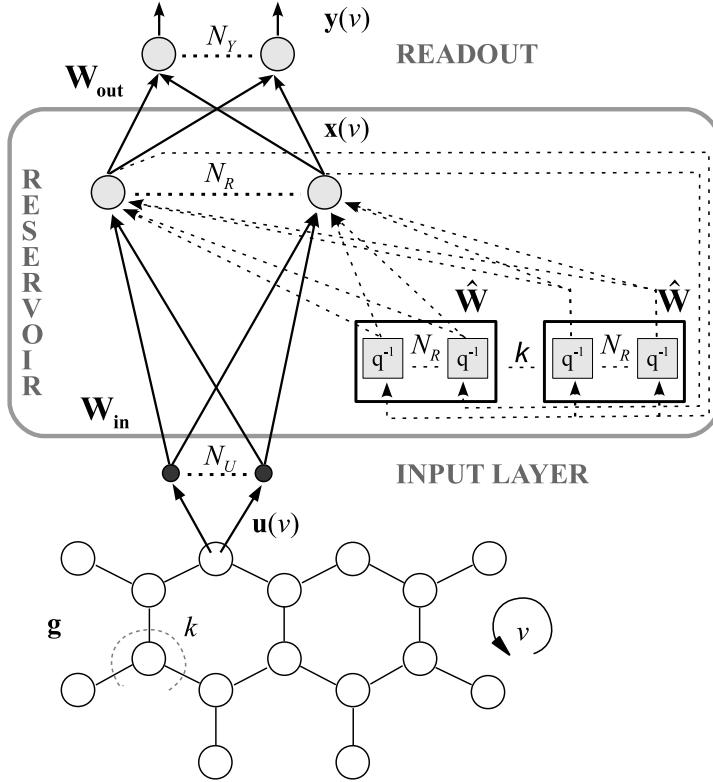
$$\hat{\mathbf{W}}_i \in \mathbb{R}^{N_R \times N_R} \quad \text{per } i = 1, \dots, k$$

una per ogni possibile vicino di un vertice (quindi una per ogni possibile stato ricorsivo). Possiamo assumere, per semplicità, che le matrici  $\hat{\mathbf{W}}_i$  siano tutte uguali, cioè trattiamo tutti i vicini di un vertice allo stesso modo<sup>2</sup>.

---

<sup>2</sup>Nel caso di grafi diretti, potrebbero essere utilizzate delle matrici  $\hat{\mathbf{W}}_i$  differenti in corrispondenza di vertici predecessori o successori, mentre nel caso di grafi posizionali, si potrebbero utilizzare matrici differenti in corrispondenza di diverse posizioni dei vertici vicini. Tutto questo però porterebbe al problema di come gestire le differenze tra diverse matrici  $\hat{\mathbf{W}}_i$  e che miglioramenti potrebbero portare. Siccome questo va oltre il contesto

Denotiamo allora tutte le matrici  $\hat{\mathbf{W}}_i$  con l'unico simbolo  $\hat{\mathbf{W}}$ .



**Figura 2.11.** Rappresentazione grafica di una Graph Echo State Network.

Una GraphESN calcola una trasduzione strutturale  $\mathcal{T}$  sul dominio dei grafi diretti. Dato un grafo di input  $\mathbf{g} \in (\mathbb{R}^{N_U})^\#$ , il reservoir calcola la funzione di encoding  $\mathcal{T}_{\text{enc}}: (\mathbb{R}^{N_U})^\# \rightarrow (\mathbb{R}^{N_R})^\#$  che mappa  $\mathbf{g}$  nello spazio strutturato degli stati  $(\mathbb{R}^{N_R})^\#$ . Per fare questo, per ogni vertice del grafo  $v \in V(\mathbf{g})$ , si deve calcolare lo stato del reservoir  $\mathbf{x}(v) \in \mathbb{R}^{N_R}$  corrispondente all'input  $\mathbf{u}(v) \in \mathbb{R}^{N_U}$ . Si definisce allora la *funzione di transizione di stato*  $\tau$ :

$$\tau: \mathbb{R}^{N_U} \times \underbrace{\mathbb{R}^{N_R} \times \dots \times \mathbb{R}^{N_R}}_k \rightarrow \mathbb{R}^{N_R} \quad (2.18)$$

che prende l'input  $\mathbf{u}(v)$ , gli stati dei vertici vicini di  $v$ , e restituisce lo stato

---

della tesi, scegliamo di trattare tutti i vertici vicini allo stesso modo.

$\mathbf{x}(v)$  nel seguente modo:

$$\begin{aligned}\tau(\mathbf{u}, \mathbf{x}_1, \dots, \mathbf{x}_k) &= \tanh\left(\mathbf{W}_{\text{in}}\mathbf{u} + \sum_{i=1}^k \hat{\mathbf{W}}\mathbf{x}_i\right) \\ \mathbf{x}(v) &= \tau(\mathbf{u}(v), \mathbf{x}(v_1), \dots, \mathbf{x}(v_k))\end{aligned}\tag{2.19}$$

Dove  $v_1, \dots, v_{|\mathcal{N}(v)|}$  è l'insieme dei vicini del vertice  $v$ . Se i vicini sono meno di  $k$  (i.e.  $|\mathcal{N}(v)| < k$ ) allora  $v_i = \text{nil}$  per  $i = |\mathcal{N}(v)| + 1, \dots, k$  e viene utilizzato lo *stato nullo* come stato dei vicini mancanti, i.e.  $\mathbf{x}(\text{nil}) = \mathbf{0} \in \mathbb{R}^{N_R}$ . Nel caso di strutture acicliche si potrebbe definire una numerazione dei vertici in accordo ad una visita in profondità in ordine posticipato a partire da un vertice radice, quindi calcolare gli stati del reservoir  $\mathbf{x}(v)$  per ogni  $v \in V(\mathbf{g})$  utilizzando in modo diretto la funzione  $\tau$ , come nell'equazione 2.19, seguendo la numerazione dei vertici. Ammettendo però anche strutture cicliche non è possibile calcolare la funzione  $\tau$  in modo diretto siccome, in questo caso, ci sono dipendenze cicliche nell'equazione 2.19 tra lo stato del vertice da calcolare e lo stato dei vertici vicini e tale equazione diventa un'equazione ricorsiva. Allora, per il *principio delle contrazioni di Banach*, se la funzione di transizione di stato  $\tau$  è contrattiva rispetto allo stato del reservoir, l'equazione ricorsiva ammette una ed una sola soluzione, che si può trovare calcolandone il punto fisso [18]. Per calcolare il punto fisso, sotto la condizione di contrattività, viene utilizzata una versione iterativa della funzione  $\tau$ . Partendo dallo stato iniziale  $\mathbf{x}_0(v) = \mathbf{0} \in \mathbb{R}^{N_R}$  per ogni vertice  $v$ , ad ogni passo iterativo  $t$  viene calcolato il nuovo stato nel seguente modo:

$$\mathbf{x}_t(v) = \tanh\left(\mathbf{W}_{\text{in}}\mathbf{u}(v) + \sum_{i=1}^k \hat{\mathbf{W}}\mathbf{x}_{t-1}(v_i)\right) \quad \forall v \in V(\mathbf{g})\tag{2.20}$$

Questo processo viene iterato fino a raggiungere una soluzione stabile per ogni vertice del grafo. In pratica, il calcolo viene stoppato quando la distanza dello stato di un vertice da una iterazione alla successiva è minore di una certa soglia  $\varepsilon$ , per ogni vertice del grafo:

$$\|\mathbf{x}_t(v) - \mathbf{x}_{t-1}(v)\|_2 \leq \varepsilon \quad \forall v \in V(\mathbf{g})\tag{2.21}$$

Il costo di questo processo su un grafo di input  $\mathbf{g}$  risulta essere  $\mathcal{O}(|V(\mathbf{g})|kN_R)$  [18], quindi scala linearmente con il numero di unità del reservoir e con il numero di vertici dei grafi di input. Tale costo è stimato sotto alcune condizioni: (a) il reservoir deve essere connesso in modo sparso, (b) i grafi di input non devono essere grafi completi ma avere un grado limitato e (c) il numero di passi iterativi per arrivare alla convergenza dev'essere un numero limitato. Tuttavia, la prima condizione è semplicemente una tipica scelta costruttiva, mentre la seconda e la terza difficilmente sono violate in problemi reali. Per esempio, nell'ambito chimico in cui questa tesi è stata

sperimentata, nel rappresentare molecole come grafi indiretti, si hanno grafi di grado difficilmente superiore a 4 e, nelle prove sperimentali effettuate, si arriva alla convergenza al massimo in qualche decina di passi iterativi.

Nel caso di trasduzioni struttura-struttura, dopo la convergenza del processo di encoding, la funzione di output  $\mathcal{T}_{\text{out}}$  è calcolata dalla funzione del readout  $g_{\text{out}}$ :

$$\mathbf{y}(v) = g_{\text{out}}(\mathbf{x}(v)) = \mathbf{W}_{\text{out}} \mathbf{x}(v) \quad (2.22)$$

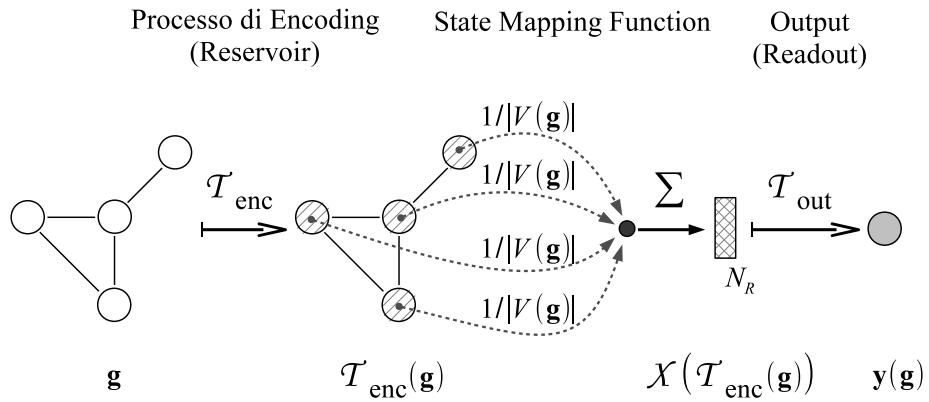
applicata nodo per nodo alla struttura nello spazio degli stati restituita dal processo di encoding. Nel caso di trasduzione struttura-elemento, un unico elemento di output  $\mathbf{y}(\mathbf{g}) \in \mathbb{R}^{N_Y}$  viene restituito per l'intero grafo di input  $\mathbf{g} \in (\mathbb{R}^{N_U})^{\#}$ , quindi la funzione di output  $\mathcal{T}_{\text{out}}$  è calcolata applicando la funzione del readout  $g_{\text{out}}$  all'output della state mapping function  $\mathcal{X}$ :

$$\mathbf{y}(\mathbf{g}) = g_{\text{out}}(\mathcal{X}(\mathcal{T}_{\text{enc}}(\mathbf{g}))) = \mathbf{W}_{\text{out}} \mathcal{X}(\mathcal{T}_{\text{enc}}(\mathbf{g})) \quad (2.23)$$

La SMF tipicamente utilizzata nelle GraphESNs è la *media degli stati*, che restituisce la media dei valori degli stati calcolati per ogni nodo del grafo di input:

$$\mathcal{X}(\mathcal{T}_{\text{enc}}(\mathbf{g})) = \frac{1}{|V(\mathbf{g})|} \sum_{v \in V(\mathbf{g})} \mathbf{x}(v) \quad (2.24)$$

In Figura 2.12 è rappresentata l'elaborazione di un grafo attraverso una GraphESN con la media degli stati. Un'altra possibile implementazione per



**Figura 2.12.** Processo di una GraphESN che mappa un grafo di input  $\mathbf{g}$  in un singolo valore di output  $\mathbf{y}(\mathbf{g})$  utilizzando la media degli stati.

$\mathcal{X}$  è lo *stato del nodo radice*, in cui viene scelto (a priori) un nodo radice per ogni grafo e viene restituito lo stato relativo a tale nodo. Quest'ultima SMF è utilizzata nel caso di alberi, o strutture gerarchiche, su cui può

essere definito un ordine topologico dei vertici ed è naturale la definizione di un nodo radice. È evidente che entrambe queste SMFs sono abbastanza rudimentali e possono portare alla perdita di molte informazioni. Infatti, come mostrato sperimentalmente in [19] e in [18], la scelta della SMF può essere determinante per ottenere buoni risultati e può dipendere dal tipo di problema. L'implementazione di SMFs avanzate è oggetto di recenti studi, ad esempio in [20] e [21] vengono introdotte due diverse SMFs con aspetti adattivi che portano a significativi miglioramenti nelle GraphESNs. Uno di questi modelli, chiamato GraphESN-NG, viene presentato nella sezione successiva.

**Contrattività ed Effetti Markoviani** Come già detto, la contrattività della funzione di transizione di stato  $\tau$  serve affinché la soluzione dell'equazione 2.19 esista e sia unica anche nel caso di cicli nella struttura di input. La funzione  $\tau$  è contrattiva rispetto allo stato del reservoir se [21]:

$$\begin{aligned} \exists C \in \mathbb{R}, 0 \leq C < 1, \text{ t.c. } \forall \mathbf{u} \in \mathbb{R}^{N_U}, \forall \mathbf{x}_1, \dots, \mathbf{x}_k, \mathbf{x}'_1, \dots, \mathbf{x}'_k \in \mathbb{R}^{N_R} : \\ \|\tau(\mathbf{u}, \mathbf{x}_1, \dots, \mathbf{x}_k) - \tau(\mathbf{u}, \mathbf{x}'_1, \dots, \mathbf{x}'_k)\| \leq C \max_{1 \leq i \leq k} \|\mathbf{x}_i - \mathbf{x}'_i\| \end{aligned} \quad (2.25)$$

per una qualche norma vettoriale  $\|\cdot\|$ . Una semplice condizione sufficiente per la contrattività in norma euclidea della funzione  $\tau$ , prendendo la tangente iperbolica come funzione di attivazione, la si può ottenere inizializzando in modo opportuno la matrice  $\hat{\mathbf{W}}$  (come per le ESNs), deve infatti valere la seguente condizione:

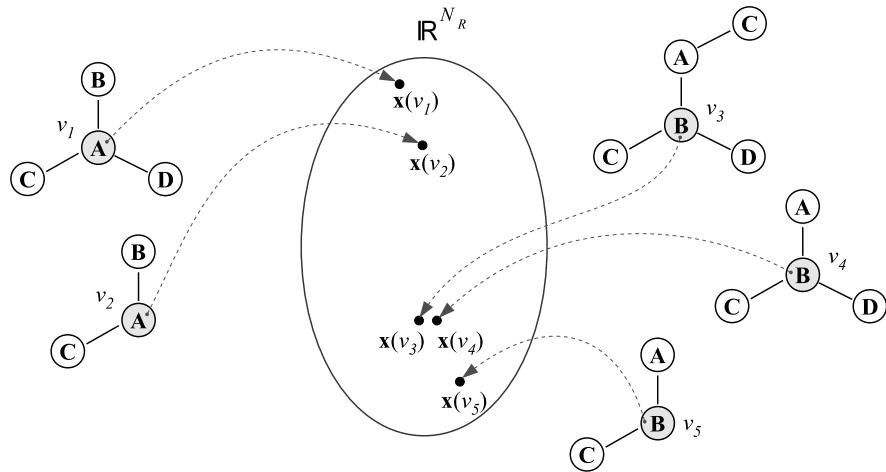
$$\|\hat{\mathbf{W}}\|_2 < 1/k \quad (2.26)$$

per la dimostrazione si può vedere [21]. Il valore  $\sigma = k\|\hat{\mathbf{W}}\|_2$ , che controlla il grado di contrattività delle dinamiche del reservoir, viene detto *coefficiente di contrattività*. Per l'equazione precedente deve valere  $\sigma < 1$ .

La contrattività della funzione  $\tau$  caratterizza le dinamiche degli stati delle GraphESNs sotto diversi punti di vista [18]:

- Grazie alla garanzia di convergenza ad una soluzione, le dipendenze cicliche nel calcolo degli stati, che rappresentano un problema nelle RecNNs standard [10], vengono facilmente gestite dalle GraphESNs.
- La garanzia di una soluzione unica implica una proprietà per le GraphESNs simile alla ESP per le ESNs. Infatti, dato un qualunque grafo di input  $\mathbf{g}$ , il grafo degli stati risultante dal processo di encoding  $\mathcal{T}_{\text{enc}}(\mathbf{g})$  dipende solamente dal grafo stesso ed è indipendente dallo stato iniziale da cui inizia il processo iterativo dell'equazione 2.20. In questo caso il washout consiste nell'applicazione della funzione di transizione di stato iterativa un numero sufficiente di volte da arrivare ad una soluzione stabile.

- Le dinamiche degli stati del reservoir sono caratterizzate da una organizzazione Markoviana che generalizza quella basata sul suffisso delle sequenze nelle ESNs. Il concetto di suffisso può essere esteso ai grafi in termini di intorno di raggio  $r$  di un vertice  $v$ . Due stati saranno tanto più simili tra loro quanto l'intorno di raggio  $r$  del vertice associato allo stato è simile, con un effetto che svanisce all'aumentare di  $r$ . In Figura 2.13 è rappresentato l'effetto dell'organizzazione Markoviana degli stati su alcuni vertici di esempio.



**Figura 2.13.** Sono raffigurati 5 vertici  $v_i$  di altrettanti grafi; le lettere rappresentano l'elemento di input associato al vertice; al centro, lo spazio degli stati  $\mathbb{R}^{N_R}$  con rappresentato lo stato del reservoir  $\mathbf{x}(v_i)$  per ognuno dei cinque vertici. Gli effetti Markoviani, dati dalla contrattività della funzione di transizione di stato, fanno sì che vertici con intorno di raggio  $r$  simile (considerando anche l'intorno di raggio 0, che è il vertice stesso) portino ad un stato tanto più simile quanto è simile il loro intorno, con un effetto che diminuisce all'aumentare di  $r$ .

Questa caratterizzazione Markoviana della funzione di transizione di stato consente di avere un sistema a stati in grado di discriminare i vertici di dati strutturati, in base al loro intorno, in modo naturale e senza training.

**Construzione e Training** Una GraphESNs viene costruita con gli stessi principi delle ESNs. Le matrici  $\mathbf{W}_{\text{in}}$  e  $\hat{\mathbf{W}}$  sono inizializzate in modo casuale,  $\hat{\mathbf{W}}$  è riscalata in modo tale che valga la condizione di contrattività dell'equazione 2.26, dopodiché sono lasciate fissate. Nel caso di trasduzione struttura-struttura il training set è composto da  $T$  esempi  $\{\langle \mathbf{g}_i, \bar{\mathbf{g}}_i \rangle \mid \mathbf{g}_i \in (\mathbb{R}^{N_U})^\#, \bar{\mathbf{g}}_i \in (\mathbb{R}^{N_Y})^\# \}_{i=1}^T$ , dove  $\mathbf{g}_i$  è l' $i$ -esimo grafo di input e  $\bar{\mathbf{g}}_i$  è il corrispondente grafo target di output (isomorfo a  $\mathbf{g}_i$ ). Nelle colonne della matrice  $\mathbf{X} \in \mathbb{R}^{N_R \times T_V}$  vengono raccolti gli stati restituiti dal reservoir per ogni vertice di ogni grafo di input (per un totale di  $T_V$

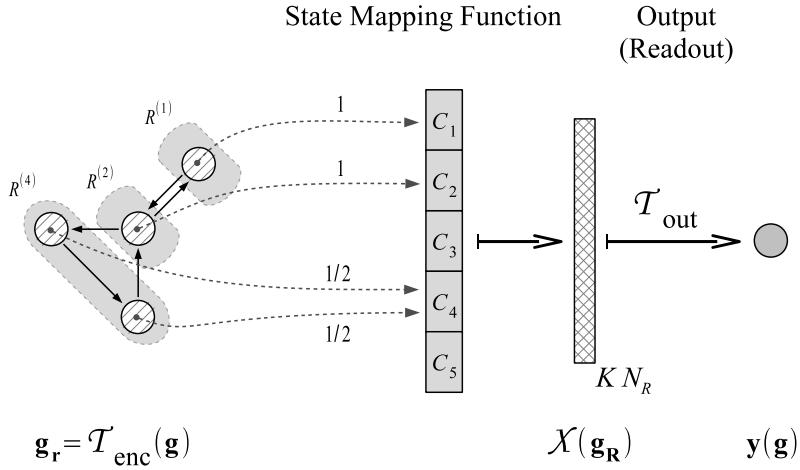
vertici), mentre la matrice  $\mathbf{Y}_{\text{target}} \in \mathbb{R}^{N_Y \times T_V}$  è costruita con le etichette di ogni vertice di ogni grafo target, arrangiati colonna per colonna. Nel caso di trasduzione struttura-elemento il training set è composto da  $T$  esempi  $\{\langle \mathbf{g}_i, \mathbf{y}_{\text{target}}(\mathbf{g}_i) \rangle \mid \mathbf{g}_i \in (\mathbb{R}^{N_U})^\#, \mathbf{y}_{\text{target}}(\mathbf{g}_i) \in \mathbb{R}^{N_Y}\}_{i=1}^T$ , dove  $\mathbf{g}_i$  è l' $i$ -esimo grafo di input e  $\mathbf{y}_{\text{target}}(\mathbf{g}_i)$  è il corrispondente valore target di output. Nelle colonne della matrice  $\mathbf{X} \in \mathbb{R}^{N_S \times T}$  vengono raccolti gli stati restituiti dalla State Mapping Function  $\mathcal{X}: (\mathbb{R}^{N_R})^\# \rightarrow \mathbb{R}^{N_S}$  per ogni grafo di input, mentre la matrice  $\mathbf{Y}_{\text{target}} \in \mathbb{R}^{N_Y \times T}$  è composta dai valori target di ogni grafo del training set disposti colonna per colonna. Il readout è quindi allenato trovando i valori dei pesi in  $\mathbf{W}_{\text{out}}$  risolvendo lo stesso problema di regressione lineare delle ESNs (dell'equazione 2.15). Da notare che, una volta calcolati tutti gli stati  $\mathbf{x}(v)$  tramite il processo di encoding, la fase di training rimane la stessa delle ESNs, conservandone tutti i vantaggi.

## GraphESN-NG

La SMF utilizzata tipicamente nelle GraphESN è la media degli stati. La limitazione di questo approccio è evidente, infatti, dato un grafo di input, ogni stato restituito dal processo di encoding viene trattato alla pari di ogni altro, aggregandoli insieme a formare l'unico stato finale. In questo modo non si tiene conto del fatto che potrebbero esserci stati più influenti per il problema, che dovrebbero avere un'importanza maggiore, ed altri insignificanti, che invece potrebbero essere ignorati.

Nelle GraphESN-NG [21] viene proposta una nuova SMF per GraphESNs, con aspetti adattivi e supervisionati, che permette di gestire in modo differente gruppi di stati simili tra loro a seconda del problema a cui viene applicata. Viene utilizzato l'algoritmo Neural Gas (NG) [63] per ripartire lo spazio degli stati in  $K$  celle di Voronoi. Dato un grafo nello spazio degli stati, ogni stato, corrispondente ad un vertice del grafo, ricade in una cella e vengono così creati  $K$  *clusters* di stati (alcuni, eventualmente, vuoti). Per ogni cluster viene calcolata la media locale degli stati, oppure viene preso il vettore nullo se il cluster è vuoto, creando una rappresentazione a dimensione fissata ( $K \cdot N_R$ ) dell'intero grafo di input, utilizzata dal readout per calcolare l'output finale (Figura 2.14). Nel readout può essere assegnato, dalla procedura di training, un diverso insieme di pesi per ogni cluster. Questo permette di trattare in modo differente insiemi di stati.

Sia la ripartizione dello spazio degli stati che l'assegnamento dei pesi del readout vengono fatti guardando ai grafi del training set, il che rende questa SMF adattiva e supervisionata. In prove sperimentali su dataset di tossicologia (in [21]) questa SMF ha portato a sensibili miglioramenti rispetto alla media degli stati utilizzata nelle GraphESN classiche.



**Figura 2.14.** Esempio del processo della SMF di GraphESN-NG per un grafo di input  $\mathbf{g}$ , con  $K = 5$ : lo spazio degli stati è suddiviso in 5 celle di Voronoi  $R^{(i)}$ ; ogni stato, del grafo risultante dal processo di encoding  $\mathbf{g}_r$ , finisce in una cella; per ogni cella si crea un cluster ( $C_1, \dots, C_5$ ) composto dagli stati caduti in tale cella; se un cluster è formato da due o più stati ne viene calcolata la media e viene creato lo stato finale  $\mathcal{X}(\mathbf{g}_r)$  concatenando le medie locali di ogni cluster (o lo stato nullo se il cluster è vuoto).

### 2.1.9 Regolarizzazione nel Reservoir Computing

Nell’ambito dell’apprendimento automatico la regolarizzazione è la tecnica utilizzata per evitare il problema dell’*overfitting* [25], quindi per diminuire l’errore di generalizzazione del modello creato ed aumentarne la robustezza rendendolo meno sensibile ad errori e piccole variazioni in input. La regolarizzazione nel Reservoir Computing (RC) [12, 64, 65, 66] può essere utile anche per scopi più specifici:

- Stabilità: nel caso di connessioni di feedback dall’output al reservoir la regolarizzazione può servire per stabilizzare la rete.
- Costi computazionali: alcuni metodi di regolarizzazione possono eliminare le connessioni del readout ritenute meno utili, diminuendo i costi computazionali.
- Interpretabilità e design del reservoir: lasciando solo le unità del reservoir ritenute più influenti si mettono in luce quali sono le features e le dinamiche più utili al problema affrontato.

Come visto nella sezione 2.1.7, il training nel RC è un problema di regressione lineare: si raccolgono gli stati prodotti dal reservoir in una matrice  $\mathbf{X} \in \mathbb{R}^{N_R \times T}$  e si apprendono i pesi delle unità del readout nella matrice  $\mathbf{W}_{\text{out}} \in$

$\mathbb{R}^{N_Y \times N_R}$  risolvendo:

$$\mathbf{W}_{\text{out}} \mathbf{X} = \mathbf{Y}_{\text{target}} \quad (2.27)$$

È importante notare che nel caso di più output ( $N_Y > 1$ ) i pesi di ogni unità (ognuna delle  $N_Y$  righe di  $\mathbf{W}_{\text{out}}$ ) possono essere allenati indipendentemente dagli altri, risolvendo un problema di regressione lineare per ognuna delle  $N_Y$  righe della matrice  $\mathbf{W}_{\text{out}}$ . Si può quindi considerare solo il caso con un output ( $N_Y = 1$ ). In questo caso la matrice  $\mathbf{W}_{\text{out}}$  e la matrice dei valori target  $\mathbf{Y}_{\text{target}}$  si riducono a vettori riga:

$$\mathbf{w}_{\text{out}} \mathbf{X} = \mathbf{y}_{\text{target}} \quad (2.28)$$

La soluzione tipica, senza regolarizzazione, è ottenuta dal metodo dei minimi quadrati:

$$\mathbf{w}_{\text{out}} = \underset{\mathbf{w}}{\operatorname{argmin}} \| \mathbf{w} \mathbf{X} - \mathbf{y}_{\text{target}} \|_2^2 \quad (2.29)$$

risolvibile in modo diretto tramite la pseudoinversa di Moore-Penrose:

$$\mathbf{w}_{\text{out}} = \mathbf{y}_{\text{target}} \mathbf{X}^+ \quad (2.30)$$

I metodi di regolarizzazione si possono suddividere in due classi: i metodi di *pruning* (eliminazione), che eliminano le connessioni meno utili annullando il relativo peso, e i metodi di *shrinking* (restringimento), che invece riducono i valori dei pesi, eventualmente portandone alcuni ad annullarsi. Esistono molti metodi, con proprietà differenti, che a seconda dello scopo per cui viene fatta regolarizzazione possono risultare più o meno adatti. Esempi di metodi di pruning [25] sono Best-Subset Selection (BSS), Forward Stepwise Selection (FSS) e Backward Deletion (BD). Nel metodo BSS, per ogni  $k \in \{0, 1, 2, \dots, N_R\}$ , viene cercato il sottoinsieme di connessioni di dimensione  $k$  che porta al minor residuo con il metodo dei minimi quadrati. Dopodiché si sceglie quale  $k$  tenere per il modello finale come quello che porta ad un minor errore di generalizzazione, oppure in modo da avere un buon trade-off tra semplicità (del modello) ed errore. Il grande problema di questo metodo è il costo computazionale molto elevato, infatti si devono risolvere un numero di problemi di regressione lineare che cresce in modo esponenziale all'aumentare di  $N_R$ , quindi è inutilizzabile con  $N_R$  grande che è la situazione tipica nel RC. Con FSS si parte con solamente la connessione del bias, e ad ogni passo  $k$  si aggiunge la connessione che porta ad un residuo minore. Questo metodo ha il vantaggio di essere molto più **efficiente** rispetto a BSS. Infatti il costo computazionale cresce in modo polinomiale all'aumentare di  $N_R$ . Si deve notare però che viene trovata la soluzione ottima localmente ad ogni passo  $k$ , per cui la soluzione finale può non essere la migliore possibile. Il metodo BD inizia con tutte le connessioni, ovvero con  $k = N_R$ , e procede all'indietro fino

a  $k = 0$  eliminando ad ogni passo la connessione che, una volta eliminata, porta ad un minor aumento del residuo. Questo metodo è analogo a FSS, con costo computazionale dello stesso ordine ed anche questo metodo può non trovare la soluzione migliore siccome effettua scelte ottime localmente.

I metodi di pruning sono utili per ottenere modelli più interpretabili e anche per aumentare l'efficienza eliminando le connessioni ritenute meno importanti. Per via della loro natura discreta (le connessioni sono mantenute o sono scartate), questi metodi soffrono però di alta varianza, portando a modelli in cui l'errore di generalizzazione non viene ridotto e risultano sensibili a piccoli errori in input [25]. I metodi di shrinking hanno invece natura più continua e per questo soffrono molto meno di tale problema e si riescono ad ottenere risultati migliori dal punto di vista della generalizzazione. Importanti metodi di shrinking sono Ridge Regression [25], Lasso [26], Elastic Net [24], Least Angle Regression [67] e Nonnegative Garrote [68]. I primi tre sono descritti nel seguito della sezione e, in particolare, Elastic Net è il metodo sfruttato nel sistema realizzato in questo lavoro.

### Ridge Regression

Ridge Regression (RR) [25] è probabilmente il metodo di regolarizzazione più utilizzato nei problemi di regressione lineare ed è ampiamente sfruttato nell'ambito del RC [12, 18]. Consiste nell'aggiungere al problema dei minimi quadrati una penalità sulla dimensione dei pesi, in modo da ottenere soluzioni con pesi più bassi:

$$\mathbf{w}_{\text{out}} = \underset{\mathbf{w}}{\operatorname{argmin}} \|\mathbf{w}\mathbf{X} - \mathbf{y}_{\text{target}}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 \quad (2.31)$$

Il parametro  $\lambda$  controlla il grado di restringimento, tipicamente scelto con procedure di validazione in modo tale da ottenere buone performance di generalizzazione. Il problema rimane di ottimizzazione non vincolata con funzione convessa come il problema dei minimi quadrati, e la soluzione può essere calcolata in modo diretto:

$$\mathbf{w}_{\text{out}} = \mathbf{y}_{\text{target}} \mathbf{X}^T (\mathbf{X} \mathbf{X}^T + \lambda \mathbf{I})^{-1} \quad (2.32)$$

Aumentando il valore del parametro  $\lambda$  i pesi vengono ristretti ma raramente vengono annullati (se non con un valore di  $\lambda$  eccessivamente alto) [25].

### Lasso

Il metodo del Lasso [26] è molto simile a RR, anche in questo caso infatti si aggiunge un termine di penalizzazione sulla dimensione dei pesi:

$$\mathbf{w}_{\text{out}} = \underset{\mathbf{w}}{\operatorname{argmin}} \|\mathbf{w}\mathbf{X} - \mathbf{y}_{\text{target}}\|_2^2 + \lambda \|\mathbf{w}\|_1 \quad (2.33)$$

La differenza fondamentale è nell'utilizzo della norma 1 invece che la norma 2 utilizzata in RR. Questa differenza porta innanzitutto ad un diverso tipo di problema da risolvere poiché la funzione obiettivo da minimizzare nell'equazione 2.33 non è più una funzione differenziabile (per via dei moduli introdotti con la norma 1). Per cui non è un problema di ottimizzazione non vincolata, ma lo si deve lo si tratta come problema di programmazione quadratica (con regione ammissibile convessa), nella forma:

$$\mathbf{w}_{\text{out}} = \underset{\mathbf{w}}{\operatorname{argmin}} \|\mathbf{w}\mathbf{X} - \mathbf{y}_{\text{target}}\|_2^2 \quad \text{tale che } \sum_{i=1}^{N_R} |\mathbf{w}_i| < t \quad (2.34)$$

Esiste però un algoritmo, basato su Least Angle Regression, che permette di trovare la soluzione del metodo del Lasso con costo dello stesso ordine del metodo RR [25]. L'altro effetto dovuto all'utilizzo della norma 1 è che, all'aumentare del parametro  $\lambda$ , i pesi vengono ristretti annullandosi molto più facilmente rispetto a RR [26], permettendo di effettuare pruning e shrinking contemporaneamente. In altri termini il metodo del Lasso tende a preferire soluzioni in cui più pesi hanno valore zero, mentre RR restituisce risultati più "smussati" in cui tutti i pesi tendono ad avere valori (piccoli) diversi da zero.

### Elastic Net

Elastic Net (EN) [24] è un metodo proposto per migliorare il metodo del Lasso in termini di performance di generalizzazione, mantenendo la proprietà di ottenere soluzioni di carattere sparso (con molti pesi a valore 0). In pratica è una combinazione dei metodi RR e Lasso, che introduce un parametro di penalizzazione sia in norma 2 che in norma 1 del vettore dei pesi:

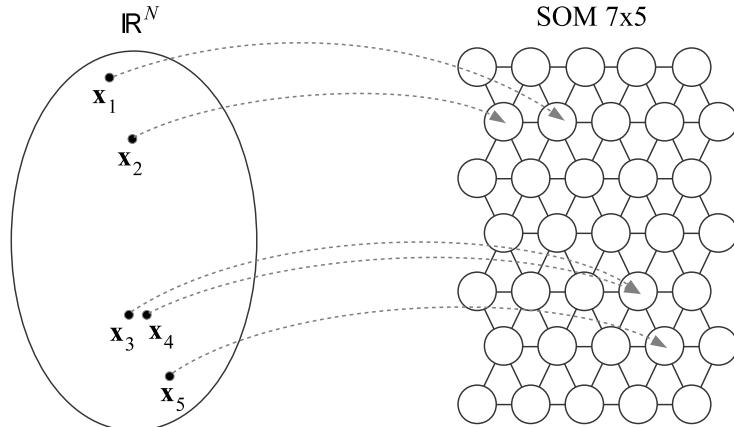
$$\mathbf{w}_{\text{out}} = \underset{\mathbf{w}}{\operatorname{argmin}} \|\mathbf{w}\mathbf{X} - \mathbf{y}_{\text{target}}\|_2^2 + 2\lambda_1\|\mathbf{w}\|_1 + \lambda_2\|\mathbf{w}\|_2^2 \quad (2.35)$$

Si noti che se il parametro  $\lambda_1$  è preso uguale a 0 il metodo si riduce a RR e, in modo analogo, con  $\lambda_2 = 0$  il metodo si riduce al metodo del Lasso. In [24] viene mostrato sperimentalmente che se RR migliora le prestazioni di generalizzazione rispetto al metodo dei minimi quadrati, EN migliora rispetto a Lasso. Inoltre, il metodo del Lasso non è ben definito fintanto che il limite alla norma 1 dei pesi non è più piccolo di un certo valore [24], e in questi casi potrebbe non trovare una soluzione. EN è invece ben definito anche in tali casi. Il comportamento di EN è quindi intermedio tra RR e Lasso, permettendo di effettuare pruning e shrinking dei pesi contemporaneamente e di ottenere buone performance in termini di generalizzazione. Come per il metodo del Lasso, la soluzione può essere calcolata in modo efficiente con un algoritmo basato su Least Angle Regression, chiamato LARS-EN [24].

### 2.1.10 Self-Organizing Map

Una Self-Organizing Map (SOM) è una rete neurale artificiale, allenata in modo non supervisionato, che viene utilizzata sia come algoritmo di vector quantization, sia come procedura per proiezione di vettori in uno spazio a bassa dimensionalità. Sebbene il termine Self-Organizing Map può riferirsi a vari tipi di approcci, in questa tesi ci riferiamo alle mappe di Kohonen [23, 69]. Le unità sono disposte su una griglia a due dimensioni<sup>3</sup> e vengono allenate utilizzando una funzione di vicinato che preserva le proprietà topologiche dei campioni di input. Questa caratteristica rende la SOM un potente strumento di analisi e visualizzazione di dati ad alta dimensionalità [70], ampiamente studiata e applicata in campi che vanno dall'ingegneria alle scienze mediche, dalla biologia all'economia [71, 72].

Una SOM definisce una mappatura da uno spazio di input  $\mathbb{R}^N$  ad una griglia regolare a due dimensioni (Figura 2.15). Ad ogni unità  $i$  è associata una *codebook* (o prototipo)  $\mathbf{m}_i \in \mathbb{R}^N$ . La griglia può essere rettangolare o esagonale, quest'ultima è tipicamente più efficace per essere visualizzata [70]. Un vettore  $\mathbf{x} \in \mathbb{R}^N$  viene confrontato con ogni  $\mathbf{m}_i$  di ogni unità  $i$ , la



**Figura 2.15.** Gli elementi di uno spazio vettoriale altamente dimensionato  $\mathbb{R}^N$  vengono mappati in una griglia (esagonale) di 35 unità, disposte su 7 righe e 5 colonne. Ogni elemento viene mappato nell'unità con la codebook più simile ad esso e, per come la SOM è costruita, elementi topologicamente vicini nello spazio vettoriale di partenza sono mappati su unità vicine.

corrispondenza migliore determina l'unità vincente  $c$ , e il vettore è *mappato* in tale unità della griglia. In questo caso si dice anche che il vettore  $\mathbf{x}$  *attiva* l'unità  $c$  della SOM, oppure anche che l'unità  $c$  della SOM *raccoglie* (o *cattura*) il vettore  $\mathbf{x}$ . Come metodo di confronto viene tipicamente utilizzata la distanza euclidea tra vettori, definendo la corrispondenza migliore come

---

<sup>3</sup>Griglie di un numero diverso di dimensioni sono ammesse, ma raramente utilizzate.

quella che restituisce la minor distanza:

$$c = \operatorname{argmin}_i \|\mathbf{x} - \mathbf{m}_i\|_2 \quad (2.36)$$

L'insieme delle codebook  $\mathbf{m}_i$ , per ogni unità  $i = 1, \dots, K$ , definisce anche una ripartizione dello spazio di input  $\mathbb{R}^N$  in  $K$  celle di Voronoi  $\mathcal{R}^{(i)} = \{\mathbf{x} \in \mathbb{R}^N \mid \|\mathbf{x} - \mathbf{m}_i\|_2 \leq \|\mathbf{x} - \mathbf{m}_j\|_2 \forall j = 1, \dots, K\}$ .

I valori delle codebook di ogni unità sono determinati da un processo di training iterativo. Partendo da un'inizializzazione casuale delle codebook  $\mathbf{m}_i(0)$  per ogni  $i$ , ad ogni passo  $t$  del processo di training, viene preso dal trainng set un vettore di input  $\mathbf{x}(t) \in \mathbb{R}^N$  che viene mappato nella SOM stabilendo l'unità vincente  $c$ . L'unità vincente e le unità che si trovano topograficamente vicine ad essa, sotto una certa distanza topografica di soglia, vengono modificate in modo tale che la codebook  $\mathbf{m}_i(t)$  si avvicini al vettore di input  $\mathbf{x}(t)$  tanto più quanto l'unità  $i$  è vicina all'unità vincente  $c$ . Questo viene fatto in accordo alla seguente regola di aggiornamento, per ogni unità  $i$ :

$$\mathbf{m}_i(t+1) = \mathbf{m}_i(t) + h_{ci}(t)[\mathbf{x}(t) - \mathbf{m}_i(t)] \quad (2.37)$$

dove  $h_{ci}(t)$  è detta *funzione di vicinato* ed è una funzione definita sui punti della griglia. Tipicamente  $h_{ci}(t) = h(\|r_c - r_i\|_2, t)$ , dove  $r_c \in \mathbb{R}^2$  e  $r_i \in \mathbb{R}^2$  sono le coordinate rispettivamente delle unità  $c$  ed  $i$  nella griglia. Un esempio di funzione  $h_{ci}(t)$ , utilizzato molto spesso, è definito in termini della funzione Gaussiana:

$$h_{ci}(t) = \alpha(t) \cdot \exp\left(-\frac{1}{2}\left(\frac{\|r_c - r_i\|_2}{\rho(t)}\right)^2\right) \quad (2.38)$$

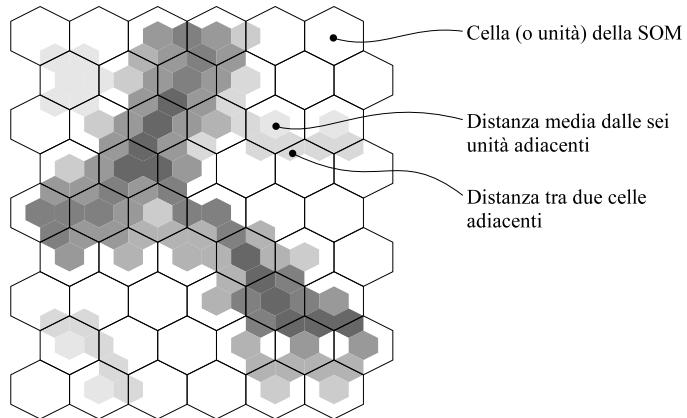
dove  $\alpha(t)$  è un fattore di scala ( $0 < \alpha(t) < 1$ ) detto *learning rate* (tasso di apprendimento), mentre il parametro  $\rho(t)$  definisce l'ampiezza della funzione di vicinato. Sia  $\alpha(t)$  che  $\rho(t)$  sono parametri che decrescono monotonicamente con il numero dei passi di training  $t$ , facendo in modo che la SOM si stabilizzi in uno stato finale dopo un numero sufficiente di passi iterativi. Infatti, la diminuzione del learning rate  $\alpha$  serve a diminuire progressivamente l'intensità dell'aggiornamento delle codebooks, mentre la diminuzione del parametro  $\rho$  restringe l'aggiornamento delle unità vicine fino ad aggiornare solamente l'unità vincente. Una buona norma pratica [73] è di suddividere il training in due fasi: la prima fase, detta *fase grezza*, in cui si definisce un ordinamento iniziale delle codebooks, la seconda, detta *fase fine*, dove i valori di ogni codebook vengono raffinati. Entrambe le fasi consistono nella stessa procedura descritta sopra ma con diversi parametri  $\alpha$ ,  $\rho$  e numero di iterazioni di training. Per la fase grezza si prendono valori di  $\alpha$  e  $\rho$  maggiori ed un numero minore di iterazioni, mentre nella fase fine si eseguono più

iterazioni con valori più piccoli di  $\alpha$  e  $\rho$ . Il risultato finale è una mappa che tende a raccogliere oggetti vicini topologicamente nello spazio di input  $\mathbb{R}^N$  in unità vicine topograficamente nella griglia [74].

Nel seguito, visualizzando una SOM, parleremo di *mappe* riferendoci a particolari visualizzazioni di griglie di unità, e di *celle* della mappa riferendoci alle unità della SOM.

### U-Matrix

Il principale punto di forza della SOM è la naturale propensione a poter essere facilmente visualizzata. Per fare questo esistono svariati metodi [70], ognuno dei quali può mettere in luce informazioni sulla mappa stessa, sui dati su cui la mappa è stata costruita oppure può servire per esaminare nuovi dati, ad esempio per scopi di classificazione. Uno dei metodi più utilizzati è la U-Matrix [75]. Con questo metodo vengono visualizzate le distanze tra unità adiacenti con valori in scala di grigio. Un esempio di U-Matrix è rappresentato in Figura 2.16.



**Figura 2.16.** Esempio di U-Matrix di una SOM: la griglia esagonale rappresenta una SOM di 9x6 unità (9 righe e 6 colonne) in cui ogni cella rappresenta una unità. Gli esagoni più piccoli, di varie tonalità di grigio, sono i valori della U-Matrix, in cui tonalità più scure indicano valori maggiori. Gli esagoni tra due celle indicano la distanza tra le due codebook associate alle due unità, mentre gli esagoni al centro delle celle indicano la distanza media rispetto alle sei celle adiacenti.

La U-Matrix permette di visualizzare le informazioni topologiche assunte dalle unità, separando aree di unità simili, distinte da zone con tonalità chiare, da altre aree attraverso zone di tonalità più scure. Le varie aree raccolgono oggetti simili dello spazio  $\mathbb{R}^N$  su cui la SOM è stata costruita.

## SOM Supervisionata

Se ad ogni vettore di input è associata una classe target, la SOM può essere allenata con algoritmi supervisionati, in modo da separare in unità diverse della SOM vettori con classi differenti. Questo approccio è utilizzato soprattutto in problemi di classificazione. Ad ogni unità della SOM è associata una classe e un nuovo vettore di input viene classificato prendendo la classe associata all'unità in cui viene mappato. Per assegnare una classe ad una unità, dopo la procedura di training della SOM, viene preso l'insieme composto dagli input del training set che hanno come vincente tale unità, la classe che gli viene assegnata è quella più frequente tra questo insieme.

Il metodo più semplice per il training supervisionato è di aggiungere una ulteriore fase fine di training, dopo il normale training non supervisionato, utilizzando l'algoritmo di Learning Vector Quantization (LVQ), come proposto in [23]. LVQ, nella sua forma base detta LVQ1, si basa sull'idea, di allontanare la codebook dell'unità vincente dal vettore di input se la classe del vettore è diversa da quella associata all'unità. Supponendo quindi di avere già allenato la SOM in modo non supervisionato, si può assegnare una classe ad ogni unità come detto in precedenza. Siccome è possibile che ci siano unità in cui nessun vettore di input viene mappato, a queste unità viene assegnata una classe nulla che, in questa fase, viene considerata come uguale a quella di qualunque vettore di input. Procedendo allo stesso modo del training non supervisionato, al passo  $t$ , dato l'input di training  $\mathbf{x}(t)$  e l'unità vincente  $c$ , l'aggiornamento delle codebooks  $\mathbf{m}_i(t)$  viene fatto in accordo alla seguente regola:

$$\begin{aligned} \mathbf{m}_c(t+1) &= \mathbf{m}_c(t) + \alpha(t)[\mathbf{x}(t) - \mathbf{m}_c(t)] \\ &\text{se } \mathbf{x}(t) \text{ è classificato correttamente} \\ \mathbf{m}_c(t+1) &= \mathbf{m}_c(t) - \alpha(t)[\mathbf{x}(t) - \mathbf{m}_c(t)] \quad (2.39) \\ &\text{se la classificazione di } \mathbf{x}(t) \text{ non è corretta} \\ \mathbf{m}_i(t+1) &= \mathbf{m}_i(t) \text{ per } i \neq c \end{aligned}$$

dove  $\alpha(t)$  è il learning rate per questa fase supervisionata, che decresce monotonicamente con il numero di iterazioni  $t$ . Si deve notare, però, che con questo algoritmo vengono modificate le singole unità senza aggiornare quelle vicine, e questo potrebbe rovinare l'ordinamento delle unità stesse sulla mappa, rovinando la visualizzazione della SOM che è una delle principali caratteristiche per cui viene presa in considerazione. Per evitare questo, nella sezione 3.1 viene proposta una modifica a tale algoritmo in cui si estende l'aggiornamento anche alle unità vicine, sia quando l'aggiornamento è di segno positivo sia quando è di segno negativo.

Un altro algoritmo di training supervisionato [76] propone di concatenare la classe al vettore di input e allenare la SOM con l'algoritmo classico non

supervisionato. In fase di classificazione gli elementi relativi alla classe vengono semplicemente eliminati dalle codebooks  $\mathbf{m}_i$  delle unità. Infine, un algoritmo più sofisticato è presentato in [77], il quale utilizza un termine di repulsione come in LVQ e, al contrario del metodo precedente (in [76]), aggiorna le codebooks solo localmente all'unità vincente nel caso questa porti ad una classificazione errata.

## 2.2 Tossicologia Computazionale

Conoscere l'impatto che un composto chimico può avere sulla salute degli esseri umani è una questione di indiscutibile importanza. Si stima che più di 100 000 composti chimici sono ampiamente utilizzati ogni giorno e tra 500 e 1000 nuovi ne vengono introdotti ogni anno, ma solo parte di questi sono testati riguardo potenziali effetti tossici, come mutagenicità o cancerogenicità [78]. Eseguire test empirici su tutti i composti che potrebbero avere effetti negativi sull'uomo è però un processo lento e costoso. Tutto questo ha portato al bisogno di strumenti computazionali in grado di prevedere gli effetti di un composto chimico basandosi solamente sulla sua struttura e su proprietà già note. La tossicologia computazionale [79] è l'area della chemioinformatica che si occupa dello studio di tali strumenti computazionali.

Gli approcci utilizzati per inferire proprietà di una molecola, come una determinata attività biologica, a partire dalla sua struttura e da un insieme di proprietà fisico/chimiche sono chiamati QSAR (Quantitative Structure Activity Relationship). I QSAR hanno subito un crescente interesse negli ultimi anni dovuto anche all'introduzione di leggi e regolamentazioni che ne incentivano l'utilizzo. In ambito europeo è entrata in vigore nel 2007 una legge che riguarda la registrazione, la valutazione, l'autorizzazione e la restrizione delle sostanze chimiche, comunemente indicata con l'acronimo REACH (Registration, Evaluation, Authorisation and Restriction of Chemical substances)<sup>4</sup>. Tale legge promuove una nuova visione per la gestione di sostanze chimiche in cui produttori ed importatori devono poter fornire informazioni dettagliate sulle sostanze trattate. L'obiettivo principale è quello di migliorare la protezione della salute umana e dell'ambiente da fattori di rischio che possono derivare da prodotti chimici. Viene richiesto che produttori e importatori di prodotti chimici valutino, identifichino e gestiscano i rischi legati alle sostanze che producono o commercializzano. In linea di principio, tutto questo si deve applicare ad ogni composto chimico, sia nuovo che esistente, industriale o domestico. Per fare questo REACH promuove l'innovazione nella valutazione di sostanze chimiche ed incoraggia l'utilizzo di metodi alternativi alle sperimentazioni su animali. L'utilizzo di metodi computazionali è specificatamente previsto e sono definiti criteri per

---

<sup>4</sup> [http://ec.europa.eu/environment/chemicals/reach/reach\\_intro.htm](http://ec.europa.eu/environment/chemicals/reach/reach_intro.htm)

l'utilizzo di modelli QSAR. In particolare, un QSAR è considerato valido per la valutazione di una sostanza se:

- Il modello è riconosciuto come scientificamente valido (in accordo ai principi OECD).
- La sostanza valutata è inclusa nel dominio applicativo del modello.
- I risultati sono appropriati per la classificazione e per la valutazione del rischio.
- È fornita un'adeguata documentazione per il metodo utilizzato.

In altre parole, l'uso di QSAR dovrebbe essere valutato in termini (1) della qualità scientifica del modello e dell'affidabilità dei dati sulla quale è basato, (2) se l'uso del modello è appropriato per il target per cui si deve valutare il composto chimico, (3) se il modello fornisce ogni informazione richiesta da REACH, e (4) se la documentazione del modello e del suo utilizzo è sufficiente per garantire una valutazione rigorosa ed indipendente da parte del regolatore.

I principi OECD ai quali REACH si riferisce sono 5 punti adottati dall'Organizzazione per la Cooperazione e lo Sviluppo Economico (OECD, appunto) per stabilire la validità dell'uso di modelli QSAR, nelle regolamentazioni che riguardano la sicurezza di prodotti chimici in ambito internazionale. Tali principi prevedono che un QSAR debba poter fornire le seguenti informazioni [22]:

1. Un *endpoint* (proprietà o attività target) ben definito: dev'essere chiara l'attività o la proprietà che il modello predice. Infatti, un certo endpoint può essere determinato da differenti metodologie sperimentali e sotto differenti condizioni. È importante quindi identificare il sistema modellato dal QSAR.
2. Un algoritmo non ambiguo: si deve garantire la trasparenza dell'algoritmo (del modello) con cui viene ottenuta la predizione, a partire dalle informazioni strutturali e dalle proprietà fisico/chimiche del composto.
3. Un dominio di applicabilità definito: si devono fornire informazioni che riguardano il tipo di strutture chimiche, di proprietà fisico/chimiche e di meccanismi di azione per il quale il modello è in grado di generare una predizione attendibile.
4. Una appropriata valutazione del modello tramite misure di *fitting* del training set, robustezza e capacità di generalizzazione.
5. Se possibile, un'interpretazione meccanicistica: si deve fornire qualche considerazione riguardo il meccanismo che lega le proprietà del

composto chimico utilizzate dal QSAR e l'endpoint che viene predetto. È comunque riconosciuto che non sempre è possibile, da un punto di vista scientifico, fornire un'interpretazione meccanicistica di un QSAR, o che può anche esserci più di una interpretazione meccanicistica di un dato modello.

### 2.2.1 Quantitative Structure Activity Relationship

I QSAR (Quantitative Structure Activity Relationship) [80, 81, 27] sono modelli matematici utilizzati per inferire una proprietà (detta *endpoint*) di una molecola, ad esempio una determinata attività biologica, a partire dalla sua struttura e da un insieme di proprietà fisico/chimiche. In altri termini lo scopo di un QSAR è di rappresentare una funzione  $\mathcal{T}$  che, data la struttura della molecola, predice una sua attività biologica:

$$\text{Attività} = \mathcal{T}(\text{Struttura}) \quad (2.40)$$

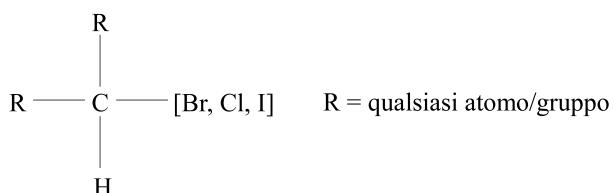
Esistono diversi approcci alla modellazione di QSAR e i più comuni sono basati sull'utilizzo di insiemi di *descrittori molecolari*. Alcuni esempi di descrittori sono proprietà fisico/chimiche della molecola, oppure indici topologici che rappresentano la geometria e la forma della molecola e le connessioni tra atomi. La scelta di un insieme di descrittori completo e rilevante è però un problema per questo tipo di approccio. Infatti può essere necessario il supporto di un esperto per una scelta di descrittori appropriata per l'endpoint da trattare e, inoltre, anche le scelte effettuate da un esperto possono risultare errate o incomplete, portando alla perdita di importanti informazioni legate all'attività biologica da predire. In aggiunta, un insieme di descrittori è generalmente adatto per uno specifico endpoint, portando alla necessità di dover scegliere un differente insieme di descrittori per ogni endpoint di cui si vuole creare un modello QSAR. Un approccio completamente differente, che evita di dover scegliere a priori un insieme di descrittori, consiste nell'utilizzare metodi di apprendimento automatico in grado di trattare direttamente dati strutturati. Molti di questi metodi sono descritti nella sezione precedente 2.1, con riferimenti anche ad applicazioni di questo tipo.

### 2.2.2 Structural Alerts

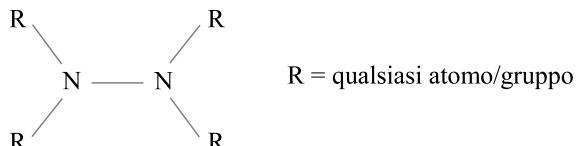
Le Structural Alerts (SAs) [82, 83] sono gruppi funzionali o sottostrutture molecolari connesse all'attività cancerogena e mutagenica di composti chimici. Rappresentano una sorta di codifica di una lunga serie di studi e sperimentazioni effettuate con lo scopo di evidenziare i meccanismi di azione di composti chimici mutagenici e cancerogeni. Una SA indica quindi che è stata riscontrata una correlazione tra un'attività mutagenica o cancerogena del composto chimico e la presenza della sottostruttura descritta dalla SA

stessa. Uno dei primi lavori orientato alla compilazione di un elenco di SAs risale agli anni 80, ad opera del chimico Ashby [84]. A partire dal lavoro di Ashby sono stati costruiti diversi insiemi di SAs, basati su studi sperimentali, su conoscenze di meccanismi chimici o su studi statistici. Nel seguito di questa tesi verrà fatto riferimento all'elenco di 30 SAs per mutagenicità compilato da Benigni e Bossa [85], unendo collezioni di SAs derivanti da altri diversi studi in un modo meno ridondante e più esaustivo possibile. Un esempio di alcune SAs, prese dall'elenco di Benigni/Bossa, è riportato in figura 2.17.

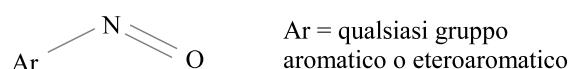
#### **SA\_8: Aliphatic alogens**



#### **SA\_13: Hydrazine**



#### **SA\_25: Aromatic nitroso group**



**Figura 2.17.** Esempio di 3 Structural Alerts prese dall'elenco in [86].

In questa tesi le SAs sono utilizzate per trovare riscontri nelle informazioni prodotte dal sistema realizzato con conoscenze derivanti da studi in ambito tossicologico. In altri lavori le SAs sono utilizzate direttamente per scopi predittivi, in sistemi QSAR. In [86] vengono impiegate per discriminare composti potenzialmente tossici, senza poter dare però informazioni su composti non tossici (se non per esclusione). In [87] vengono invece utilizzate per filtrare i risultati restituiti da un modello QSAR basato su Support Vector Machine. In questo caso lo scopo è di ridurre la percentuale di falsi negativi, quindi prevenire che molecole pericolose possano essere classificate come sicure.

## Capitolo 3

# Descrizione del Sistema Proposto

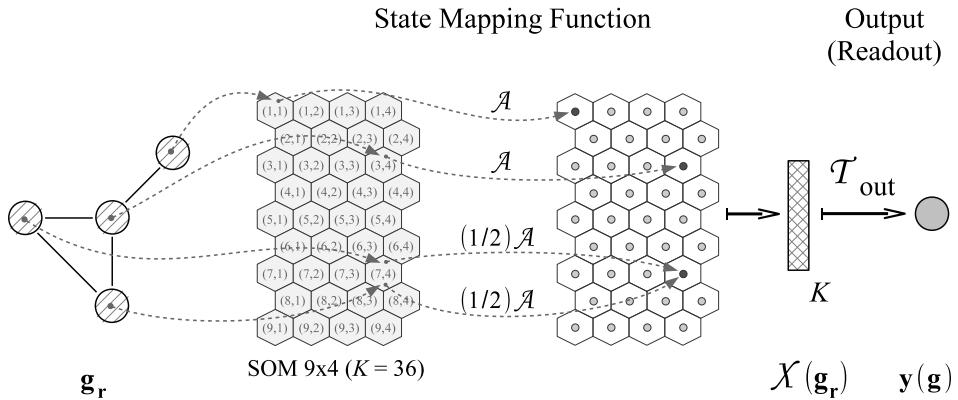
In questo capitolo viene descritto il sistema proposto in questa tesi, prima attraverso un quadro generale del funzionamento, poi descrivendone gli aspetti dettagliati nelle sezioni successive 3.1 e 3.2.

L’obiettivo principale è di creare un modello di Reservoir Computing (RC) per grafi in grado di fornire informazioni ed elementi utili per un’analisi qualitativa della predizione. Per fare questo si è scelto di estendere il modello GraphESN, implementando un’apposita State Mapping Function (SMF) e sfruttando tecniche di pruning per il training del readout. La SMF prende spunto da quella implementata in GraphESN-NG, basandosi sulla costruzione di un numero  $K$  di clusters per la rappresentazione a dimensione fissata del grafo restituito dal processo di encoding. Una SOM allenata in modo supervisionato viene utilizzata per la ripartizione dello spazio degli stati del reservoir. La SOM consente di visualizzare, su una mappa bi-dimensionale, un insieme di elementi utili all’analisi della predizione, legati alla ripartizione dello spazio degli stati e al modo in cui vengono creati i clusters. La SMF realizzata in questo sistema è innovativa sia nel contesto delle SMFs e sia per i suoi aspetti supervisionati. Grazie al training supervisionato della SOM, infatti, si crea una ripartizione dello spazio degli stati (ed una disposizione delle unità della SOM) che cattura aspetti significativi rispetto al problema in esame. Questo permette di differenziarsi efficacemente dai metodi a descrittori nei quali invece dev’essere scelto a priori, a seconda del problema, un insieme di caratteristiche con cui rappresentare un grafo.

Nella creazione di questo modello ci restringiamo al caso di problemi classificazione binaria ma, come viene discussso in seguito, l’estensione a problemi di regressione o classificazione con più di due classi è molto semplice da ottenere. Dalla procedura di training viene determinato un peso associato ad ogni unità della SOM, tale peso può essere rappresentato graficamente sulla mappa. Un grafo di input, che dev’essere classificato, attiva determinate

unità della SOM a seconda della sua topologia, e la predizione del modello è guidata dai pesi associati a tali unità. La visualizzazione delle unità attivate da un grafo, assieme ai pesi associati ad esse, può dare indicazioni importanti sul modo in cui il modello ha ottenuto la predizione.

Chiameremo GraphESN-SOM il sistema proposto in questa tesi. Il funzionamento della SMF del sistema GraphESN-SOM è rappresentato graficamente in Figura 3.1. Sia  $\mathbf{g}$  un grafo di input e sia  $\mathbf{g}_r$  il grafo mappato, dal processo di encoding, nello spazio degli stati del reservoir (i.e.  $\mathbf{g}_r = \mathcal{T}_{\text{enc}}(\mathbf{g})$ ). Ogni stato  $\mathbf{x}(v) \in \mathbb{R}^{N_R}$ , associato ad un vertice del grafo  $\mathbf{g}_r$  (i.e.  $v \in V(\mathbf{g}_r)$ ), viene quindi mappato in una unità della SOM. Per ogni unità della SOM  $i$  si crea un cluster, eventualmente vuoto, composto dagli stati mappati su tale unità. Ad ogni stato contenuto in ogni cluster viene applicata una *funzione di aggregazione*  $\mathcal{A}: \mathbb{R}^{N_R} \rightarrow \mathbb{R}$  che restituisce un valore scalare per ogni stato. Calcolando la media, localmente ad ogni cluster, dei valori restituiti dalla funzione di aggregazione, oppure prendendo un valore nullo nel caso il cluster sia vuoto, si ottiene un singolo valore in corrispondenza di ogni unità della SOM. La concatenazione di tali valori crea il vettore finale  $\mathcal{X}(\mathbf{g}_r) \in \mathbb{R}^K$  restituito dalla SMF, che è una rappresentazione a dimensione fissata del grafo di input  $\mathbf{g}$ .



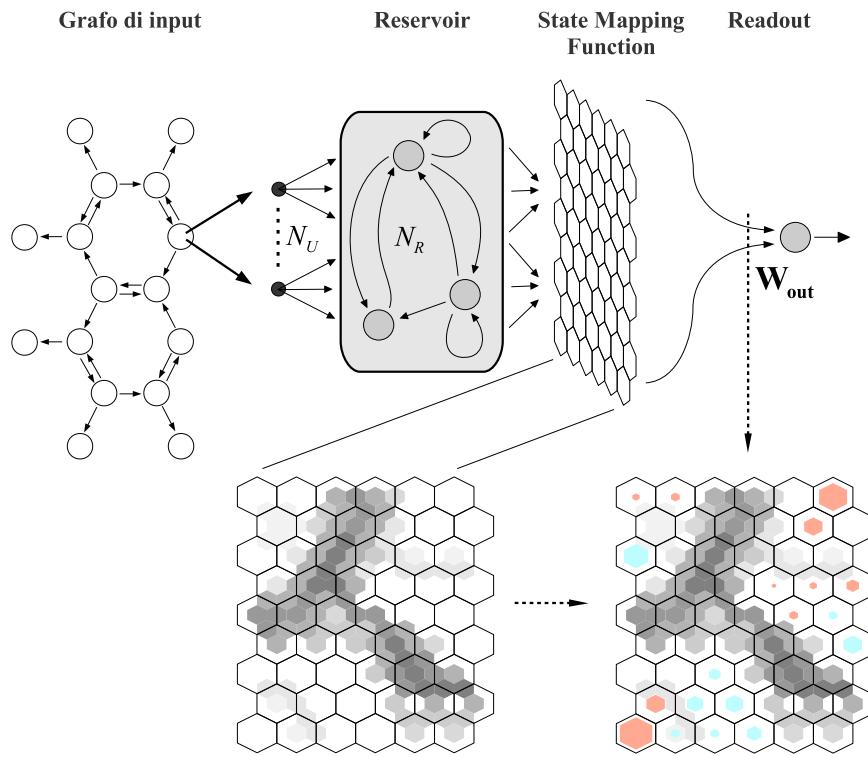
**Figura 3.1.** Funzionamento della State Mapping Function di GraphESN-SOM per un grafo di input  $\mathbf{g}$ .

La riduzione ad un solo elemento ottenuta dalla funzione  $\mathcal{A}$  porta con se alcuni vantaggi che riguardano l'interpretazione della risposta, mentre può non influire particolarmente sulle performance, come discusso nella sezione successiva 3.1 e verificato sperimentalmente nella sezione 4.6. I vantaggi riguardano il fatto che, con un singolo valore ottenuto in corrispondenza di ogni cluster, nel readout viene assegnato un singolo peso ad ogni unità della SOM. Questo permette di (1) rappresentare graficamente il peso sulla mappa in corrispondenza di ogni cella, (2) esporre un'equazione di funzionamento del modello in termini di attivazione delle unità della SOM, (3) visualizzare un

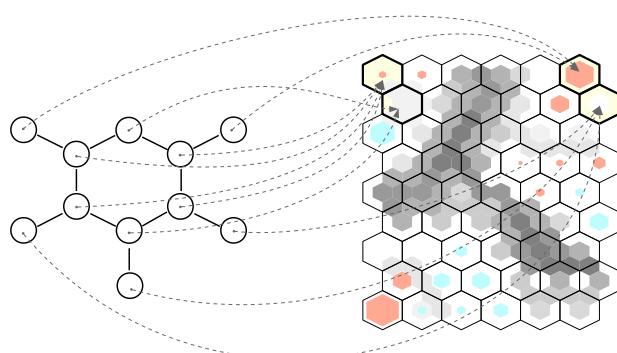
peso in corrispondenza di ogni vertice di un grafo di input che può mettere in luce quali sono le componenti strutturali che più hanno influito nella predizione. Tutto questo va nella direzione di fornire un maggior numero di elementi per un’analisi qualitativa della risposta. Inoltre, introducendo una tecnica di pruning per il training del readout, questa può eliminare le unità della SOM che raccolgono le features (gli stati del reservoir) meno influenti per il problema (e lasciare solo le unità che raccolgono le features più influenti) semplicemente eliminando il peso associato ad esse. Un altro vantaggio lo si ha nell’essere indipendenti dalla dimensione del reservoir, che può avere anche centinaia di unità. Mantenere l’intero stato del reservoir per creare lo stato finale, invece che ridurlo ad un solo valore, porterebbe ad avere un vettore  $\mathcal{X}(\mathbf{g}_r)$  di dimensioni troppo elevate e il training del readout sarebbe impraticabile computazionalmente (a meno di non ridurre a poche unità la SOM, il che però andrebbe contro i motivi per cui viene utilizzata, rendendone, ad esempio, poco interessante la visualizzazione). L’output finale viene, infine, calcolato in accordo all’equazione 2.23 delle GraphESNs, utilizzando il vettore di  $K$  elementi (uno per ogni unità della SOM) restituito dalla SMF, come illustrato nella figura 3.1. Il training dei pesi del readout viene fatto utilizzando Elastic Net che permette di fare regolarizzazione e pruning contemporaneamente. Il pruning serve a semplificare il modello e concentrare su un numero limitato di features la predizione, in modo da facilitarne l’interpretazione. Un altro vantaggio del pruning si ha in termini di efficienza (in fase di test) siccome, annullando molti pesi, il calcolo della funzione di output è notevolmente semplificato.

In Figura 3.2 è riportata una rappresentazione grafica dell’architettura finale di una GraphESN-SOM. La SOM, costruita in fase di training, può essere visualizzata utilizzando, per esempio, una U-Matrix (in basso al centro nella figura) e i pesi del readout si possono rappresentare in corrispondenza di ogni unità (in basso a destra) con dimensioni grafiche che rispecchiano le dimensioni del peso stesso. Nel caso di problemi di classificazione, si può assegnare un colore differente ad ogni peso a seconda della classe verso cui dirigono la classificazione. In Figura 3.3 viene mostrato come un grafo di input viene mappato sulla SOM. Ogni vertice è mappato in una unità della SOM in base al valore dello stato del reservoir associato ad esso (che dipende dall’intorno del vertice nel grafo stesso). Il peso associato a tali unità guida la classificazione del grafo. Nell’esempio in figura, se in rosso sono rappresentati pesi di valore positivo, la classificazione del grafo sarà positiva.

Nella sezione 3.1 vengono descritti i dettagli della State Mapping Function, viene illustrato l’algoritmo di training supervisionato della SOM utilizzato nelle prove sperimentali e vengono introdotte alcune possibili funzioni di aggregazione che si possono utilizzare nella SMF. Nella sezione 3.2 è illustrata la procedura di training del readout e si discutono i vantaggi portati dal pruning. Nella sezione 3.4 vengono discussi i costi computazionali della GraphESN-SOM. Infine, nella sezione 3.5 viene descritto un sistema simile,



**Figura 3.2.** Rappresentazione grafica di una GraphESN-SOM. La SOM può essere visualizzata utilizzando una U-Matrix (in basso al centro) e i pesi del readout si possono rappresentare in corrispondenza di ogni unità (in basso a destra) con dimensioni grafiche che rispecchiano le dimensioni del peso stesso.



**Figura 3.3.** Un grafo di input viene mappato sulla SOM. Se in rosso sono rappresentati pesi di valore positivo, la classificazione del grafo sarà positiva.

basato sulla scomposizione del grafo in frammenti, che viene utilizzato per un confronto con GraphESN-SOM nella sezione 4.5. Nel capitolo 4 vengono giustificate le varie scelte effettuate attraverso una serie di risultati sperimentali.

### 3.1 State Mapping Function

Definiamo meglio il funzionamento della SMF. Dato un insieme di grafi di input  $\mathcal{G} \subseteq (\mathbb{R}^{N_U})^\#$  e un training set  $\mathcal{T}$ :

$$\mathcal{T} = \{\langle \mathbf{g}, \mathbf{y}_{\text{target}}(\mathbf{g}) \rangle \mid \mathbf{g} \in \mathcal{G}, \mathbf{y}_{\text{target}}(\mathbf{g}) \in \mathbb{R}^{N_Y}\}$$

composto da un insieme di coppie  $\langle \mathbf{g}, \mathbf{y}_{\text{target}}(\mathbf{g}) \rangle$  dove  $\mathbf{g}$  è un grafo di input e  $\mathbf{y}_{\text{target}}(\mathbf{g})$  è il valore target associato a tale grafo, sia

$$\mathcal{R} = \{\mathbf{x}(v) \in \mathbb{R}^{N_R} \mid \mathbf{g} \in \mathcal{G}, v \in V(\mathbf{g})\}$$

l'insieme degli stati del reservoir calcolati per ogni vertice di ogni grafo di input del training set. Una SOM con  $K$  unità, allenata in modo supervisionato come descritto nella sezione successiva 3.1.1, ripartisce l'insieme  $\mathcal{R}$  ottenendo le codebooks  $m_1, \dots, m_K \in \mathbb{R}^{N_R}$ . Lo spazio degli stati del reservoir  $\mathbb{R}^{N_R}$  è così suddiviso in  $K$  celle di Voronoi  $\mathcal{R}^{(i)}$  per ogni  $i = 1, \dots, K$ :

$$\mathcal{R}^{(i)} = \{\mathbf{x} \in \mathbb{R}^{N_R} \mid \|\mathbf{x} - \mathbf{c}_i\|_2 \leq \|\mathbf{x} - \mathbf{c}_j\|_2 \quad \forall j = 1, \dots, K\}$$

Sia  $\mathbf{g}_r = \mathcal{T}_{\text{enc}}(\mathbf{g})$  il grafo risultante dal processo di encoding per un grafo di input  $\mathbf{g}$ . Denotiamo con  $V^{(i)}(\mathbf{g}_r)$  l'insieme dei vertici di  $\mathbf{g}_r$  che ricadono nell' $i$ -esima cella di Voronoi (formando l' $i$ -esimo cluster):

$$V^{(i)}(\mathbf{g}_r) = \{v \in V(\mathbf{g}_r) \mid \mathbf{x}(v) \in \mathcal{R}^{(i)}\}$$

L'applicazione della funzione di aggregazione  $\mathcal{A}: \mathbb{R}^{N_R} \rightarrow \mathbb{R}$  ad ogni stato  $\mathbf{x}(v)$ , relativo ad un vertice di un insieme  $V^{(i)}(\mathbf{g}_r)$ , crea un insieme di valori locali all' $i$ -esimo cluster la cui media costituisce il valore  $\mathcal{A}^{(i)}(\mathbf{g}_r) \in \mathbb{R}$ , per ogni  $i = 1, \dots, K$ :

$$\mathcal{A}^{(i)}(\mathbf{g}_r) = \begin{cases} \frac{1}{|V^{(i)}(\mathbf{g}_r)|} \sum_{v \in V^{(i)}(\mathbf{g}_r)} \mathcal{A}(\mathbf{x}(v)) & \text{se } |V^{(i)}(\mathbf{g}_r)| > 0 \\ 0 \in \mathbb{R} & \text{altrimenti} \end{cases} \quad (3.1)$$

La concatenazione dei valori  $\mathcal{A}^{(i)}(\mathbf{g}_r)$  crea il vettore  $\mathcal{X}(\mathbf{g}_r) \in \mathbb{R}^K$  restituito dalla SMF per il grafo di input  $\mathbf{g}$ .

### 3.1.1 Training Supervisionato della SOM

Come detto la SOM viene allenata in modo supervisionato, utilizzando l'algoritmo proposto in questa sezione. Il training è fatto in tre diverse fasi (come descritto anche nella sezione 2.1.10), le prime due non supervisionate, dette rispettivamente fase grezza e fase fine, mentre l'ultima è la fase di training supervisionato. Dato un training set  $\mathcal{T}$  per una GraphESN-SOM, composto da  $T$  elementi ( $|\mathcal{T}| = T$ ), definiamo l'insieme  $\mathcal{T}_V$ :

$$\mathcal{T}_V = \{\langle \mathbf{x}(v), \mathbf{y}_{\text{target}}(\mathbf{g}) \rangle \mid \mathbf{x}(v) \in V(\mathcal{T}_{\text{enc}}(\mathbf{g})) \quad \forall \langle \mathbf{g}, \mathbf{y}_{\text{target}}(\mathbf{g}) \rangle \in \mathcal{T}\} \quad (3.2)$$

composto dagli stati  $\mathbf{x}(v)$  restituiti dal processo di encoding per ogni vertice di ogni grafo  $\mathbf{g}$  del training set  $\mathcal{T}$ , associati al valore target  $\mathbf{y}_{\text{target}}(\mathbf{g})$  del grafo stesso. Tale insieme costituisce il training set della SOM in una GraphESN-SOM. Nel seguito parlando di *stati* ci riferiamo quindi agli elementi di input della SOM nella SMF della GraphESN-SOM.

Nell'Algoritmo 3.1 sono riportati i passi di una fase di training non supervisionato, per un numero  $T_E$  di epoche. Le codebooks  $\mathbf{m}_i(0)$ , di ogni unità della SOM  $i = 1, \dots, K$ , sono inizializzate in modo casuale nel range dei valori degli stati  $\mathbf{x}(v)$  del training set  $\mathcal{T}_V$ , dopodiché questa procedura viene ripetuta prima per la fase grezza, poi per la fase fine, con parametri  $T_E$ ,  $\alpha_{\text{ini}}$ ,  $\rho_{\text{ini}}$  e  $\rho_{\text{fin}}$  differenti. La funzione di vicinato  $h_{ci}(\alpha(s), \rho(s))$  (si veda

---

**Algoritmo 3.1** Fase di training non supervisionato della SOM

---

1.  $s \leftarrow 0$
  2.  $T_S \leftarrow T_E \cdot |\mathcal{T}_V|$
  3.  $\alpha(s) \leftarrow \alpha_{\text{ini}}$
  4.  $\rho(s) \leftarrow \rho_{\text{ini}}$
  5. **for**  $e \leftarrow 1$  **to**  $T_E$  **do**
  6.   **for all**  $\langle \mathbf{x}(v), \mathbf{y}_{\text{target}}(\mathbf{g}) \rangle \in \mathcal{T}_V$  **do** {scelti in modo casuale (senza ripetizioni)}
  7.     Trova l'unità vincente  $c$ , tra ogni unità  $i$ , per il vettore  $\mathbf{x}(v)$ , i.e  $c = \operatorname{argmin}_i \|\mathbf{x}(v) - \mathbf{m}_i(s)\|_2$
  8.     **for**  $i \leftarrow 1$  **to**  $K$  **do** {i.e. per ogni unità della SOM}
  9.        $\mathbf{m}_i(s+1) \leftarrow \mathbf{m}_i(s) + h_{ci}(\alpha(s), \rho(s))[\mathbf{x}(v) - \mathbf{m}_i(s)]$
  10.      **end for**
  11.      $\alpha(s+1) \leftarrow \alpha_{\text{ini}}(1 - (s/T_S))$
  12.      $\rho(s+1) \leftarrow \rho_{\text{ini}} + s((\rho_{\text{fin}} - \rho_{\text{ini}})/(T_E - 1))$
  13.      $s \leftarrow s + 1$
  14.   **end for**
  15. **end for**
-

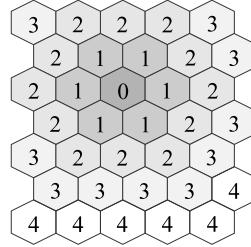
la sezione 2.1.10), nella riga 9, è definita come segue:

$$h_{ci}(\alpha(s), \rho(s)) = \begin{cases} \alpha(s)G_{\rho(s)}(c, i) & \text{se } G_{\rho(s)}(c, i) \geq 1 \times 10^{-6} \\ 0 & \text{altrimenti} \end{cases} \quad (3.3)$$

dove  $G_{\rho(s)}(c, i)$  è la funzione gaussiana:

$$G_{\rho(s)}(c, i) = \exp \left( -0.5 \left( \frac{\text{hex}(c, i)}{\rho(s)} \right)^2 \right) \quad (3.4)$$

in cui  $\text{hex}(c, i)$  è la funzione che restituisce la distanza tra l'unità  $c$  e l'unità  $i$  sulla griglia esagonale della SOM, come rappresentato in Figura 3.4. Completate le due fasi di training non supervisionato viene eseguita la fase di



**Figura 3.4.** Su ogni cella è indicata la distanza rispetto alla cella in grigio più scuro (con distanza 0 da se stessa) su una griglia esagonale. Se la cella più scura è l'unità  $c$  della SOM, allora il numero su ogni cella, di un'unità  $i$ , è il valore restituito dalla funzione  $\text{hex}(c, i)$ .

training supervisionato seguendo la procedura riportata nell'Algoritmo 3.2, del tutto simile alla procedura non supervisionata, a parte l'aggiornamento delle codebooks (nelle righe 11 e 13) che viene fatto come in LVQ1 (nell'equazione 2.39) con un termine di repulsione in caso di classificazione errata, aggiornando però anche le unità vicine nella griglia tramite la funzione di vicinato  $h_{ci}$ .

Per ognuna delle tre fasi devono essere scelti, in modo appropriato, i parametri  $T_E$ ,  $\alpha_{\text{ini}}$ ,  $\rho_{\text{ini}}$  e  $\rho_{\text{fin}}$ . Dei buoni valori consentono alle codebooks di disporsi in modo da rappresentare i dati di training della SOM (come descritto meglio nel paragrafo successivo 3.1.2) dopo le due fasi non supervisionate e di disporsi in modo da separare (dove possibile) gruppi di dati con target differente durante la fase supervisionata (come si vede nelle prove sperimentali nella sezione 4.2). Buoni valori possono essere presi in funzione della dimensione della SOM e del numero di elementi nel training set  $\mathcal{T}_V$ , come ad esempio viene fatto in [88] per il training classico (non supervisionato). Nella sezione 4.2 sono riportati i parametri scelti in tale modo utilizzati nelle prove sperimentali di questa tesi. In questo modo si evita di selezionarli tramite

---

**Algoritmo 3.2** Fase di training supervisionato della SOM

---

1.  $s \leftarrow 0$
2.  $T_S \leftarrow T_E \cdot |\mathcal{T}_V|$
3.  $\alpha(s) \leftarrow \alpha_{\text{ini}}$
4.  $\rho(s) \leftarrow \rho_{\text{ini}}$
5. Assegna ad ogni unità  $i$  la classe  $\mathbf{y}_i$  uguale al valore target  $\mathbf{y}_{\text{target}}(\mathbf{g})$  più frequente tra quelli associati ai vettori  $\mathbf{x}(v)$  mappati su tale unità; se nessun vettore viene mappato in  $i$  allora  $\mathbf{y}_i = \text{NULL}$ .
6. **for**  $e \leftarrow 1$  **to**  $T_E$  **do**
7.   **for all**  $\langle \mathbf{x}(v), \mathbf{y}_{\text{target}}(\mathbf{g}) \rangle \in \mathcal{T}_V$  **do** {scelti in modo casuale (senza ripetizioni)}
8.     Trova l'unità vincente  $c$ , tra ogni unità  $i$ , per il vettore  $\mathbf{x}(v)$ , i.e  $c = \operatorname{argmin}_i \|\mathbf{x}(v) - \mathbf{m}_i(s)\|_2$
9.     **for**  $i \leftarrow 1$  **to**  $K$  **do** {i.e. per ogni unità della SOM}
10.       **if**  $\mathbf{y}_{\text{target}}(\mathbf{g}) = \mathbf{y}_c$  **or**  $\mathbf{y}_c = \text{NULL}$  **then**
11.          $\mathbf{m}_i(s+1) \leftarrow \mathbf{m}_i(s) + h_{ci}(\alpha(s), \rho(s))[\mathbf{x}(v) - \mathbf{m}_i(s)]$
12.        **else**
13.          $\mathbf{m}_i(s+1) \leftarrow \mathbf{m}_i(s) - h_{ci}(\alpha(s), \rho(s))[\mathbf{x}(v) - \mathbf{m}_i(s)]$
14.        **end if**
15.     **end for**
16.     Se il vettore  $\mathbf{x}(v)$ , nell'epoca precedente  $e-1$ , era mappato in una unità diversa da  $c$ , allora aggiorna la classe dell'unità  $c$  e la classe dell'unità in cui  $\mathbf{x}(v)$  era mappato prima.
17.      $\alpha(s+1) \leftarrow \alpha_{\text{ini}}(1 - (s/T_S))$
18.      $\rho(s+1) \leftarrow \rho_{\text{ini}} + s((\rho_{\text{fin}} - \rho_{\text{ini}})/(T_E - 1))$
19.      $s \leftarrow s + 1$
20.   **end for**
21. **end for**

---

una griglia di ricerca che comprenderebbe dodici iperparametri (quattro per ognuna delle tre fasi), quindi estremamente onerosa computazionalmente.

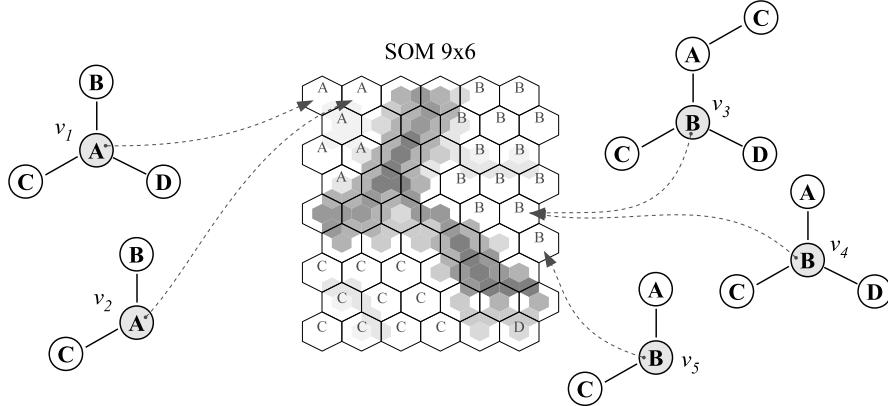
### 3.1.2 Disposizione delle Unità della SOM

Ogni unità della SOM, con la sua codebook, rappresenta un punto nello spazio di input (che nella SMF è lo spazio degli stati del reservoir  $\mathbb{R}^{N_R}$ ) che, attraverso la procedura di training, si dispone in modo da rappresentare un certo insieme di elementi. Questa disposizione è legata anche alla posizione delle unità stesse nella griglia della SOM, e può essere comodamente visualizzata, per esempio, con una U-Matrix. Il modo in cui si dispongono è caratterizzato da un insieme di fattori tra cui di particolare importanza è l'effetto markoviano sugli stati del reservoir descritto nella sezione 2.1.8. Capire i motivi per cui le unità si dispongono in un certo modo è fondamentale

per riuscire a leggere correttamente le mappe ottenute dalla SOM e quindi capire perché i vertici di un grafo attivano certe unità, che ne determinano la classificazione.

Come per ogni metodo di Vector Quantization, le  $K$  codebooks  $m_i$  della SOM tendono ad approssimare la distribuzione dei vettori di training, disponendosi in numero maggiore nelle regioni con una maggiore densità di esempi e in numero minore dove gli esempi sono pochi. Questo viene fatto cercando anche di mantenere le proprietà topologiche dei campioni di input, riportandole sulla griglia di unità, in modo tale che elementi simili portino all'attivazione di unità adiacenti. I campioni di input, in questo caso, sono gli stati del reservoir, mentre gli esempi di training sono gli stati prodotti per ogni vertice di un grafo del training set su cui la GraphESN-SOM è stata allenata. Gli effetti markoviani sugli stati del reservoir, dovuti alla contrattività della funzione di transizione di stato, fanno sì che due stati siano tanto più simili quanto lo sono gli intorni dei due relativi vertici, con un effetto che svanisce considerando intorni di raggio maggiore. Se, per esempio, consideriamo tre vertici  $v_1$ ,  $v_2$  e  $v_3$ , i primi due con la stessa etichetta, ma con intorno di raggio 1 differente, mentre il terzo con una etichetta differente rispetto ai primi due, allora negli stati  $\mathbf{x}(v_1), \mathbf{x}(v_2), \mathbf{x}(v_3) \in \mathbb{R}^{N_R}$ , prodotti dal reservoir in corrispondenza dei rispettivi vertici, si avrà che  $\mathbf{x}(v_1)$  ed  $\mathbf{x}(v_2)$  saranno più simili tra loro di quanto lo possa essere  $\mathbf{x}(v_3)$  rispetto ad ognuno degli altri due. Tutto questo porta la SOM a disporsi come raffigurato, con un esempio, in Figura 3.5. Visualizzando la SOM attraverso una U-Matrix si possono distinguere varie aree che rappresentano stati simili, separate tra di loro dalle zone scure della U-Matrix. Ad ogni cella può essere associata un'etichetta come quella più frequente tra quelle associate ai vertici di training mappati su tale cella. Tipicamente le aree si formano rappresentando insiemi di vertici con etichette uguali, tanto più ampie quanto era il numero di esempi con tale elemento nel training set. All'interno di ogni area le unità si dispongono in modo tale che vertici con intorno simile attivino unità adiacenti e due celle differenti rappresentano vertici con diversi intorni. Nell'esempio in Figura 3.5, i vertici  $v_3$  e  $v_4$  vengono mappati nella stessa unità perché il loro intorno è diverso solamente considerando un raggio maggiore di 1, e questo ha meno peso sul valore dello stato, mentre i vertici  $v_1$  e  $v_2$  vengono mappati in due unità diverse, ma comunque adiacenti, siccome il loro intorno di raggio 1 è differente, ma molto simile. Quindi, dato un grafo da classificare, ogni suo vertice attiva una unità della SOM in base alla sua etichetta e alla struttura (e le etichette dei vertici) del suo intorno. Per cui, nel complesso, un grafo attiva determinate unità della SOM a seconda della topologia locale ad ogni suo vertice.

Fissato ogni altro parametro del modello, l'aumentare del coefficiente di contrattività del reservoir  $\sigma$  (dell'equazione 2.26) fa sì che lo stato prodotto dal reservoir, per un certo vertice, sia condizionato dal valore di vertici più lontani (o meglio, da differenze in intorni di raggio maggiore) in modo più



**Figura 3.5.** Esempio di disposizione delle unità della SOM. I vertici  $v_3$  e  $v_4$  sono mappati nella stessa unità perché il loro intorno è diverso solamente considerando un raggio maggiore di 1, mentre i vertici  $v_4$  e  $v_5$  (e in modo analogo i vertici  $v_1$  e  $v_2$ ) sono mappati in due unità diverse, ma comunque adiacenti, siccome il loro intorno di raggio 1 è differente, ma molto simile.

consistente rispetto ad un  $\sigma$  minore. Questo può portare a SOM in cui le aree descritte sopra, distinte per l'elemento associato al vertice, siano meno evidenti, poiché ha molta influenza sui valori degli stati anche l'intorno del vertice.

### 3.1.3 Funzione di Aggregazione

La funzione di aggregazione  $\mathcal{A}$  determina come rappresentare (e distinguere) gli stati del reservoir  $\mathbf{x}(v)$  con un singolo valore. Possibili funzioni di aggregazione sono le seguenti:

- Funzione di *binarizzazione*: restituisce il valore 1 se il vettore  $\mathbf{x}(v)$  è diverso dal vettore nullo, 0 altrimenti:

$$\mathcal{A}(\mathbf{x}(v)) = \begin{cases} 1 & \text{se } \mathbf{x}(v) \neq \mathbf{0} \in \mathbb{R}^{N_R} \\ 0 & \text{altrimenti} \end{cases} \quad (3.5)$$

- *Media*: viene calcolata la media degli elementi nel vettore  $\mathbf{x}(v)$ :

$$\mathcal{A}(\mathbf{x}(v)) = \frac{1}{N_R} \sum_{j=1}^{N_R} [\mathbf{x}(v)]_j \quad (3.6)$$

- *Somma*: viene calcolata la somma degli elementi nel vettore  $\mathbf{x}(v)$ . Dal punto di vista dell'informazione codificata in un singolo valore questo

metodo è del tutto simile alla media dei valori:

$$\mathcal{A}(\mathbf{x}(v)) = \sum_{j=1}^{N_R} [\mathbf{x}(v)]_j \quad (3.7)$$

- *Distanza* con la codebook  $m_i$ : siccome il vettore  $\mathbf{x}(v)$  è un vettore che è mappato in una unità  $i$  della SOM, può essere interessante distinguere (con un singolo valore) vettori differenti in base alla distanza dalla codebook dell'unità in cui sono mappati. Per fare questo si può utilizzare la seguente funzione:

$$\mathcal{A}(\mathbf{x}(v)) = \frac{1}{1 + \left( \frac{\|\mathbf{x}(v) - m_i\|_2}{\gamma} \right)^2} \quad (3.8)$$

per una qualche costante  $\gamma$  che dipende dalla dimensione degli elementi dei vettori  $\mathbf{x}(v)$ . Tale funzione restituisce un valore tra 0 ed 1, dove valori vicini ad 1 indicano che  $\mathbf{x}(v)$  e la codebook  $m_i$  sono molto simili, mentre valori tendenti a 0 indicano una maggiore distanza tra il vettore e la codebook.

Siccome l'obiettivo è migliorare l'interpretazione della risposta data dal sistema, si hanno diversi vantaggi nell'utilizzare una funzione di aggregazione, riducendosi ad utilizzare un solo valore:

- La possibilità di assegnare un unico peso nel readout ad ogni unità della SOM permette di rappresentare graficamente il peso sulla mappa in corrispondenza di ogni cella, oppure in corrispondenza di ogni vertice di un grafo di input mettendo in luce le componenti strutturali che influiscono maggiormente nella predizione.
- È possibile esporre un'equazione lineare che descrive il funzionamento del modello in termini di attivazione delle unità della SOM.
- Il pruning del readout, che a sua volta porta vantaggi nel semplificare l'analisi della risposta, può essere fatto agevolmente con le tecniche di pruning più diffuse ed utilizzate. Mantenendo l'intero stato del reservoir si avrebbero gruppi di features da gestire in modo più articolato per poter fare altrettanto.
- Infine, siccome il reservoir viene tipicamente costruito con un numero elevato di unità (nelle prove sperimentali fatte in questa tesi si utilizzano reservoir di 30 unità, ma in genere sono costruiti anche con centinaia di unità per migliorare le prestazioni), se si mantiene l'intero stato nella risposta della SMF si hanno difficoltà a costruire SOM di grandi dimensioni per via dell'occupazione di memoria e delle risorse

computazionali necessarie al training del readout. A parità di prestazioni, per migliorare l'interpretazione, è preferibile avere una mappa più grande in cui si può cogliere visivamente il motivo della risposta del sistema.

Si può notare che utilizzando la funzione di binarizzazione i vettori restituiti dalla SMF sono vettori a valori binari e i pesi ottenuti dal training del readout diventano immediatamente interpretabili. Infatti, nella combinazione lineare con cui viene calcolato l'output finale, di fatto sono solamente i pesi ad essere sommati (poiché moltiplicati per 1 o 0), quindi a classificare un grafo di input. Guardando segno e dimensione del peso si può quindi capire direttamente che influenza ha sulla classificazione. Un esempio di questo lo si può vedere nella sezione 4.7. Nella sezione 4.3 viene fatto invece un confronto sperimentale dell'impatto sulle performance predittive della scelta della funzione di aggregazione. Lo scopo è di verificare se utilizzando valori binari, che portano a vantaggi in termini di analisi della risposta, si riescano comunque ad ottenere buone performance, al livello di funzioni di aggregazioni che restituiscono valori continui (come, ad esempio, la funzione che somma i valori delle componenti).

## 3.2 Training del Readout

Il training del readout segue la stessa procedura delle GraphESNs, utilizzando però la tecnica di regolarizzazione Elastic Net per determinare i pesi in  $\mathbf{W}_{\text{out}}$ . Questa tecnica ha la proprietà di annullare i pesi molto più facilmente rispetto ad altre tecniche di regolarizzazione come Ridge Regression, consentendo di effettuare pruning e restringimento dei pesi contemporaneamente aumentando i parametri  $\lambda_1$  e  $\lambda_2$  che controllano il grado di regolarizzazione. Dato un training set  $\mathcal{T}$  su un insieme di grafi di input  $\mathcal{G} \subseteq (\mathbb{R}^{N_U})^{\#}$ :

$$\mathcal{T} = \{\langle \mathbf{g}, \mathbf{y}_{\text{target}}(\mathbf{g}) \rangle \mid \mathbf{g} \in \mathcal{G}, \mathbf{y}_{\text{target}}(\mathbf{g}) \in \mathbb{R}^{N_Y}\}$$

composto da  $T$  elementi, cioè  $T = |\mathcal{T}|$ , si costruisce la matrice  $\mathbf{X} \in \mathbb{R}^{K \times T}$  concatenando i vettori restituiti dalla SMF, colonna per colonna, per ogni grafo di input  $\mathbf{g}_i$ . Sia  $\mathbf{Y}_{\text{target}} \in \mathbb{R}^{N_Y \times T}$  la matrice ottenuta concatenando, colonna per colonna, gli output del training set  $\mathbf{y}_{\text{target}}(\mathbf{g}_i)$ . I pesi della matrice  $\mathbf{W}_{\text{out}}$ , utilizzando Elastic Net, sono ottenuti risolvendo il seguente problema di minimizzazione:

$$\begin{aligned} \mathbf{w}_{\text{out}}^j &= \underset{\mathbf{w}}{\operatorname{argmin}} \|\mathbf{w}\mathbf{X} - \mathbf{y}_{\text{target}}^j\|_2^2 + 2\lambda_1\|\mathbf{w}\|_1 + \lambda_2\|\mathbf{w}\|_2^2 \\ \text{per } j &= 1, \dots, N_Y \end{aligned} \tag{3.9}$$

dove  $\mathbf{w}_{\text{out}}^j \in \mathbb{R}^{1 \times K}$  e  $\mathbf{y}_{\text{target}}^j \in \mathbb{R}^{1 \times T}$  sono le  $j$ -esime righe rispettivamente della matrice  $\mathbf{W}_{\text{out}}$  e della matrice  $\mathbf{Y}_{\text{target}}$ .

### 3.2.1 Vantaggi del Pruning

Come già descritto nella sezione 2.1.9, utilizzare tecniche di regolarizzazione, per il training dei pesi del readout, è indispensabile per evitare problemi di overfitting ed ottenere buone prestazioni. La regolarizzazione può essere fatta restringendo la dimensione dei pesi oppure attraverso il pruning delle connessioni del readout, eliminando quelle ritenute meno importanti (annullando i pesi associati ad esse). Con tecniche come Lasso ed Elastic Net si riesce ad effettuare il restringimento dei pesi e il pruning contemporaneamente. Nella GraphESN-SOM viene utilizzato Elastic Net per fare sia regolarizzazione, quindi per migliorare le performance ed evitare problemi di overfitting, sia per sfruttare alcuni aspetti del pruning.

Ad ogni unità della SOM è associato un peso del readout, diciamo allora che una unità della SOM è *annullata* dal pruning se il peso corrispondente è annullato dalla procedura di training. Ognuna di queste unità ha il compito di raccogliere un insieme di vertici del grafo da classificare, e la classificazione del grafo dipende dall'insieme di pesi associati alle unità che hanno raccolto almeno un vertice. In altri termini, ad ogni vertice di un grafo di input viene associato un peso e la predizione del modello dipende dall'insieme dei pesi (non nulli) associati ai vertici del grafo. In questo contesto il pruning può portare a diversi vantaggi:

- Semplifica l'analisi della predizione riducendo il numero di elementi che determinano la risposta del sistema: se ad ogni vertice di un grafo da classificare corrisponde un peso non nullo, che contribuisce alla risposta del modello, la predizione è molto più difficile da analizzare piuttosto che se, nello stesso grafo, i pesi non nulli sono presenti su pochi vertici. In quest'ultimo caso, infatti, si riesce a capire quali sono le parti del grafo che influiscono effettivamente sulla risposta.
- Permette di individuare facilmente i grafi che il modello non riesce a classificare: questo succede quando ogni vertice di un grafo viene mappato su una unità della SOM annullata dal pruning, perciò ogni peso associato ad un vertice è un peso nullo. Seppure potrebbe sembrare uno svantaggio, soprattutto in termini di performance, per la mancata capacità di generalizzare su alcuni grafi, in realtà consente di capire con chiarezza quando il modello non è sicuro della predizione e non ha abbastanza elementi per classificare il grafo. Quindi può dare una chiara informazione sull'attendibilità della risposta, che verrebbe comunque data come uguale al bias del readout.
- Migliora l'efficienza del readout in fase di test eliminando molte connessioni dal calcolo della risposta finale.

Di contro, un pruning eccessivo può avere effetti molto negativi sulle prestazioni, vanificandone tutti i possibili vantaggi. Per questo motivo si deve

trovare un buon trade-off tra quantità di pruning e prestazioni del modello. Negli esperimenti effettuati, riportati nel capitolo successivo, si sono ottenuti alti livelli di pruning (in genere oltre il 70% delle unità della SOM sono state annullate) con prestazioni analoghe a quelle ottenute senza pruning.

### 3.3 Classificazione Multi-classe e Regressione

Il sistema descritto fin qui può essere esteso per problemi di classificazione non binaria e per problemi di regressione. L'unico punto critico riguarda l'algoritmo supervisionato di training della SOM nella SMF. Nel caso di classificazione non binaria si può comunque utilizzare l'algoritmo LVQ1, o la sua estensione utilizzata in questa tesi (descritta nel paragrafo 3.1.1), per la fase supervisionata, oppure l'algoritmo proposto in [77] che può trattare anche un numero elevato di classi target. Nel caso di regressione invece il training supervisionato della SOM diventa problematico, anche se di recente è stato proposto un algoritmo di LVQ per problemi di regressione [89] che potrebbe essere utilizzato nella fase di training supervisionato della SOM. In alternativa si può utilizzare l'algoritmo non supervisionato prendendo un numero maggiore di unità della SOM; infatti, effetti simili al training supervisionato si possono ottenere prendendo un numero sufficiente di unità, siccome, anche nel caso di training supervisionato, la separazione di vettori con diverso target su unità differenti avviene se i vettori sono diversi, per cui, prendendo abbastanza unità (e se il training è fatto in un numero sufficiente di passi, soprattutto nella fase fine), gli stessi vettori possono essere separati anche nel caso non supervisionato.

Un altro punto di cui si deve tenere conto, a seconda del tipo di problema, è il modo in cui si possono visualizzare le informazioni utili ad interpretare la risposta del sistema. Nel resto della tesi si faranno esempi su problemi di classificazione binaria che è il caso più comodo e semplice. Nel caso di classificazione non binaria si possono visualizzare informazioni analoghe in modo separato per ogni classe, considerando ogni classe in opposto a tutte le altre. Nel caso di regressione invece si può procedere come fatto di seguito per la classificazione binaria, ma ovviamente, per la natura del problema, risulterà più complessa la lettura delle informazioni.

### 3.4 Costi Computazionali

Il sistema descritto condivide con il paradigma del Reservoir Computing l'efficienza e la praticità dell'approccio. Il processo di encoding, fatto attraverso il reservoir, è lo stesso delle GraphESNs (si veda la sezione 2.1.8) e ha costo, assumendo un limite al grado massimo  $k$  di ogni vertice, lineare nel numero di vertici del grafo di input. Per un confronto, sotto la stessa assunzione, i costi dei kernel per grafi Optimal Assignment ed Expected Match, proposti in

[90], sono rispettivamente cubico e quadratico nel numero di vertici. Anche la SMF, con l'utilizzo della SOM per ripartire lo spazio degli stati, ha costo che scala linearmente nel numero di vertici dei grafi di input e nel numero di unità  $K$ . Infine, il training del readout, composto da un semplice strato di unità lineari, viene fatto utilizzando Elastic Net che viene risolto con attraverso l'algoritmo LARS-EN, tipicamente più efficiente rispetto ad altri tipi di readout, allenati con algoritmi iterativi basati su discesa del gradiente, o rispetto alle Support Vector Machines come nei metodi a kernel. Nel dettaglio, consideriamo separatamente il costo del processo di encoding, della SMF e del processo di output.

**Encoding** Il processo di encoding è lo stesso delle GraphESNs, un grafo di input viene mappato in un grafo nello spazio degli stati utilizzando il processo iterativo descritto nell'equazione 2.20. Dato un grafo di input  $\mathbf{g}$ , l'intero processo di encoding ha costo:

$$\mathcal{O}(T_F|V(\mathbf{g})|kN_R M) \quad (3.10)$$

dove  $T_F$  è il numero massimo di iterazioni necessarie a raggiungere il punto fisso del processo iterativo,  $|V(\mathbf{g})|$  è il numero di vertici del grafo  $\mathbf{g}$ ,  $k$  è il grado massimo per i vertici di  $\mathbf{g}$ ,  $N_R$  è il numero di unità del reservoir ed  $M$  è il numero massimo di unità a cui ogni unità del reservoir è connessa. Assumendo che i grafi da elaborare non siano completamente connessi, che il reservoir sia connesso in modo sparso e che si arrivi al punto fisso con poche iterazioni, allora il costo del processo di encoding scala in modo lineare rispetto al numero di vertici del grafo e rispetto al numero di unità del reservoir. Da notare che le assunzioni fatte rispecchiano situazioni reali (anche rispetto alle configurazioni classiche di ESN e GraphESN). Negli esperimenti fatti in questa tesi, in problemi di tossicologia, il reservoir è costruito in modo sparso, con il 25% delle connessioni, si arriva al punto fisso in al più qualche decina di passi iterativi e i grafi hanno grado massimo  $k = 4$ .

Siccome il reservoir non è allenato, il costo di questo processo è uguale sia in fase di test che in fase di training. In particolare, se il training set è composto da un numero  $T$  di grafi, per un totale di  $T_V$  vertici ( $T_V = \sum_{\mathbf{g} \in \mathcal{G}} |V(\mathbf{g})|$ ), sotto le assunzioni precedenti, il costo della fase di encoding nella procedura di training è:

$$\mathcal{O}(T_V N_R) \quad (3.11)$$

**State Mapping Function** Nella SMF descritta precedentemente le procedure relative alla SOM sono quelle di costo predominante: la costruzione in fase di training e il mapping degli stati in fase di test. Consideriamo la fase di training. Siano  $T_V$  il numero totale di vertici per tutti i grafi del training

set e  $T_E$  il numero totale di epoche nel training della SOM considerando ogni fase (grezza, fine e supervisionata). Ad ognuna delle  $T_E$  epoche vengono presentati alla SOM i  $T_V$  stati calcolati nella fase di encoding, corrispondenti ai rispettivi vertici di input. Per ogni stato viene cercata l'unità vincente con costo  $\mathcal{O}(KN_R)$ , dove  $K$  è il numero di unità della SOM, e vengono aggiornate le codebooks di ogni unità con costo  $\mathcal{O}(KN_R)$ . Il costo complessivo del training della SOM è quindi:

$$\mathcal{O}(T_E T_V K N_R) \quad (3.12)$$

che è lineare del numero di vertici di training, nel numero di unità della SOM e nella dimensione del reservoir.

In fase di test, dato un grafo di input  $\mathbf{g}$ , si deve mappare ogni stato, ottenuto dal processo di encoding, relativo ad un vertice del grafo, nelle unità della SOM. Questo viene fatto cercando l'unità vincente con costo  $\mathcal{O}(KN_R)$ . In fase di test la SMF ha quindi costo:

$$\mathcal{O}(|V(\mathbf{g})|KN_R) \quad (3.13)$$

per ogni grafo di input.

**Output** Il readout viene allenato con il metodo Elastic Net utilizzando l'algoritmo LARS-EN [24]. Siano  $T$  il numero di grafi nel training set e  $K$  il numero di unità della SOM, per ognuna delle  $N_Y$  righe  $\mathbf{w}_{\text{out}}^j$  della matrice dei pesi del readout  $\mathbf{W}_{\text{out}}$ , vengono trovati i  $K$  pesi con costo dell'ordine  $\mathcal{O}(K^3 + (T + K)K^2) = \mathcal{O}(K^3 + TK^2)$ . Considerando  $N_Y = 1$ , allora il training del readout ha costo:

$$\mathcal{O}(K^3 + TK^2) \quad (3.14)$$

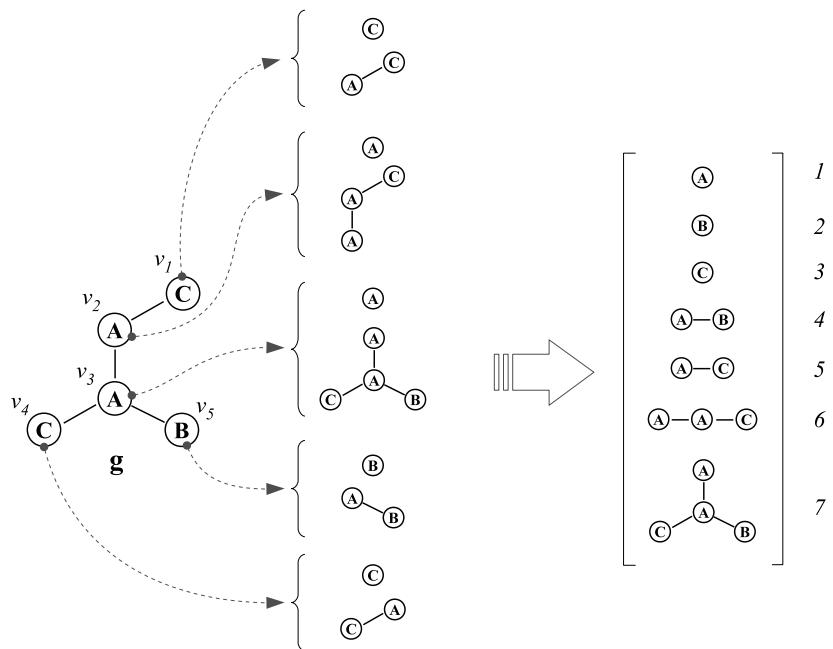
In fase di test l'output viene calcolato in accordo alla funzione di output 2.23, con costo  $\mathcal{O}(N_Y K)$ .

### 3.5 Un Modello di Confronto

In questa sezione viene introdotto un modello a descrittori chiamato Metodo a Frammenti (MF). Nel seguito, parlando di *frammenti* di un grafo  $\mathbf{g}$  ci riferiamo a sottografi di  $\mathbf{g}$ . In particolare ogni intorno di un vertice  $v \in V(\mathbf{g})$  (si veda la sezione 2.1) è un frammento per un grafo  $\mathbf{g}$ . L'idea di base di MF è di rappresentare ogni grafo con un vettore binario a dimensione fissata, in cui ogni elemento del vettore indica la presenza di un determinato frammento nel grafo stesso. I frammenti, associati ad ogni elemento del vettore binario, sono gli intorni di ogni vertice di un insieme di grafi in un training set. La risposta, per dato un grafo di input, si ottiene utilizzando il vettore che

rappresenta il grafo in un modello di regressione lineare, allenato sui dati del training set. Come spiegato meglio più avanti, il funzionamento di questo modello è analogo, per alcuni aspetti, a quello di GraphESN-SOM, in modo particolare se viene utilizzata la funzione di binarizzazione (utilizzata nelle prove sperimentali di questa tesi) come funzione di aggregazione nella SMF. L'analogia è utile in un confronto tra i due metodi per mettere in luce alcune importanti caratteristiche di GraphESN-SOM. Le differenze e le analogie tra i due metodi sono discusse nel paragrafo 3.5.2, mentre nella sezione 4.5 viene fatto un confronto sperimentalmente.

### 3.5.1 Metodo a Frammenti



**Figura 3.6.** Da ogni vertice del grafo  $\mathbf{g}$  si ricavano gli intorni di raggio  $r \in \{0, 1\}$  (al centro), dopodiché si crea l'insieme di frammenti composto dagli intorni unici ricavati dal grafo (sulla destra). Da notare che i vertici  $v_1$  e  $v_4$  hanno gli intorni uguali, quindi portano agli stessi frammenti 3 e 5.

L'idea di MF è quindi di scomporre un grafo in un insieme di frammenti, come illustrato in Figura 3.6, formati dall'intorno di raggio  $r = 0, \dots, R$  di ogni suo vertice per un fissato valore di  $R$ . Da una fase di training si ricavano tutti i frammenti, ottenuti in tale modo, per ogni grafo del training set, ottenendo un insieme di frammenti  $\mathcal{F}$ . Sia  $\text{int}_R(\mathbf{g})$  una funzione che, dato un grafo  $\mathbf{g}$  ed una costante  $R$ , restituisce l'insieme composto da tutti gli intorni di raggio  $r = 0, \dots, R$  di ogni vertice  $v \in V(\mathbf{g})$ . Dato un training

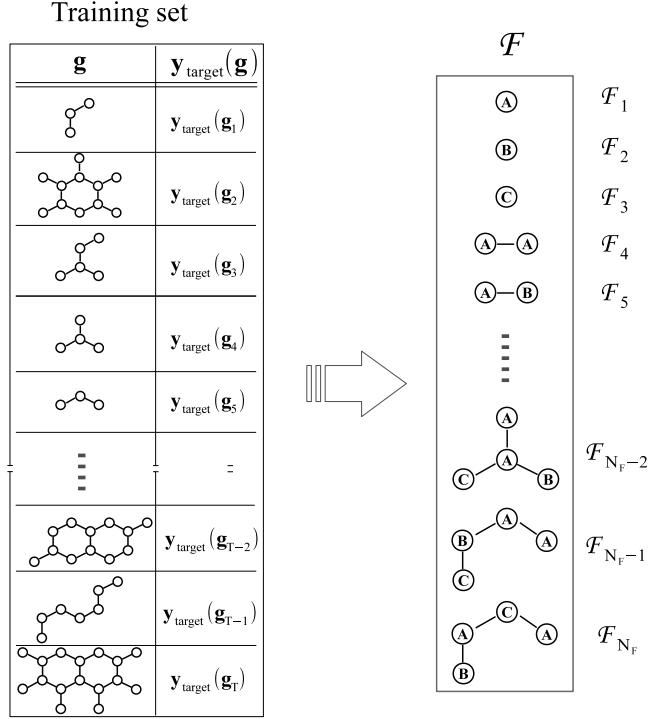
set  $\mathcal{T}$ , composto da  $T$  elementi, su un insieme di grafi di input  $\mathcal{G} \subseteq (\mathbb{R}^{N_U})^\#$ :

$$\mathcal{T} = \{\langle \mathbf{g}, \bar{\mathbf{y}}(\mathbf{g}) \rangle \mid \mathbf{g} \in \mathcal{G}, \bar{\mathbf{y}}(\mathbf{g}) \in \mathbb{R}^{N_Y}\}$$

e fissato un valore  $R$ , l'insieme  $\mathcal{F}$  si ricava come segue:

$$\mathcal{F} = \bigcup_{\langle \mathbf{g}, \bar{\mathbf{y}}(\mathbf{g}) \rangle \in \mathcal{T}} \text{int}_R(\mathbf{g}) \quad (3.15)$$

Indichiamo con  $N_F = |\mathcal{F}|$  il numero dei frammenti nell'insieme  $\mathcal{F}$  e assumiamo che sia definito un ordinamento (arbitrario ma fissato) sugli elementi di  $\mathcal{F}$ , allora denotiamo con  $\mathcal{F}_i$  l' $i$ -esimo elemento di  $\mathcal{F}$ , per  $i = 1, \dots, N_F$ . In Figura 3.7 è rappresentato l'insieme dei frammenti  $\mathcal{F}$  ottenuto a partire da un training set  $\mathcal{T}$ . Dato un grafo  $\mathbf{g}$  si crea un vettore  $\mathcal{F}(\mathbf{g}) \in \mathbb{R}^{N_F}$  in cui

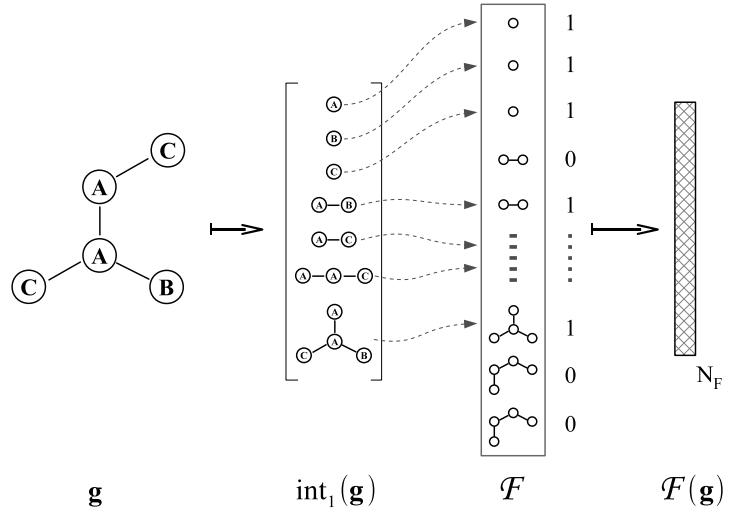


**Figura 3.7.** Si costruisce l'insieme di frammenti  $\mathcal{F}$  prendendo tutti gli intorni di raggio  $r = 0, \dots, R$ , per un qualche valore fissato di  $R$ , di ogni vertice  $v$  di ogni grafo del training set.

l' $i$ -esimo elemento è definito nel seguente modo:

$$[\mathcal{F}(\mathbf{g})]_i = \begin{cases} 1 & \text{se } \mathcal{F}_i \in \text{int}_R(\mathbf{g}) \\ 0 & \text{altrimenti} \end{cases} \quad (3.16)$$

come illustrato in Figura 3.8. Il vettore  $\mathcal{F}(\mathbf{g})$  è una rappresentazione a



**Figura 3.8.** Un grafo  $\mathbf{g}$  è scomposto in un insieme di frammenti formato dagli intorni di raggio  $r \in \{0, 1\}$  di ogni suo vertice (il risultato della funzione  $\text{int}_R(\mathbf{g})$ ), dopodiché si crea il vettore  $\mathcal{F}(\mathbf{g}) \in \mathbf{R}^{N_F}$  prendendo un elemento di valore 1 in corrispondenza di ogni frammento dell'insieme  $\mathcal{F}$  presente tra i frammenti ottenuti dal grafo  $\mathbf{g}$ , oppure un elemento di valore 0 per i frammenti che invece non fanno parte del grafo.

dimensione fissata di un grafo di input  $\mathbf{g}$ . Allora il modello di regressione lineare che restituisce la risposta  $\mathbf{y}(\mathbf{g})$  per un grafo di input  $\mathbf{g}$  con MF si può definire come segue:

$$\mathbf{y}(\mathbf{g}) = w_0 + \mathbf{w}\mathcal{F}(\mathbf{g})$$

dove  $\mathbf{w} \in \mathbf{R}^{1 \times N_F}$  è il vettore dei pesi e  $w_0 \in \mathbf{R}$  è il valore del bias. Per semplicità si può includere il bias come primo elemento del vettore  $\mathbf{w}$ , aggiungendo un elemento con valore fissato a +1 all'inizio del vettore  $\mathcal{F}(\mathbf{g})$ , l'equazione diventa quindi:

$$\mathbf{y}(\mathbf{g}) = \mathbf{w}\mathcal{F}(\mathbf{g}) \quad (3.17)$$

I valori del vettore  $\mathbf{w}$  possono essere trovati con un qualunque metodo lineare come quello utilizzato per il training del readout in GraphESN-SOM. In particolare può essere utilizzato Elastic Net anche in questo caso, effettuando un pruning dei pesi in  $\mathbf{w}$ , associati ai frammenti dell'insieme  $\mathcal{F}$ , che porta ad una selezione di un sottoinsieme di frammenti, scartando i frammenti a cui viene associato un peso nullo.

### 3.5.2 Analogie e Differenze con GraphESN-SOM

Il vettore  $\mathcal{F}(\mathbf{g})$  è una rappresentazione a dimensione fissata di un grafo di input  $\mathbf{g}$  ottenuta con MF, ed è l'analogo del vettore  $\mathcal{X}(\mathbf{g}_r)$  (descritto nella sezione 3.1) restituito dalla SMF in GraphESN-SOM con la funzione di binarizzazione come funzione di aggregazione. Tali vettori sono utilizzati da entrambi i metodi, allo stesso modo, in un modello lineare per il calcolo dell'output. Nel vettore  $\mathcal{F}(\mathbf{g})$  gli elementi con valore 1 codificano la presenza nel grafo  $\mathbf{g}$  di determinati frammenti ottenuti dagli intorni dei vertici del grafo. In GraphESN-SOM, poiché uno stato del reservoir  $\mathbf{x}(v)$  è una rappresentazione del vertice  $v$  e del suo intorno con effetti Markoviani (come spiegato nella sezione 3.1), le unità della SOM attivate dagli stati dei vertici di un grafo dipendono dall'insieme dei vertici stessi e dai loro intorni, il vettore  $\mathcal{X}(\mathbf{g}_r)$  è quindi un vettore in cui gli elementi con valore 1 codificano, in modo simile a MF, la presenza nel grafo  $\mathbf{g}$  di un vertice con un certo tipo di intorno. Perciò in entrambi i modelli la risposta per un grafo di input è in funzione dell'insieme dei vertici del grafo stesso e dei loro intorni. Questa somiglianza rende interessante un confronto dettagliato tra i due modelli, fatto anche con alcune prove sperimentali nella sezione 4.5.

Alcuni punti importanti differenziano i due metodi. Prendiamo una situazione d'esempio in cui, dopo aver allenato una GraphESN-SOM ed un MF su uno stesso training set, si vuole ottenere una predizione per un grafo in cui è presente un nuovo frammento  $\mathcal{F}_n$  che non appare in nessun grafo di training. Con MF il frammento  $\mathcal{F}_n$  e tutti i frammenti che includono  $\mathcal{F}_n$  vengono scartati dal calcolo della risposta, siccome non appartengono all'insieme  $\mathcal{F}$ , per cui non possono influire sulla predizione. Con GraphESN-SOM, invece, ogni vertice del grafo che è parte del nuovo frammento  $\mathcal{F}_n$  produce uno stato del reservoir mappato nell'unità della SOM che meglio lo rappresenta; inoltre, gli stati del reservoir di ogni altro vertice del grafo sono influenzati dalla presenza del frammento  $\mathcal{F}_n$ . Per cui, in GraphESN-SOM, il vettore restituito dalla SMF risente anche di frammenti nuovi. GraphESN-SOM risulta quindi essere un metodo più flessibile rispetto a MF, siccome la predizione, al contrario di MF, può tenere conto anche di frammenti non presenti nei dati di training. Per gli stessi motivi GraphESN-SOM è anche maggiormente robusto e meno sensibile ad errori (o mancanze) nei dati di training.

Un'altra differenza riguarda la possibilità di aggiungere valori continui nelle etichette vettoriali associate ai vertici dei grafi. Definendo MF si è assunto implicitamente che le etichette possano avere solo valori in un insieme discreto e finito di elementi (codificati come vettori nello spazio  $\mathbb{R}^{N_U}$ ), per esempio simboli di un alfabeto finito. In molti casi può essere necessario utilizzare anche valori continui per codificare certe informazioni che riguardano il vertice. Ad esempio, in ambito chimico, rappresentando molecole come grafi dove i vertici sono gli atomi, si possono includere misure,

con valori continui, di caratteristiche fisico/chimiche degli atomi nella codifica del vertice, in aggiunta ad altre informazioni come la codifica del tipo di atomo stesso. In altri termini, ci sono casi in cui le etichette potrebbero avere un qualunque valore nello spazio  $\mathbb{R}^{N_U}$ , ed ogni etichetta di un vertice avere quindi un valore differente. In GraphESN-SOM questi casi sono gestiti in modo naturale dal modello, senza bisogno di alcuna particolare attenzione, per gli stessi motivi per cui, come spiegato nell'esempio precedente, è in grado di elaborare anche dati non presenti nel training set. Opposto invece è il comportamento di MF che, per come lo si è definito, non può trattare in modo appropriato questi casi. Infatti, per la mancanza di flessibilità del metodo, tutti i frammenti di un grafo non presente nel training set verrebbero scartati dal calcolo della risposta, a meno che non ci sia una corrispondenza esatta con un frammento nell'insieme  $\mathcal{F}$ , che nel caso di etichette con valori continui accadrebbe raramente. Inoltre, lo stesso insieme di frammenti  $\mathcal{F}$  risulterebbe composto da un numero eccessivo di elementi, ognuno dei quali rappresenterebbe un caso estremamente specifico. Tutto questo porterebbe ad un modello praticamente inutile, in grado di dare una risposta in uscita solo per i grafi già presenti nel training set.

Infine, un'ulteriore differenza, che riguarda in generale tutti i metodi basati su descrittori, compreso quindi MF. GraphESN-SOM, a differenza di tali metodi, può catturare aspetti topologici significativi nella struttura del grafo in modo adattivo. Questo è possibile grazie all'informazione contenuta in modo implicito nello stato prodotto dal reservoir per ogni vertice di un grafo, nel quale è codificata, con effetti Markoviani, l'intera struttura del grafo stesso. Tale informazione può essere sfruttata dal training supervisionato della SOM che può disporre le unità in modo da cogliere aspetti significativi rispetto al problema in esame. La mappatura dei vertici di un grafo sulle unità della SOM, che porta alla rappresentazione a dimensione fissata  $\mathcal{X}(\mathbf{g}_r)$  restituita dalla SMF, dipende quindi dal problema ed è costruita in modo supervisionato sul training set. Nei metodi a descrittori invece il grafo viene scomposto in un insieme di caratteristiche ed informazioni che devono essere decise a priori. In MF questo avviene scegliendo il parametro  $R$  con il quale dev'essere stabilita a priori l'ampiezza massima dell'intorno di ogni vertice. Nella scelta di questo valore si deve tenere conto anche dei costi computazionali necessari per il calcolo di tutti i frammenti, che sono esponenziali in  $R$ , e della dimensione dell'insieme di frammenti  $\mathcal{F}$  che se è eccessiva può portare a problemi nel training del modello di regressione lineare. In GraphESN-SOM invece ogni stato del reservoir racchiude l'informazione su ogni possibile intorno del vertice ed è calcolato con costo lineare nel numero di vertici del grafo.



# Capitolo 4

# Risultati Sperimentali

In questo capitolo sono riportati i risultati di un insieme di prove sperimentali sul sistema GraphESN-SOM introdotto in questa tesi. Gli esperimenti riguardano problemi di tossicologia ottenuti da 3 dataset descritti in appendice A. Una prima parte di prove ha lo scopo di testare le singole componenti del sistema e giustificare una serie di scelte modellistiche (come l'utilizzo di valori binari nella State Mapping Function). Le scelte fatte portano ad una configurazione del sistema utilizzata nelle prove successive per un confronto con altri sistemi. In particolare, nella sezione 4.2 viene confrontato il training supervisionato della SOM, nel funzionamento della GraphESN-SOM, con una versione non supervisionata. Nella sezione 4.3 vengono confrontate la funzione di binarizzazione e la funzione di somma come funzioni di aggregazione nella SMF ed un ulteriore confronto è fatto rispetto ad una SMF che non utilizza la funzione di aggregazione. Nella sezione 4.4 vengono approfonditi gli effetti del pruning delle connessioni del readout (i.e. del training via Elastic Net). Nella sezione 4.5 il sistema GraphESN-SOM è confrontato con il Metodo a Frammenti introdotto nella sezione 3.5. Nella sezione 4.6 viene fatto un confronto con altri modelli di Reservoir Computing per grafi e con risultati allo stato dell'arte su ognuno dei problemi. Infine, nella sezione 4.7 viene riportato un esempio di funzionamento del sistema nell'analisi di una predizione.

## 4.1 Configurazioni di Base

I risultati riportati nel resto del capitolo, riferiti a GraphESN-SOM, sono ottenuti con la procedura e a partire dalle configurazioni di base descritte in questa sezione. I datasets utilizzati per le sperimentazioni sono Bursi, ISSCAN e PTC, descritti nell'appendice A. I tre dataset definiscono complessivamente 6 problemi di tossicologia su insiemi di molecole. In particolare, i dataset Bursi e ISSCAN definiscono problemi di classificazione binaria di mutagenicità, in cui una molecola è classificata come mutagenica (attiva)

o non mutagenica (non attiva). Per il problema definito sul dataset Bursi ci riferiamo semplicemente con il nome del dataset, mentre per il problema definito sul dataset ISSCAN parleremo di “ISSCAN (SAL)”. Il dataset PTC definisce 4 problemi di classificazione binaria di cancerogenicità per 4 differenti tipi di roditori: topi maschi (MM), topi femmine (FM), ratti maschi (MR), ratti femmine (FR). In questo caso una molecola è considerata attiva se è classificata come cancerogena, non attiva altrimenti. Ci riferiamo ai quattro problemi definiti su PTC con “PTC (MM)”, “PTC (MR)”, “PTC (FM)”, “PTC (FR)” a seconda del tipo di roditore.

Per ogni dataset le molecole sono rappresentate come grafi indiretti (senza distinguere tra tipi differenti di legami tra atomi), con grado massimo  $k = 4$  (che è il grado massimo riscontrato su ognuno dei 3 dataset). Le etichette di ogni vertice sono composte da  $m$  componenti per la rappresentazione binaria 1-of- $m$  dell’atomo associato al vertice, una componente per la carica dell’elemento atomico (con valori  $-1, +1$  oppure  $0$ ) ed una componente per indicare se l’atomo è aromatico (con valore  $+1$  se lo è,  $0$  altrimenti), questo porta ad avere la dimensione di input  $N_U = 16$  per Bursi,  $31$  per ISSCAN (SAL) e  $24$  per i 4 problemi di PTC. I problemi sono tutti di classificazione binaria, quindi  $N_Y = 1$ , con target  $+1$  ad indicare le molecole attive,  $-1$  quelle inattive.

Le performance, in termini di accuracy di test, per ogni parametrizzazione, sono ottenute dalla media su 5 inizializzazioni differenti del reservoir (cioè inizializzazioni differenti delle matrici  $\mathbf{W}_{\text{in}}$  e  $\hat{\mathbf{W}}$ ). L’inizializzazione delle unità della SOM è casuale ma costante (a parità di numero di unità e dimensione delle codebooks). Il reservoir è di dimensione  $N_R = 30$ , con connettività del 25%. Il fattore di scala dell’input è  $w_{\text{in}} = 0.01$ , ovvero i valori della matrice dei pesi  $\mathbf{W}_{\text{in}}$  sono inizializzati in modo casuale tra  $-0.01$  e  $+0.01$ . Il coefficiente di contrattività  $\sigma$  è scelto dalla procedura di model selection, tra un insieme di valori generalmente compresi tra  $1$  e  $3$ . Da notare che prendendo valori di  $\sigma$  maggiori di  $1$  si viola la condizione di contrattività  $2.26$  e non si ha garantita la convergenza del processo di encoding iterativo. Nei casi in cui il processo di encoding non converga, per una certa inizializzazione del reservoir, tale inizializzazione viene scartata e ne viene provata un’altra. Le performance su ogni dataset sono ottenute con una procedura di *cross-validation* stratificata con *5 folds* (parti) per i quattro problemi di PTC e per ISSCAN (SAL), mentre per Bursi è utilizzato il test set (già separato nel dataset originale). La test accuracy riportata nelle tabelle seguenti è la media sulle 5 folds della cross-validation per i quattro problemi di PTC e ISSCAN (SAL), in questo caso tra parentesi è riportata la deviazione standard delle performance delle 5 inizializzazioni casuali del reservoir mediata sulle 5 folds, e quella ottenuta sul test set nel caso di Bursi, in questo caso tra parentesi è riportata la deviazione standard delle performance ottenute sul test set dalle 5 inizializzazioni casuali del reservoir. I parametri (che non fissati in questa sezione) sono scelti da una procedura

di model selection utilizzando una ulteriore cross-validation stratificata con 5 folds per ogni fold di training in PTC ed ISSCAN, e sul training set in Bursi.

I risultati ottenuti con una configurazione di base di GraphESN-SOM sono utilizzati nelle sezioni successive come confronto con varianti di tale configurazione o con modelli differenti. In questa configurazione di base è utilizzata la funzione di binarizzazione come funzione di aggregazione (descritta nella sezione 3.1.3), il training della SOM è fatto con la procedura supervisionata descritta nella sezione 3.1.1 e il readout è allenato con Elastic Net. La dimensione della mappa della SOM è fissata ad un valore ragionevole per ogni problema, sulla base di considerazioni generali sul problema stesso (rumorosità e approssimazione nota del target<sup>1</sup>), sulla qualità dei clusters e sui tempi di training. Le mappe sono quindi di dimensione  $50 \times 30$  unità ( $K = 1500$ ) per Bursi,  $27 \times 16$  unità ( $K = 432$ ) per ISSCAN (SAL) e PTC (MR),  $34 \times 20$  unità ( $K = 680$ ) per PTC (FM), PTC (FR) e PTC (MM). Con la procedura di model selection sono selezionati i valori per il coefficiente  $\sigma$  ed i parametri  $\lambda_1$  e  $\lambda_2$  di Elastic Net. In modo uguale per ogni problema, il coefficiente sigma è stato scelto tra l'insieme di valori che vanno da 1.0 a 2.8 con passo 0.2, mentre i parametri  $\lambda_1$  e  $\lambda_2$  sono presi sempre uguali l'uno all'altro con valori nell'insieme  $\{0.1, 1, 2, 4, 8, 16\}$ .

## 4.2 SOM Supervisionata

In questa sezione viene illustrato il funzionamento della SOM supervisionata confrontandolo, nel funzionamento della GraphESN-SOM, con una versione non supervisionata. Il confronto viene fatto prima mostrando un esempio rappresentativo in cui si vede che l'algoritmo supervisionato riesce a separare alcuni elementi di training (gli stati associati ai vertici dei grafi) in differenti unità della SOM grazie al target differente, mentre l'algoritmo non supervisionato non riesce a fare altrettanto. Dopodiché si confrontano le performance in termini di generalizzazione, con i risultati ottenuti utilizzando entrambe le versioni nei 6 problemi di tossicologia.

### 4.2.1 Risultati

La procedura di training supervisionato della SOM, utilizzata nelle prove riportate di seguito, è stata fatta come descritto nella sezione 3.1.1. I parametri, per ognuna delle tre fasi di training, sono presi in funzione della dimensione della SOM e del numero di elementi nel training set, nel modo riportato nella Tabella 4.1.

---

<sup>1</sup> Alcuni dei dataset presi in considerazione utilizzano come valore target il risultato del test di laboratorio *Ames test* per la tossicità di un composto, il quale ha un'affidabilità di circa l'85% [91]. Ottenere un fitting troppo elevato significherebbe includere certamente errori nel modello.

#	$T_E$	$\alpha_{\text{ini}}$	$\rho_{\text{ini}}$	$\rho_{\text{fin}}$
(1)	$\lceil 10 K/T_V \rceil$	0.5	$\max(1, \lceil \frac{\max(N_{\text{row}}, N_{\text{col}})}{4} \rceil)$	$\max(1, \lceil \rho_{\text{ini}}^{(1)}/4 \rceil)$
(2)	$4 T_E^{(1)}$	$\alpha_{\text{ini}}^{(1)}/10$	$\rho_{\text{fin}}^{(1)}$	1
(3)	$T_E^{(2)}$	$\alpha_{\text{ini}}^{(2)}$	$\rho_{\text{fin}}^{(2)}/2$	0.1

**Tabella 4.1.** Elenco dei parametri utilizzati nelle prove per ognuna delle tre fasi di training. Le fasi sono: (1) fase grezza, (2) fase fine e (3) fase supervisionata.  $N_{\text{row}}$  ed  $N_{\text{col}}$  sono rispettivamente il numero di righe e il numero di colonne della griglia della SOM,  $K = N_{\text{row}}N_{\text{col}}$  è il numero totale di unità e  $T_V = |\mathcal{T}_V|$  è il numero di elementi nel training set della SOM  $\mathcal{T}_V$ .

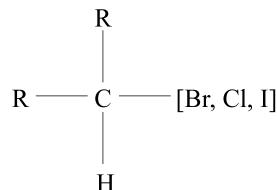
Per un confronto “equo” (su uno stesso numero di epoch), anche la procedura di training non supervisionato è stata fatta con tre fasi: le prime due, la fase grezza e la fase fine di training non supervisionato, in comune con la procedura supervisionata, mentre la terza fase in questo caso è un’ulteriore fase non supervisionata, fatta con l’Algoritmo 3.1 (lo stesso utilizzato per le prime due fasi). I parametri invece sono gli stessi della procedura supervisionata, in Tabella 4.1, per ognuna delle tre fasi, dove chiaramente la terza fase in questo caso è non supervisionata come le prime due. Quindi, nel training supervisionato e nel training non supervisionato utilizzati di seguito, le prime due fasi sono uguali, mentre si differenziano unicamente nell’ultima fase, che è una fase supervisionata nel primo caso, oppure una fase addizionale di training fine, in cui i valori delle codebook vengono ulteriormente raffinati, nel secondo caso.

Per vedere gli effetti del training supervisionato nella GraphESN-SOM prendiamo un caso in cui un insieme di stati simili tra loro ha una distribuzione dei valori target fortemente sbilanciata verso un valore (positivo o negativo), ed un altro insieme di stati simili tra loro ha una distribuzione dei valori target sbilanciata verso il valore opposto, e che tra i valori degli stati dei due insiemi ci sia una differenza piccola ma significativa. In questi casi, infatti, il training non supervisionato può faticare molto a raccogliere su unità diverse gli stati dei due diversi insiemi (cioè ad assegnare valori a due o più codebooks in modo tale che ognuna rappresenti un diverso insieme di stati), ma piuttosto tenderà a raccoglierli in una sola unità, poiché non ha motivo di separarli (se la differenza non è sufficientemente grande). Invece, il training supervisionato, condizionato anche dal target degli stati, dovrebbe riuscire meglio a separare su due (o più) unità gli stati dei due diversi insiemi. Questa differenza tra i due tipi di training è più evidente quando gli stati dei due insiemi (con target prevalente di valore opposto) sono molto simili tra loro, oppure quando tali stati sono in numero limitato sul training set, in questo caso, infatti, il training non supervisionato dovrebbe compiere un elevato numero

di iterazioni per riuscire a separarli in unità differenti in base solamente al loro valore. Potenzialmente, però, anche il training non supervisionato può separare i due insiemi di stati (anche senza conoscere il target), se questi hanno valori differenti, a patto di avere un numero abbastanza elevato di unità e di compiere abbastanza iterazioni dell'algoritmo di training in modo che diverse unità si dispongano in modo da raccogliere i diversi insiemi di stati. Da notare, comunque, che gli stati devono avere valore diverso per essere separati, anche utilizzando il training supervisionato, infatti se due stati hanno lo stesso valore ma diverso target vengono sempre e comunque mappati nella stessa unità.

Vediamo allora un caso in cui si può avere una situazione come quella descritta sopra nei problemi di tossicologia presi in esame in questi esperimenti. In Figura 4.1 è rappresentata la *structural alert SA\_8* (presa dalla collezione di SAs per mutagenicità in [85]), composta da un atomo alogeno (cloro (Cl), bromo (Br), o iodio (I)), legato ad un atomo di carbonio (C) alifatico. Gli

#### **SA\_8: Aliphatic alogens**

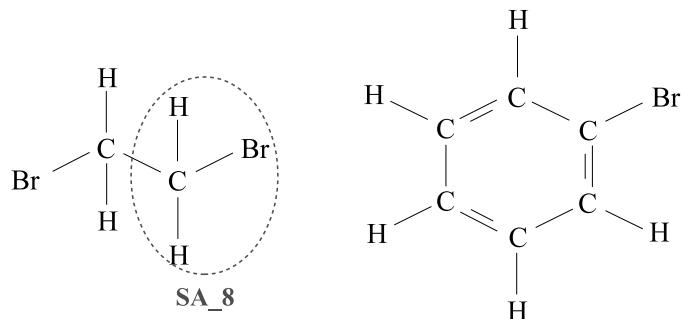


R = qualsiasi atomo/gruppo

**Figura 4.1.** *Structural alert SA\_8* [85], composta da un atomo di bromo (Br), o cloro (Cl) o iodio (I), legato ad un carbonio (C) alifatico.

Atomi alifatici sono gli atomi che non fanno parte di un anello aromatico e si distinguono, appunto, dagli atomi aromatici che invece fanno parte di un anello, come rappresentato in Figura 4.2.

Come già detto nella sezione 2.2.2, le SAs indicano che è stata riscontrata una correlazione tra attività mutagenica (o cancerogena) e la presenza della sottostruttura descritta dalla SA nel composto chimico. Se infatti prendiamo il dataset Bursi (il cui problema è di mutagenicità) e verifichiamo il target dei composti che contengono la SA\_8, circa il 75% di queste ha target positivo. Se invece prendiamo i composti che contengono un alogeno (Br, Cl o I) legato ad un carbonio aromatico, quelli con target positivo sono solamente il 25% circa. Nella GraphESN-SOM, prendendo la rappresentazione a grafo delle molecole e considerando i vertici relativi agli alogenzi collegati ad un carbonio, tali vertici avranno uno stato del reservoir simile tra loro (a parità di elemento chimico, cioè un bromo avrà uno stato simile ad un altro bromo, un cloro ad un cloro, e così via) siccome anche il loro intorno di raggio uno è simile (si veda la sezione 2.1.8), composto da un atomo di carbonio, mentre



**Figura 4.2.** Nella molecola di sinistra l'atomo di bromo (Br) è legato ad un atomo di carbonio (C) alifatico, riconoscibile dal fatto che è in una catena aperta di carboni e non in un anello. Nella molecola di destra, invece, l'atomo di bromo (Br) è legato ad un atomo di carbonio (C) aromatico (che fa parte di un anello). Quindi, la *structural alert* SA\_8 (in Figura 4.1) combacia solamente con la sottostruttura bromo-carbonio della molecola di sinistra e non con quella della molecola di destra.

il target associato ad essi sarà positivo nella maggior parte dei vertici se il carbonio è alifatico, mentre sarà perlopiù negativo se il carbonio è aromatico. Questo ci porta alla situazione in cui possiamo considerare due insiemi di stati, in cui i valori degli stati di ogni insieme sono simili tra loro, tra i due insiemi di stati c'è una piccola ma significativa differenza, ed ogni insieme ha una prevalenza di valori target opposta all'altro. Possiamo quindi vedere se il training supervisionato riesce a separare meglio (su più unità della SOM) gli stati dei due insiemi rispetto al training non supervisionato. I due insiemi presi in considerazione sono composti rispettivamente dagli stati dei vertici relativi agli atomi di un certo alogeno (bromo, cloro oppure iodio) legato ad un carbonio alifatico per l'insieme a prevalenza di target positivi, ad un carbonio aromatico per l'altro insieme.

Ricordiamo che nella codifica dell'atomo utilizzata in questi esperimenti (vedi sezione 4.1) viene utilizzata una componente per indicare se l'atomo è aromatico, per cui lo stato relativo ad un alogeno legato ad un carbonio aromatico risente di questa componente e si differenzia meglio da quello legato ad un carbonio alifatico. In altri termini, gli intorni di raggio 1 dei vertici relativi agli atomi di un carbonio aromatico sono diversi da quelli relativi agli atomi di un carbonio alifatico. Senza la codifica dell'aromaticità gli stati associati a carboni aromatici sono comunque simili tra loro e si differenziano da quelli dei carboni alifatici, infatti i primi sono ottenuti da vertici che fanno parte di una struttura ad anello di carboni che quindi avranno sempre un intorno molto simile tra loro, diverso da quello dei vertici dei carboni alifatici. Di conseguenza, gli stati dei vertici di atomi di alogeni legati ad un carbonio sarebbero comunque differenti nel caso di carbonio aromatico o alifatico, anche senza la codifica esplicita dell'aromaticità. Ovviamente però questa è molto utile

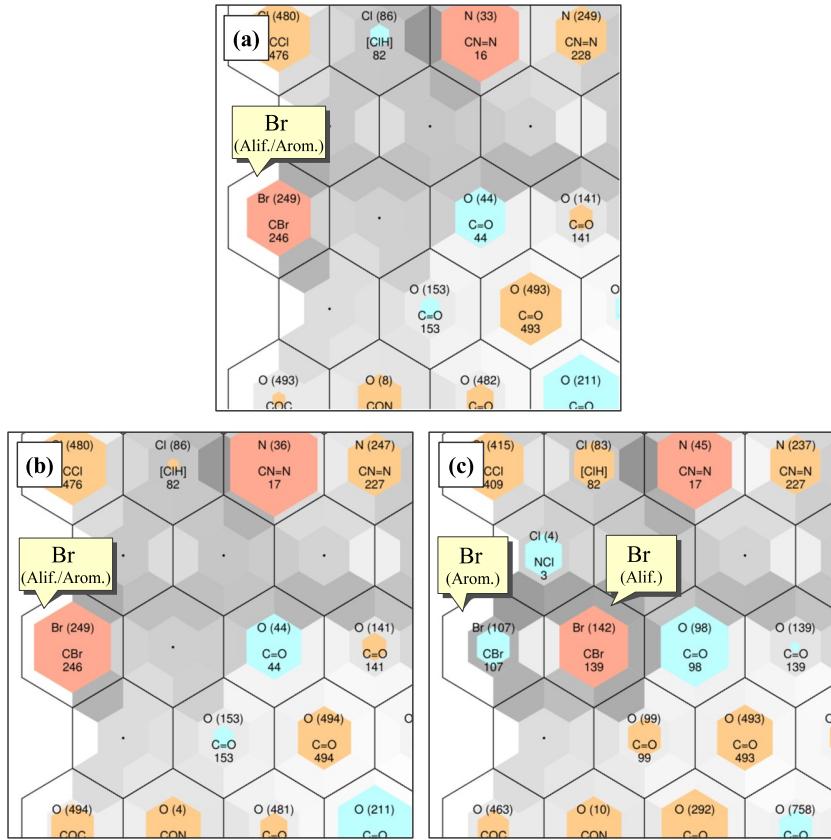
per aumentare la differenza nei due casi.

Vediamo allora un esempio riguardo i vertici di alogeni appena descritti. Il dataset utilizzato è Bursi per via del maggior numero di composti e dell'ottima qualità dei dati rispetto agli altri dataset presi in considerazione. In Figura 4.3 sono riportati tre riquadri, ognuno dei quali riguarda una mappa ottenuta da una GraphESN-SOM, allenata sul dataset Bursi, partendo sempre dalla stessa configurazione iniziale, con un numero  $K = 420$  di unità nella SOM. I riquadri sono sulla cella che raccoglie gli stati dei vertici degli atomi di bromo legati ad un atomo di carbonio.

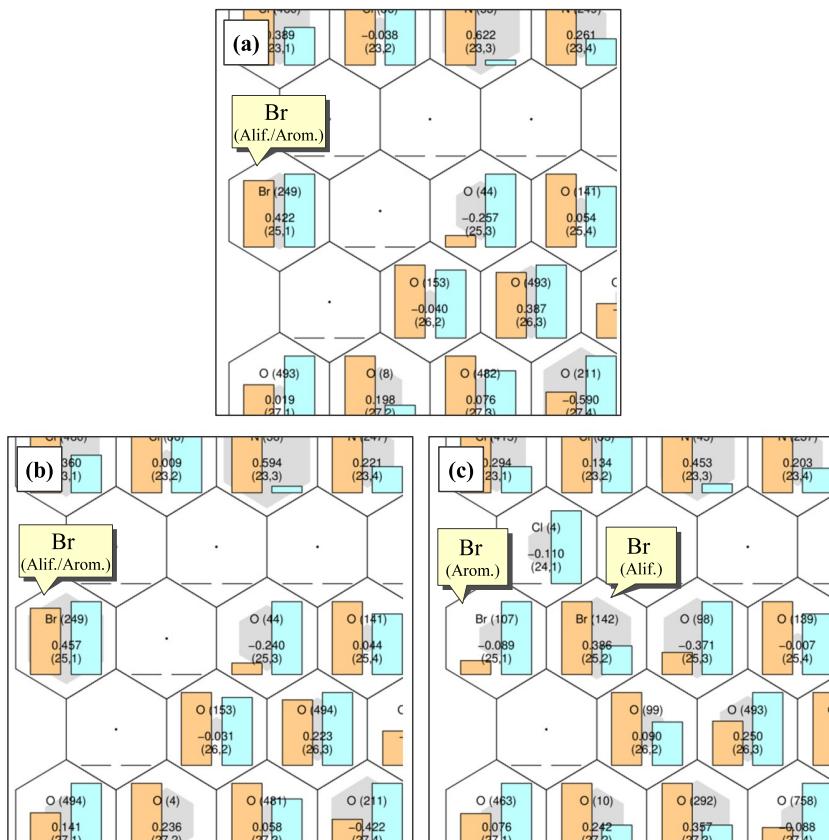
Nel riquadro (a) il training della SOM è stato fermato dopo le prime due fasi e rappresenta quindi la situazione in cui arrivano sia il training non supervisionato, sia il training supervisionato, e da cui poi la terza fase agisce in modo differente nelle due procedure di training. In ogni caso la GraphESN-SOM è stata allenata completamente e i pesi assegnati nel readout sono rappresentati graficamente in corrispondenza di ogni cella (che rappresenta una unità della SOM). Nel riquadro (b) il training della SOM è completato con le terza fase non supervisionata, nel riquadro (c) invece la terza fase è quella supervisionata. Nei riquadri (a) e (b), dove il training è non supervisionato, una sola cella raccoglie tutti gli atomi di bromo senza discriminare tra quelli legati ad un tipo diverso di carbonio (aromatico o alifatico). Per capire il tipo di carbonio legato ad ogni atomo di bromo, indicato in figura dall'etichetta a sfondo giallo, è stata fatta un'analisi del tipo di vertici raccolti dalla cella visualizzando le molecole di training con un vertice catturato dalla cella in esame, evidenziando tale vertice (in questo modo è facile capire visivamente il tipo di vertici catturati guardando caso per caso nelle molecole visualizzate). Nel riquadro (c), invece, come previsto, la fase di training supervisionata separa su due diverse unità i due diversi tipi di vertici degli atomi di bromo.

In Figura 4.4 è riportato lo stesso esempio visualizzando però un istogramma dei valori target dei vertici raccolti su ogni cella. Da questa figura si vede chiaramente che con il training supervisionato i vertici vengono separati in celle diverse grazie al fatto che hanno una prevalenza di target differenti, nella cella di sinistra vengono lasciati i vertici con target prevalentemente negativo, nella cella di destra finiscono i vertici con target prevalentemente positivo, che corrispondono, come si ci aspettava, ad atomi di bromo legati, rispettivamente, ad atomi di carbonio aromatici e ad atomi di carbonio alifatici. Quindi, a partire dalla situazione del riquadro (a), ottenuta dopo le prime due fasi di training, il training supervisionato separa i vertici, relativi agli atomi di bromo, su due unità in modo tale che ognuna raccolga vertici che hanno lo stato del reservoir simile tra loro ma con una distribuzione del target (fortemente) sbilanciata verso un valore, mentre il training non supervisionato lascia la situazione inalterata, mantenendo tutti i vertici in un'unica cella.

Nelle figure precedenti si può inoltre vedere come la GraphESN-SOM

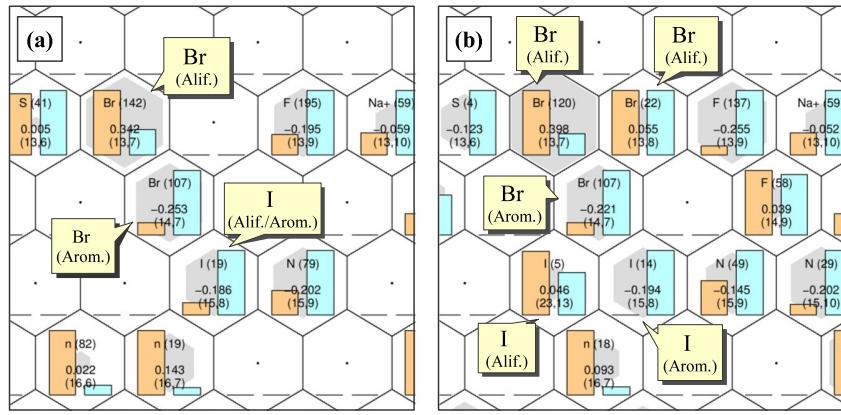


**Figura 4.3.** In ogni riquadro è riportata una parte di una mappa ottenuta dal training di una GraphESN-SOM a partire dalla stessa configurazione iniziale. In ogni cella è riportato, in alto, l'elemento chimico più frequente associato ai vertici raccolti dalla cella con a fianco la frequenza stessa, sotto invece c'è la stringa SMILES [92] dell'intorno di raggio 1 più frequente tra tutti i vertici raccolti (con sotto la frequenza). Ad esempio, la cella del bromo (Br), nel riquadro (a), ha raccolto 249 vertici di bromo, dei quali 246 sono connessi ad un atomo di carbonio. Sullo sfondo (in grigio) c'è la U-Matrix della SOM, mentre in corrispondenza di ogni cella c'è la rappresentazione grafica dei pesi assegnati dal readout della GraphESN-SOM, in rosso o arancione i pesi di segno positivo, in azzurro i pesi di segno negativo, senza nulla se il peso è nullo. Nel riquadro (a) il training della SOM è stato fatto con le due fasi non supervisionate (fase grezza e fase fine), in (b) la SOM è allenata come in (a) con in più un'ulteriore fase finale non supervisionata, in (c) la SOM è allenata come in (a) con in più la fase finale supervisionata, la quale separa i vertici di bromo su due celle.



**Figura 4.4.** I tre riquadri corrispondono alle stesse mappe della figura 4.3, qui però in corrispondenza di ogni cella c'è un istogramma che rappresenta la percentuale dei diversi valori target associati ai vertici raccolti dalla cella stessa, in arancione i target positivi (+1), in azzurro i target negativi (-1). Nel riquadro (c) è evidente l'effetto del training supervisionato che porta alla separazione dei vertici degli atomi di bromo guidata dal valore target.

può trarre vantaggio dal training supervisionato della SOM, assegnando pesi di segno opposto alle unità che raccolgono i due tipi diversi di bromo rispecchiando le distribuzioni dei valori target, mentre assegna un peso positivo in modo indiscriminato a tutti i vertici di un bromo se il training è non supervisionato nonostante la distribuzione dei valori target sia uniforme (e non sbilanciata verso target positivi). Un altro esempio del tutto analogo a quello appena descritto è riportato in Figura 4.5. In questo caso la SOM è



**Figura 4.5.** Come in Figura 4.4, in ogni riquadro è riportata una parte di una mappa ottenuta dal training di una GraphESN-SOM a partire dalla stessa configurazione iniziale. Nel riquadro (a) il training della SOM è stato fatto in modo non supervisionato, mentre nel riquadro (b) la SOM è stata allenata con il training supervisionato. In questo caso anche il training non supervisionato separa i vertici degli atomi di bromo poiché la SOM è composta da un numero maggiore di unità (rispetto all'esempio precedente). Ma il training supervisionato fa comunque qualcosa in più: isola in una cella tutti i vertici di atomi di iodio (I) legati ad un carbonio aromatico siccome hanno tutti target negativo, e separa i vertici degli atomi di bromo alifatici lasciando nella cella di sinistra una maggiore percentuale di vertici a target positivo.

costruita con 680 unità, di più quindi rispetto all'esempio precedente. Questo fa sì che i vertici degli atomi di bromo sono separati anche con il training non supervisionato, nel riquadro (a). Il training supervisionato però, in più, separa i vertici degli atomi di iodio (I) a seconda se sono legati ad un carbonio aromatico o alifatico, in accordo alla SA\_8, in modo del tutto simile all'esempio precedente per gli atomi di bromo. Da notare che, siccome gli atomi di iodio sono in numero molto ridotto (19 in tutto), difficilmente questi vengono separati da un training non supervisionato, se non dopo molte iterazioni della procedura di training in modo da presentare alla SOM gli stati di tali vertici un numero elevato di volte. Un altro miglioramento apportato dalla procedura supervisionata, nel riquadro (b), rispetto a quella non supervisionata, è nella separazione degli atomi di bromo legati ad un carbonio alifatico (in alto nella figura), facendo in modo che un gruppo di

vertici con target uniformemente distribuito venga raccolto da una cella a fianco, lasciando così una percentuale maggiore di vertici con target positivo nella cella di sinistra.

Per concludere il confronto tra i due tipi di training vediamo le prestazioni, in termini di generalizzazione, ottenute dalla GraphESN-SOM sui dataset di tossicologia utilizzando entrambi i metodi di training della SOM. In Tabella 4.2 è riportata l'accuracy di test per ogni problema, utilizzando i due tipi di training. Nel caso della SOM supervisionata, sono riportati i risultati ottenuti con la configurazione di base della GraphESN-SOM descritta nella sezione 4.1. Nel caso della SOM non supervisionata sono invece riportati i risultati ottenuti con la stessa procedura, a partire dalle stesse configurazioni, utilizzando però il training non supervisionato della SOM.

<b>Dataset</b>	<b>SOM non supervisionata</b>	<b>SOM supervisionata</b>
Bursi	82.03% ( $\pm 0.54$ )	82.82% ( $\pm 0.94$ )
ISSCAN (SAL)	73.07% ( $\pm 2.07$ )	73.64% ( $\pm 1.76$ )
PTC (FM)	61.41% ( $\pm 2.69$ )	61.65% ( $\pm 2.91$ )
PTC (FR)	67.76% ( $\pm 2.02$ )	68.49% ( $\pm 1.70$ )
PTC (MM)	65.29% ( $\pm 3.57$ )	66.84% ( $\pm 2.99$ )
PTC (MR)	56.18% ( $\pm 2.46$ )	57.91% ( $\pm 3.82$ )

**Tabella 4.2.** Test accuracy per ogni problema ottenute dalla GraphESN-SOM allenando la SOM con i due diversi metodi di training.

Come si vede dalla tabella, il training supervisionato della SOM porta ad un leggero miglioramento della test accuracy in ognuno dei 6 problemi. Seppure il miglioramento non è particolarmente significativo, il fatto che sia diffuso e sistematico in ogni problema, evidenzia che non è un miglioramento casuale e che la GraphESN-SOM sfrutta il training supervisionato della SOM per guadagnarne in termini di generalizzazione. Inoltre, un buon miglioramento, di quasi un punto percentuale, è ottenuto nel dataset Bursi che è il primo dataset a cui facciamo riferimento per il numero maggiore di molecole di training e la minore variabilità dei risultati.

#### 4.2.2 Conclusioni

Nella sezione 3.1.1 è stata introdotta una procedura per il training supervisionato della SOM, sperimentata nella SMF della GraphESN-SOM e utilizzata anche negli esperimenti descritti di seguito. Nelle prove fatte si vede come la procedura per il training supervisionato riesce a separare gruppi di vertici in celle diverse della SOM in un modo che è guidato anche dal valore target (oltre che dalla distanza tra valori degli stati). Infatti, le unità della SOM si dispongono, quando possibile, in modo da raccogliere vertici con target prevalentemente positivo o negativo (e che portano ad uno stato simile). Il

training non supervisionato, invece, non riesce a fare altrettanto e tende a raggruppare in un'unica cella gli stati che invece il training supervisionato riesce a separare in celle differenti grazie al valore target. Le unità si dispongono in modo da raccogliere gruppi di vertici con stati simili senza riguardo per il valore target, anche in quei casi in cui leggere differenze, tra stati comunque simili, determinano un target differente.

La procedura di training supervisionato della SOM può quindi portare ad una ripartizione dello spazio degli stati significativa rispetto al problema in esame. In questo modo, la mappatura dei vertici di un grafo sulle unità della SOM, che rappresenta la scomposizione del grafo da cui si ottiene una rappresentazione a dimensione fissata, viene fatta in funzione del problema. Inoltre, l'assegnamento dei pesi nel readout, in corrispondenza di ogni unità della SOM, può trarre vantaggio dalla separazione dei vertici ottenuta dal training supervisionato, assegnando un peso che rispecchia la distribuzione del target nella cella, cioè di segno negativo se la cella raccoglie maggiormente vertici con target negativo, positivo se la cella raccoglie perlopiù vertici con target positivo. Il training supervisionato infatti porta la GraphESN-SOM ad ottenere un generale miglioramento delle performance, in confronto al training non supervisionato, nei 6 problemi di tossicologia.

Ulteriori approfondimenti potrebbero riguardare la procedura di training supervisionato, provando alcune tecniche alternative, più o meno sofisticate rispetto a quella utilizzata qui, basata su LVQ1, come quelle descritte in [76] o in [77].

### 4.3 Valori Binari e Valori Continui

In questa sezione vengono confrontate la funzione di binarizzazione (dell'equazione 3.5) e la funzione di somma (dell'equazione 3.7) come funzioni di aggregazione nella SMF. Il confronto viene fatto in termini di prestazioni nell'utilizzo in GraphESN-SOM nei 6 problemi di tossicologia. Per gli scopi di questa tesi la funzione di binarizzazione porta a diversi vantaggi discussi nella sezione 3.1.3. L'obiettivo è verificare l'eventuale diminuzione delle prestazioni rispetto all'utilizzo di una funzione di aggregazione che restituisce valori continui (come appunto la funzione che somma gli elementi) da cui il readout, potenzialmente, potrebbe trarre maggiori benefici rispetto ai più semplificativi valori binari. Un ulteriore confronto è fatto rispetto ad una SMF che invece non utilizza una funzione di aggregazione, restituendo la concatenazione delle medie degli stati del reservoir locali ad ogni cluster, in modo analogo a quanto fatto in GraphESN-NG (descritto nella sezione 2.1.8). Dal momento che la funzione di aggregazione stessa è stata introdotta per sfruttarne diversi vantaggi, quali la possibilità di creare delle mappe con un numero maggiore di unità oppure associare un solo semplice peso ad ogni

unità della SOM, si vuole verificare un’eventuale perdita di performance rispetto ad una SMF che non utilizza una funzione di aggregazione.

#### 4.3.1 Risultati

Nella Tabella 4.3 sono riportati i risultati di test accuracy nei 6 problemi di tossicologia, utilizzando una GraphESN-SOM con SMF che utilizza la funzione di binarizzazione e la funzione di somma. Nel primo caso sono utilizzate le configurazioni di base descritte nella sezione 4.1. Nel secondo caso sono riportati i risultati ottenuti con la stessa procedura e le stesse configurazioni a parte l’uso della funzione di somma come funzione di aggregazione e dei parametri  $\lambda_1$  e  $\lambda_2$  scelti dalla procedura di model selection nell’insieme  $\{1 \times 10^{-3}, 5 \times 10^{-3}, 1 \times 10^{-2}, 5 \times 10^{-2}, 1 \times 10^{-1}, 5 \times 10^{-1}\}$ .

Dataset	Valori binari	Somma
Bursi	82.82% ( $\pm 0.94$ )	81.41% ( $\pm 0.88$ )
ISSCAN (SAL)	73.64% ( $\pm 1.76$ )	72.72% ( $\pm 1.96$ )
PTC (FM)	61.65% ( $\pm 2.91$ )	60.32% ( $\pm 2.35$ )
PTC (FR)	68.49% ( $\pm 1.70$ )	64.45% ( $\pm 1.10$ )
PTC (MM)	66.84% ( $\pm 2.99$ )	64.51% ( $\pm 3.61$ )
PTC (MR)	57.91% ( $\pm 3.82$ )	56.62% ( $\pm 4.27$ )

**Tabella 4.3.** Accuracy di test media e deviazione standard di GraphESN-SOM con funzione di aggregazione a valori binari e con funzione di aggregazione con somma delle componenti.

Nei risultati riportati in Tabella 4.3 si vede come le prestazioni con valori binari risultino migliori su ognuno dei 6 problemi rispetto ad utilizzare la somma. Nei due problemi PTC (FR) e PTC(MM) il risultato è superiore anche di due punti percentuali. Significativo inoltre è l’aumento di più di un punto percentuale sul dataset Bursi.

Nella Tabella 4.4 sono riportati i risultati di test accuracy utilizzando una SMF senza funzione di aggregazione confrontati con i risultati ottenuti con la SMF a valori binari (gli stessi della tabella precedente, riportati nuovamente per migliorare la lettura). Per la SMF senza funzione di aggregazione sono stati scelti, tramite la procedura di model selection, il parametro sigma nello stesso insieme di valori delle prove precedenti, mentre i parametri  $\lambda_1$  e  $\lambda_2$  nell’insieme  $\{5 \times 10^{-5}, 1 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}, 5 \times 10^{-3}, 1 \times 10^{-2}\}$ . Nelle prove effettuate in questo caso sono state utilizzate mappe di dimensioni  $10 \times 6$  ( $K = 60$  unità). La scelta di mappe di dimensioni minori rispetto al caso in cui venga utilizzata una funzione di aggregazione è dovuta a motivi di costi computazionali. In questo caso infatti il vettore  $\mathcal{X}(\mathbf{g}_r)$  restituito dalla SMF ha dimensione  $KN_R$  poiché consiste nella concatenazione delle medie degli stati del reservoir locali ad ogni cluster. Il costo del training

del readout diviene  $\mathcal{O}((KN_R)^3 + T(KN_R)^2)$  contro il costo  $\mathcal{O}(K^3 + TK^2)$  nel caso si utilizzi una funzione di aggregazione (si può vedere la sezione 3.4 per i dettagli). Siccome  $N_R$  compare in un termine cubico, i tempi di training sono notevolmente maggiori nel caso non si utilizzi una funzione di aggregazione. Aumentano notevolmente anche le dimensioni delle strutture dati (soprattutto matrici) utilizzate per il training del readout. Ad esempio la matrice  $X$  ha dimensione  $KN_R \times T$ , dove  $T$  è il numero di elementi nel training set. Il valore di  $N_R$  (il numero di unità del reservoir) è stato preso uguale a 30, che lo si può considerare come un valore minimo in sistemi di Reservoir Computing (tipicamente composti da reservoir con centinaia di unità), che quindi è preferibile non diminuire ulteriormente. Tutto questo obbliga a prendere un valore di  $K$  minore rispetto al caso venga utilizzata una funzione di aggregazione, ovvero mappe di dimensioni minori. Per fare un esempio pratico, sul calcolatore utilizzato per questi esperimenti, con CPU Intel Celeron da 2.13 GHz e 3GB di memoria RAM DDR2, nel problema ISSCAN (SAL), il training del readout via Elastic Net (con  $\lambda_{1,2} = 1 \times 10^{-4}$ ), senza funzione di aggregazione, impiega circa 60 secondi con una mappa  $10 \times 6$  ( $K = 60$ ), circa 10 minuti con una mappa  $17 \times 10$  ( $K = 170$ ) mentre con una mappa  $27 \times 16$  ( $K = 432$ ) sono richiesti più di 6 GB di memoria (troppi per le risorse computazionali a disposizione). Gli stessi tempi con l'utilizzo della funzione di aggregazione a valori binari sono di circa 0.3 secondi con mappa  $10 \times 6$ , circa 0.5 secondi con una mappa  $17 \times 10$  e circa 2 secondi con una mappa  $27 \times 16$ . Se il training di un singolo modello impiega 10 minuti, la procedura di validazione utilizzata sul dataset ISSCAN per ottenere i risultati di test, con model selection su un insieme di 60 iperparametri (ottenuti dalla combinazione dei 10 valori di  $\sigma$  e 6 valori di  $\lambda_{1,2}$ ), descritta nella sezione 4.1), richiederebbe più di 50 giorni per essere completata. Per questo motivo si è obbligati ad utilizzare mappe di dimensioni minori per le prove senza funzione di aggregazione.

Dataset	Valori binari	Senza funzione di aggregazione
Bursi	82.82% ( $\pm 0.94$ )	82.63% ( $\pm 0.48$ )
ISSCAN (SAL)	73.64% ( $\pm 1.76$ )	72.23% ( $\pm 1.98$ )
PTC (FM)	61.65% ( $\pm 2.91$ )	60.05% ( $\pm 2.05$ )
PTC (FR)	68.49% ( $\pm 1.70$ )	65.46% ( $\pm 1.73$ )
PTC (MM)	66.84% ( $\pm 2.99$ )	63.80% ( $\pm 3.14$ )
PTC (MR)	57.91% ( $\pm 3.82$ )	57.90% ( $\pm 2.24$ )

**Tabella 4.4.** Accuracy di test media e deviazione standard di GraphESN-SOM con funzione di aggregazione a valori binari e senza funzione di aggregazione (in cui la SMF restituisce la concatenazione delle medie degli stati del reservoir locali ad ogni cluster).

Dalla Tabella 4.4 si vede come anche in questo caso la funzione di aggregazione a valori binari ottiene risultati migliori in modo sistematico su ogni dataset. In particolare, su PTC (FM) e su PTC (MM) le performance risultano migliori di circa 3 punti percentuali. All’incirca le stesse performance sono ottenute invece sul dataset Bursi. È comunque probabile che la versione senza funzione di aggregazione sia svantaggiata dall’utilizzo di mappe di piccole dimensioni, e con mappe di dimensioni maggiori possa ottenere risultati almeno uguali alla versione con valori binari.

#### 4.3.2 Conclusioni

L’uso della funzione di binarizzazione come funzione di aggregazione nella SMF non comporta quindi un calo di prestazioni rispetto all’utilizzo di valori continui ma anzi, in contrasto alle previsioni iniziali, nei 6 problemi di tossicologia su cui sono state effettuate le sperimentazioni, portano ad un guadagno in termini di generalizzazione. Non si ha una perdita di performance neppure rispetto ad una SMF che restituisce la concatenazione delle medie degli stati del reservoir locali ad ogni cluster (i.e. senza funzione di aggregazione), anche perché, in questo caso, si è costretti ad utilizzare mappe di dimensioni minori per motivi di costi computazionali. Questi risultati incoraggiano e motivano ulteriormente l’utilizzo della funzione di aggregazione a valori binari, che già porta a vantaggi in termini di interpretazione della risposta data dal sistema, come discusso nella sezione 3.1.3.

Potendo disporre di maggiori risorse computazionali si potrebbe approfondire il comportamento di un sistema senza funzione di aggregazione, con il quale verrebbe mantenuta tutta l’informazione restituita dal reservoir, introducendo un algoritmo di pruning a gruppi per selezionare le unità della SOM (poiché in questo caso ad ognuna di esse sarebbero associati gruppi di pesi nel readout e non un unico peso). Si deve notare però che dal punto di vista dell’analisi della risposta si avrebbero comunque svantaggi. Si perderebbe la possibilità di associare un peso direttamente interpretabile ad ogni unità della SOM, quindi in corrispondenza di ogni vertice di un grafo. Inoltre l’equazione generale del modello non sarebbe altrettanto chiara ed esplicita come quella ottenuta utilizzando una funzione di aggregazione, in particolare a valori binari.

### 4.4 Pruning e Regolarizzazione

In questa sezione vengono approfonditi gli effetti del pruning delle connessioni del readout nella GraphESN-SOM, facendo un confronto con una regolarizzazione del training che si limita a restringere il valore dei pesi senza annullarli. L’obiettivo è mostrare che il pruning porta ad una semplificazione del sistema riuscendo comunque a mantenere buone prestazioni. I vantaggi del pruning sono discussi nella sezione 3.2. Per fare questo viene fatto un

confronto delle performance predittive e della percentuale di unità annullate, tra il training fatto con Elastic Net (EN), utilizzato come metodo di pruning, e con Ridge Regression (RR), che è un metodo di regolarizzazione che porta ad un restringimento dei pesi, entrambi descritti nella sezione 2.1.9.

#### 4.4.1 Risultati

Ricordiamo innanzitutto che ad ogni unità della SOM è associato un peso del readout e che parliamo di unità *annullate* se il peso corrispondente è annullato dalla procedura di training (i.e. gli viene assegnato valore nullo). In Tabella 4.5 sono riportati i risultati ottenuti con GraphESN-SOM con le configurazioni di base descritte nella sezione 4.1, nei 6 problemi di tossicologia oggetto di queste sperimentazioni, utilizzando EN per il training del readout. In Tabella 4.6 sono invece riportati i risultati ottenuti utilizzando RR per il training del readout. Nella seconda colonna è riportata l'accuracy di test media e la deviazione standard, nella colonna *unità totali* è riportato il numero di unità della SOM, nella colonna *unità annullate* è riportata la percentuale di unità annullate dal training, nella colonna *unità effettive* è riportato il numero di unità con un peso associato diverso da zero.

<b>Dataset</b>	<b>Test accuracy</b>	<b>Unità totali</b>	<b>Unità annullate</b>	<b>Unità effettive</b>
Bursi	82.82% ( $\pm 0.94$ )	1500	68.41% ( $\pm 0.91$ )	474
ISSCAN (SAL)	73.64% ( $\pm 1.76$ )	432	81.49% ( $\pm 0.84$ )	80
PTC (FM)	61.65% ( $\pm 2.91$ )	680	92.61% ( $\pm 0.54$ )	50
PTC (FR)	68.49% ( $\pm 1.70$ )	680	94.58% ( $\pm 0.57$ )	37
PTC (MM)	66.84% ( $\pm 2.99$ )	680	89.14% ( $\pm 0.79$ )	74
PTC (MR)	57.91% ( $\pm 3.82$ )	432	87.17% ( $\pm 0.85$ )	55

**Tabella 4.5.** GraphESN-SOM con readout allenato via Elastic Net.

<b>Dataset</b>	<b>Test Acc.</b>	<b>Unità totali</b>	<b>Unità annullate</b>	<b>Unità effettive</b>
Bursi	82.60% ( $\pm 0.70$ )	1500	36.87% ( $\pm 2.07$ )	947
ISSCAN (SAL)	73.38% ( $\pm 1.91$ )	432	50.33% ( $\pm 1.81$ )	215
PTC (FM)	62.09% ( $\pm 2.25$ )	680	57.91% ( $\pm 3.59$ )	286
PTC (FR)	65.87% ( $\pm 1.88$ )	680	54.47% ( $\pm 5.76$ )	310
PTC (MM)	66.95% ( $\pm 3.12$ )	680	58.15% ( $\pm 3.32$ )	285
PTC (MR)	57.37% ( $\pm 1.90$ )	432	45.85% ( $\pm 5.07$ )	234

**Tabella 4.6.** GraphESN-SOM con readout allenato via Ridge Regression.

Da un confronto sulle due tabelle, si vede innanzitutto che il training via EN (in Tabella 4.5) ottiene migliori performance in termini di test accuracy rispetto al training con RR (in Tabella 4.6) in 5 problemi su 6, anche se la differenza è in ogni caso minima. Solamente in PTC (FR) infatti la differenza dell'accuracy di test tra i due tipi di training supera il punto percentuale, in favore del training con EN. In generale però si può sicuramente affermare che EN non porta ad un calo delle prestazioni predittive rispetto a RR. Inoltre, molto significativo è il numero di unità effettive, con associato un peso non nullo. Da notare che anche con RR si hanno buone percentuali di unità annullate, che vanno dal 36% in Bursi fino al 58% in PTC (MM). Questo, però, non è dovuto ad effetti di pruning del metodo RR ma al fatto che, per come funziona la SOM (a riguardo si possono vedere le sezioni 2.1.10 e 3.1.2), ci sono unità che non raccolgono alcuno stato del reservoir in fase di training e, a tali unità, è naturale che il training del readout gli assegna un peso nullo (indipendentemente dalla tecnica utilizzata). Utilizzando EN il numero di unità effettive è molto minore rispetto a RR, su ognuno dei problemi. Per esempio, in Bursi, il training con RR porta a 947 unità effettive, che corrisponde al 36% di unità annullate, mentre con EN si scende a quasi la metà rispetto a RR, con 474 unità effettive, che corrisponde a circa il 68% di unità annullate. Molto più accentuato questo effetto in PTC (FR), dove le unità effettive con EN sono addirittura un decimo rispetto a RR. È quindi evidente che con EN si ottengono quindi sistemi molto più semplici, rispetto a RR, con un numero molto maggiore di unità annullate, pur mantenendo ottime prestazioni di generalizzazione.

#### 4.4.2 Conclusioni

Il training del readout via EN permette di annullare molte unità della SOM, lasciando solamente, nelle prove effettuate, dalla metà fino ad un decimo di unità rispetto ad un training con RR. Le prestazioni in termini di generalizzazione non risentono degli effetti del pruning, ma anzi, in alcuni casi, portano a migliori risultati. La bontà, sempre in termini di capacità di generalizzazione, di tecniche di training che combinano metodi di pruning e shrinking, come viene fatto da EN, nell'ambito del Reservoir Computing, trova riscontri anche in letteratura, ad esempio in [64] dove una combinazione tra Backward Deletion e RR porta ad un miglioramento delle prestazioni rispetto ad utilizzare solamente RR.

Prove con un tuning migliore dei parametri  $\lambda_1$  e  $\lambda_2$  del metodo EN, qui scelti con valori uguali l'uno all'altro, potrebbero inoltre portare a migliori risultati, sia in termini di pruning (di numero di unità annullate) che di generalizzazione. Questo però a scapito di un maggiore costo del processo di model selection dal momento che si avrebbero un maggior numero di iperparametrizzazioni (dovendo provare diverse combinazioni tra i due parametri).

## 4.5 Confronto con Metodo a Frammenti

In questa sezione vengono confrontati GraphESN-SOM e il Metodo a Frammenti (MF) introdotto nella sezione 3.5. Il confronto, interessante per l'analogia del funzionamento dei due metodi, viene fatto in termini di performance di generalizzazione nei 6 problemi di tossicologia e di semplicità nei modelli in termini del numero di parametri liberi da cui dipende la risposta.

### 4.5.1 Risultati

Nelle prove effettuate MF è stato implementato utilizzando una codifica in SMILES canonico [92] dei frammenti derivanti dalle molecole di input. Ogni frammento è quindi rappresentato con una stringa univoca indipendente dall'ordine degli atomi nella sua struttura. Questo ha permesso di implementare in modo efficiente la ricerca di frammenti nell'insieme  $\mathcal{F}$  (si veda la sezione 3.5), fondamentale nella costruzione dello stesso insieme  $\mathcal{F}$  e nella costruzione del vettore  $\mathcal{F}(\mathbf{g})$  (i.e. il vettore con cui viene rappresentata la scomposizione in frammenti di una molecola). La codifica SMILES permette di distinguere atomi aromatici (codificati con lettere minuscole) rispetto ad atomi alifatici e di codificare la carica di un atomo (con un simbolo + o - a seconda del tipo di carica). Inoltre viene inserito il tipo di legami tra atomi in modo esplicito se questi sono diversi da legami singoli (in particolare, il simbolo = rappresenta un legame doppio tra due atomi e il simbolo # un legame triplo). Ad esempio, con la stringa N+O- si denota un atomo di azoto (N) con carica positiva legato ad un atomo di ossigeno (O) con carica negativa. Con la stringa N=C=O si denota un atomo di carbonio (C) legato ad un atomo di ossigeno (O) e ad un atomo di azoto (N) con doppi legami. Con la stringa c1ccccc1 si denota un ciclo di carboni (C) aromatici (il numero 1 indica che i due atomi di carbonio che precedono tale numero sono legati assieme a chiudere il ciclo). Altre informazioni, per esempio riguardo l'isomeria, possono essere inserite nella rappresentazione SMILES, ma nell'uso in queste prove sono utilizzate solo le tre indicate (i.e. carica, aromaticità e tipo di legami). Però, mentre la carica associata ad un atomo e il tipo di legami sono sempre inseriti, un atomo aromatico viene codificato solamente se il frammento include l'intero ciclo aromatico. Per cui può succedere che alcuni frammenti codificati con SMILES perdano l'informazione riguardante l'aromaticità di alcuni suoi vertici. Tutto questo fa sì che le informazioni utilizzate da MF si debbano considerare differenti da quelle utilizzate in GraphESN-SOM. In quest'ultimo caso infatti, come specificato nella sezione 4.1, non vengono distinti tipi diversi di legami tra atomi, mentre invece la codifica dell'aromaticità (oltre che della carica) è sempre presente nella rappresentazione dell'atomo come vertice di un grafo.

In Tabella 4.7 sono riportati i risultati ottenuti con MF nei 6 problemi di tossicologia. Tali risultati sono ottenuti con la stessa procedura di validazione

utilizzata per GraphESN-SOM (descritta nella sezione 4.1). Il training è fatto con Elastic Net e i parametri  $\lambda_1$  e  $\lambda_2$  sono scelti con la procedura di model selection nell’insieme di valori  $\{0.5, 1, 2, 4, 8, 16\}$  con valori sempre uguali l’uno all’altro. Il parametro  $R$  (si veda la sezione 3.5) è scelto con la procedura di model selection nell’insieme di valori  $\{0, 1, 2\}$ . Valori maggiori di 2 per il parametro  $R$  non si sono potuti prendere poiché portano ad un numero eccessivo di frammenti (nell’insieme  $\mathcal{F}$ ) che rende impossibile il training per mancanza di risorse computazionali. Ad esempio, nel dataset Bursi, con  $R = 3$ , si ottengono più di 17 000 frammenti (unici), che è la dimensione del vettore con il quale viene rappresentata ogni molecola. Con questa dimensione sono richiesti più di 6 GB di memoria per il training con Elastic Net. Nella colonna *frammenti totali* sono riportati il numero di frammenti ottenuti da MF sul training set (i.e. la cardinalità dell’insieme  $\mathcal{F}$ ), mediati sulle 5 folds della cross-validation nel caso dei dataset PTC e ISSCAN. Nella colonna *frammenti eliminati* c’è la percentuale di frammenti a cui è stato assegnato un peso nullo dal processo di training. Nella colonna *frammenti effettivi* c’è il numero di frammenti con peso associato diverso da zero.

<b>Dataset</b>	<b>Test accuracy</b>	<b>Framm. totali</b>	<b>Framm. eliminati</b>	<b>Framm. effettivi</b>
Bursi	81.96%	4344	86.40%	591
ISSCAN (SAL)	73.49%	1780	85.00%	267
PTC (FM)	63.31%	1071	99.38%	7
PTC (FR)	67.51%	1092	96.24%	41
PTC (MM)	62.18%	1041	85.14%	155
PTC (MR)	62.17%	1048	96.70%	35

**Tabella 4.7.** Risultati ottenuti con il Metodo a Frammenti nei 6 problemi di tossicologia.

Nella Tabella 4.5 di pagina 86 sono riportati i risultati analoghi per GraphESN-SOM, con funzione di aggregazione a valori binari e training via Elastic Net. Dal confronto dei risultati riportati nelle due tabelle si vede come in 4 problemi su 6 GraphESN-SOM ottiene migliori performance di generalizzazione rispetto a MF. Significativo il fatto che GraphESN-SOM è migliore di quasi un punto percentuale sul dataset Bursi, composto da una maggiore quantità e qualità dei dati che permettono di avere risultati poco variabili (quindi più affidabili). Inoltre con GraphESN-SOM si ottengono quasi sempre modelli più semplici. Guardando infatti al numero di unità effettive per GraphESN-SOM e al numero di frammenti effettivi in MF, che per entrambi i metodi sono il numero di parametri da cui dipende la risposta finale (quindi indici della semplicità del modello), GraphESN-SOM porta a modelli più semplici in 4 casi su 6. Da notare che un modello porta a

migliori risultati di generalizzazione rispetto all’altro nei casi in cui risulta dipendere da un minor numero di parametri, a conferma del fatto che una maggiore semplicità del modello porta a migliori performance predittive. Infine, interessante è anche il confronto tra unità totali in GraphESN-SOM e frammenti totali in MF, che sono in generale molto minori nel primo caso. Questo ha forti ripercussioni sull’efficienza del training del readout siccome, in entrambi i casi, costituisce la dimensione del vettore che rappresenta una molecola di input, su cui viene appunto eseguito il training.

#### 4.5.2 Conclusioni

Nella sezione 3.5 si sono discussi i vantaggi di GraphESN-SOM rispetto ad un approccio come MF. In questa sezione si è visto che GraphESN-SOM può portare anche a migliori prestazioni di generalizzazione nei problemi di tossicologia su cui sono stati sperimentati entrambi i metodi. Inoltre, nonostante un funzionamento più complesso rispetto a MF, GraphESN-SOM risulta portare a modelli più semplici per quanto riguarda il numero di parametri da cui dipende la predizione. Quest’ultimo aspetto è un vantaggio sia per l’analisi della risposta (per gli stessi motivi per cui vengono utilizzate tecniche di pruning), sia, potenzialmente, in termini di performance di generalizzazione per il principio di parsimonia per il quale si dovrebbe sempre utilizzare un modello più semplice per prevenire overfitting e perdita di generalizzazione.

### 4.6 Performance su Datasets di Tossicologia

In questa sezione sono confrontate le performance predittive di GraphESN-SOM, nei 6 problemi di tossicologia, con le performance ottenute da approcci simili di Reservoir Computing per grafi (GraphESN e GraphESN-NG) e con risultati allo stato dell’arte in ogni problema. Lo scopo è verificare le capacità di generalizzazione di GraphESN-SOM guardando come si pone rispetto a risultati ottenuti da modelli simili e a risultati allo stato dell’arte, seppure è da sottolineare che le performance non sono il principale obiettivo di questo modello.

#### 4.6.1 Risultati

I risultati riportati per GraphESN-SOM sono ottenuti con la procedura e la configurazione di base descritte nella sezione 4.1, utilizzando la funzione di aggregazione a valori binari. I risultati per GraphESN e GraphESN-NG sono presi da [21] in cui il processo di model selection è lo stesso utilizzato per GraphESN-SOM.

In Tabella 4.8 sono riportati i risultati nel problema Bursi. La Support Vector Machine (SVM) fa riferimento al metodo utilizzato in [93]. Si può

<b>Metodo</b>	<b>Test Accuracy</b>
GraphESN-SOM	82.82% ( $\pm 0.94$ )
GraphESN	75.82% ( $\pm 0.55$ )
GraphESN-NG	79.24% ( $\pm 0.64$ )
SVM (Ferrari)	83.20%

**Tabella 4.8.** Confronto sul dataset Bursi.

vedere come le performance ottenute con GraphESN-SOM sono superiori di 7 punti percentuali rispetto a GraphESN e di oltre 3 punti rispetto a GraphESN-NG, ottenendo quindi risultati molto migliori rispetto ai due metodi simili. Inoltre, le performance di GraphESN-SOM sono molto vicine ai risultati allo stato dell'arte ottenute dalla SVM utilizzata in [93]. Si deve sottolineare che in quest'ultimo metodo si basa su un insieme di 27 descrittori ricavati da un processo preliminare di features selection che utilizza diversi altri metodi per estrarre e selezionare i descrittori migliori per lo specifico task. In GraphESN-SOM, invece, viene data direttamente la struttura del composto con solamente informazioni riguardo il tipo di atomi e l'eventuale aromaticità e carica. Oltre il fatto che i costi computazionali dei metodi a kernel (in particolare delle Support Vector Machines) sono tipicamente superiori a i costi computazionali di GraphESN-SOM (come discusso nella sezione 3.4). Infine, l'82.82% ottenuto è molto vicino alla percentuale dell'85% di attendibilità del *test di Ames* da laboratorio per mutagenicità con il quale sono classificate le molecole nel dataset Bursi [87]. L'85% rappresenta quindi una soglia limite che non si dovrebbe superare poiché, in tal caso, vorrebbe dire che il modello ha appreso anche gli errori sperimentali.

<b>Metodo</b>	<b>Test Accuracy</b>
GraphESN-SOM	73.64% ( $\pm 1.76$ )
SAs Benigni/Bossa	78.00%

**Tabella 4.9.** Confronto sul dataset ISSCAN.

In Tabella 4.9 sono riportate le performance ottenute per ISSCAN (SAL), confrontate con i risultati riportati in [83] ottenuti classificando i composti a partire da un insieme di Structural Alerts (SAs) per mutagenicità. Come spiegato nella sezione 2.2.2 le SAs sono il frutto di conoscenze specifiche che derivano da varie fonti, come studi sperimentali, teorici o statistici. Per cui le informazioni contenute nella lista di SAs sono ben più di quelle ottenibili guardando al solo dataset, come invece viene fatto utilizzando un modello come GraphESN-SOM. Il 73.6% ottenuto con GraphESN-SOM è per cui da considerarsi un buon risultato poiché si avvicina al 78% ottenuto dalla lista di SAs che rappresenta una conoscenza derivata e raffinata negli anni per riconoscere potenziali attività mutageniche nei composti chimici.

<b>Metodo</b>	<b>FM</b>	<b>FR</b>	<b>MM</b>	<b>MR</b>
GraphESN-SOM	61.7 ( $\pm 2.9$ )	68.5 ( $\pm 1.7$ )	66.8 ( $\pm 3.0$ )	57.9 ( $\pm 3.8$ )
GraphESN	60.4 ( $\pm 0.9$ )	67.1 ( $\pm 0.1$ )	65.0 ( $\pm 0.7$ )	57.4 ( $\pm 3.3$ )
GraphESN-NG	62.5 ( $\pm 1.7$ )	66.7 ( $\pm 1.7$ )	64.8 ( $\pm 1.4$ )	61.2 ( $\pm 2.2$ )
Graph Kernels	64.5	66.9	66.4	65.7

**Tabella 4.10.** Test accuracy media e deviazione standard per i quattro problemi PTC.

In Tabella 4.10 sono riportati i risultati ottenuti nei quattro problemi dal dataset PTC, confrontati con GraphESN, GraphESN-NG e con risultati ottenuti in [42] utilizzando diversi kernel per grafi. In ognuno dei 4 problemi GraphESN-SOM ottiene migliori risultati rispetto a GraphESN, mentre in 2 problemi su 4 (i.e. FR e MM) ottiene migliori risultati rispetto a GraphESN-NG. Nei confronti dei kernel per grafi, a parte in MR dove la differenza è molto significativa in sfavore di GraphESN-SOM, in FM e MM i risultati sono analoghi e in FR vengono ottenuti migliori risultati. In Tabella 4.11

<b>Metodo</b>	<b>FM</b>	<b>FR</b>	<b>MM</b>	<b>MR</b>
MG-Kernel	64.7 ( $\pm 1.2$ )	70.1 ( $\pm 0.6$ )	69.1 ( $\pm 1.5$ )	62.5 ( $\pm 1.2$ )
OA-Kernel	65.3 ( $\pm 0.9$ )	70.4 ( $\pm 1.1$ )	67.8 ( $\pm 1.7$ )	63.4 ( $\pm 2.0$ )

**Tabella 4.11.** Test accuracy media e deviazione standard ottenuti da metodi basati su kernel nei quattro problemi PTC.

sono riportati altri risultati allo stato dell'arte ottenuti da metodi a kernel sui problemi PTC in [51]. Questi risultati non sono però pienamente confrontabili con quelli riportati in Tabella 4.10 poiché sono ottenuti con un differente processo di validazione. In questo caso sono stati presi i migliori risultati ottenuti sulla scelta dei parametri del kernel, eseguendo una model selection solo sui parametri della SVM. I risultati ottenuti con GraphESN-SOM sono comunque comparabili, in particolare nei casi FR ed MM in cui si hanno risultati molto vicini a quelli ottenuti con i metodi a kernel. Tenendo conto anche della rumorosità dei dati del dataset PTC, i risultati ottenuti con GraphESN-SOM su questo dataset (a parte nel problema MR) si possono quindi considerare sullo stesso piano di risultati allo stato dell'arte, ottenuti con metodi a kernel computazionalmente più onerosi di GraphESN-SOM.

#### 4.6.2 Conclusioni

Le performance predittive di GraphESN-SOM sono quindi migliori, sulla maggior parte dei problemi sperimentati, di approcci simili basati su Reservoir Computing per grafi, GraphESN e GraphESN-NG in particolare. In più si raggiungono performance comparabili a risultati allo stato dell'arte.

In particolare nel dataset Bursi si raggiungono ottimi risultati, molto vicini ai migliori ottenuti su tale dataset e molto vicini al limite imposto dalla rumorosità del problema. Il fatto di riuscire ad ottenere buone prestazioni è molto importante anche ai fini dell'analisi della risposta, poiché, altrimenti, si avrebbe un modello che restituisce risposte poco attendibili e poco interessanti.

## 4.7 Analisi della Predizione

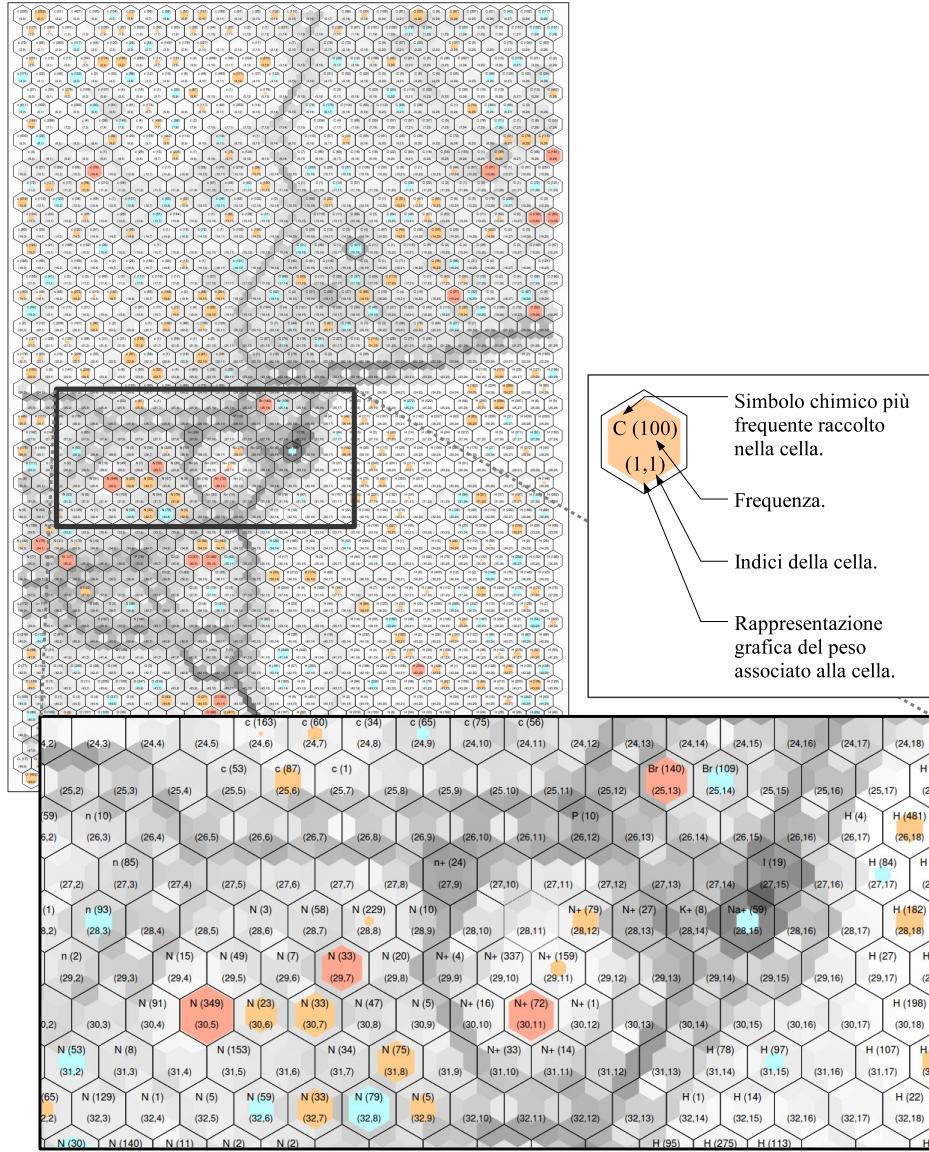
In questa sezione viene mostrato un esempio di funzionamento del sistema GraphESN-SOM, introdotto in questa tesi, nella classificazione di un composto chimico. Un modello è allenato sulla parte di training del dataset Bursi (si veda l'appendice A). Il problema è quindi di classificazione di un composto chimico come tossico (in particolare mutagено) oppure non tossico. Nel primo caso la risposta del modello è di segno positivo (valore target +1), nel secondo caso di segno negativo (valore target -1). Il modello di GraphESN-SOM è allenato utilizzando la configurazione di base descritta in 4.1, con State Mapping Function (SMF) a valori binari e training del readout via Elastic Net (EN). Il valore del parametro  $\sigma$  (i.e. il coefficiente di contrattività del reservoir) è preso di 1.8, mentre  $\lambda_1$  e  $\lambda_2$  (i.e. i parametri del metodo EN) sono presi entrambi di valore 4. La SOM utilizzata nella SMF è presa di dimensione  $50 \times 30$  unità ( $K = 1500$ ).

In Figura 4.6 è raffigurata la mappa ottenuta dalla SOM dopo il training del modello. Su ogni cella della mappa è riportato, in alto, il simbolo atomico più frequente associato ai vertici raccolti dalla cella (durante il processo di training), con a fianco la frequenza stessa, mentre in basso sono riportati gli indici identificativi della cella. Sempre in corrispondenza di ogni cella è riportata anche una rappresentazione grafica della dimensione del peso associato alla cella nel readout di GraphESN-SOM. In alto, sopra la mappa, è riportato il valore del bias (del readout). Dato che il bias ha segno negativo, i pesi su ogni cella sono colorati in questo modo:

- Azzurro: peso negativo.
- Arancione: peso positivo di dimensione inferiore al bias (in valore assoluto).
- Rosso: peso positivo di dimensione superiore al bias (in valore assoluto).

I pesi positivi hanno due colorazioni per visualizzarne meglio la dimensione, prendendo il bias come soglia poiché un peso arancione da solo non sarebbe sufficiente a far sì che una molecola sia classificata positiva, ma dovrebbe essere accompagnato da un altro peso positivo, mentre un peso rosso sarebbe invece sufficiente, da solo, a far cambiare segno alla classificazione. Sullo sfondo, infine, è presente la U-Matrix della SOM (si veda la sezione 2.1.10),

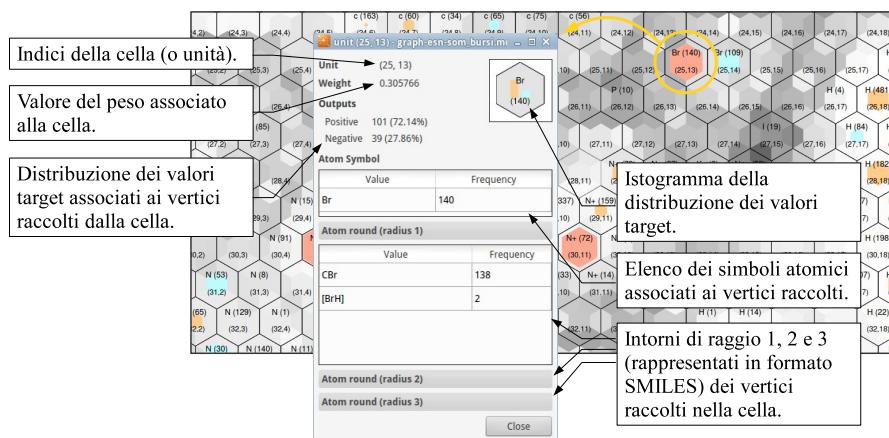
SOM 50x30 | Bias = -0.28



**Figura 4.6.** Mappa (di 50x30 unità) ottenuta dalla SOM dopo il training di GraphESN-SOM sul dataset Bursi (sulla parte di training). Nel dettaglio della mappa, in basso, si possono vedere le informazioni riportate in corrispondenza di ogni cella, descritte nella legenda sulla destra. Le celle in cui non è presente il simbolo chimico sono celle che non hanno raccolto nessun vertice (in fase di training). Le celle senza un peso associato (visibile in colore azzurro, arancione o rosso) corrispondono alle unità della SOM annullate dal pruning del readout. Ogni vertice di una molecola da classificare attiva una cella della mappa, e la classificazione della molecola è data dalla somma dei pesi delle celle attivate più il valore del bias (riportato sopra alla mappa).

grazie alla quale si possono distinguere varie aree della mappa, separate da colorazioni più scure, che raccolgono vertici simili. Per esempio, nel dettaglio della mappa riportato in basso, si può distinguere un'area che raccoglie i vertici di atomi di azoto (N), a partire da sinistra andando verso il centro, a sua volta separata in una sottoarea che raccoglie gli azotini con carica positiva ( $N^+$ ). Sulla destra è visibile una parte dell'area (molto estesa) che raccoglie i vertici di atomi di idrogeno (H). In alto sulla destra si può, infine, distinguere un'area (composta da sole due celle) che raccoglie i vertici associati ad atomi di bromo (Br).

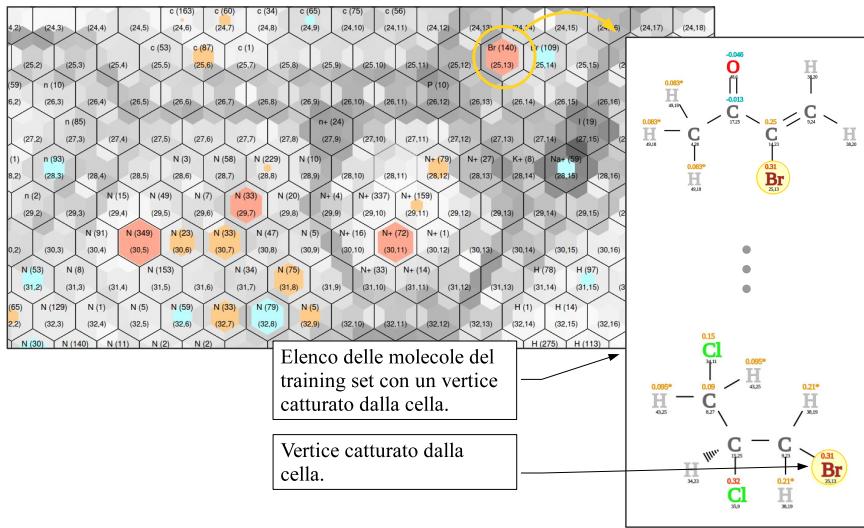
Per ogni cella si possono visualizzare un insieme di informazioni utili a capire il tipo di vertici raccolti, quindi a comprendere il funzionamento del modello nella classificazione di un grafo. In Figura 4.7 è mostrato un esempio in cui si visualizzano i dettagli relativi ad una cella che raccoglie vertici di atomi di bromo (Br). In particolare, è possibile vedere il valore esatto del peso associato alla cella, la distribuzione dei valori target associati ai vertici raccolti (con un relativo istogramma) e i frammenti (rappresentati in formato SMILES) relativi agli intorni di raggio 1,2 e 3 dei vertici raccolti. In Figura



**Figura 4.7.** Esempio di visualizzazione dei dettagli relativi ad una cella che raccoglie vertici di atomi di bromo (Br).

4.8 viene visualizzato un elenco di tutte le molecole del training set con un vertice catturato dalla cella in esame. Ogni informazione è ovviamente relativa ai dati di training su cui il modello è stato costruito, ma risultano utili a capire, ad esempio, il motivo per cui un vertice viene mappato in una determinata cella.

Ogni vertice di una molecola da classificare attiva una cella della mappa in base al tipo di atomo associato al vertice e all'intorno del vertice stesso (come descritto nei dettagli nel capitolo 3). Grazie all'utilizzo della SMF a valori binari, la classificazione della molecola è data dalla somma dei pesi associati alle celle attivate, più il valore del bias. Il funzionamento del modello può



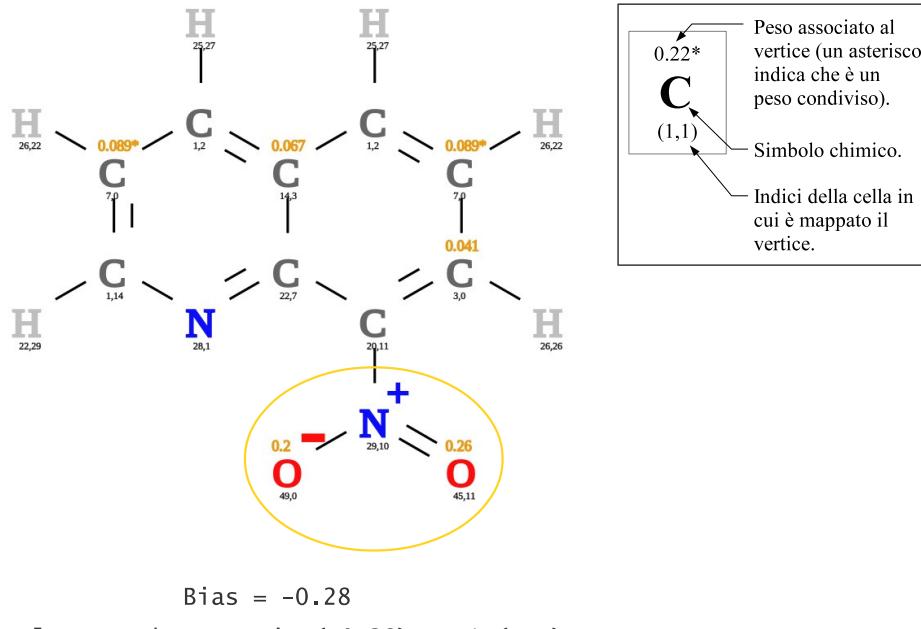
**Figura 4.8.** Per una cella che raccoglie vertici di atomi di bromo (Br) viene visualizzato un elenco di tutte le molecole del training set con un vertice catturato da tale cella. Nell'elenco è evidenziato (con uno sfondo giallo) il vertice catturato dalla cella.

essere quindi sintetizzato dalla seguente equazione:

$$\begin{aligned}
 \mathcal{C}(M) = & \text{sign}(-0.28 + 0.22 \cdot \text{Hit}(0,1) - 0.045 \cdot \text{Hit}(0,5) \\
 & + 0.081 \cdot \text{Hit}(0,6) - 0.12 \cdot \text{Hit}(0,8) + 0.095 \cdot \text{Hit}(0,9) \\
 & + 0.11 \cdot \text{Hit}(0,10) + 0.068 \cdot \text{Hit}(0,14) \dots \\
 & + 0.39 \cdot \text{Hit}(49,9) - 0.081 \cdot \text{Hit}(49,14) + 0.013 \cdot \text{Hit}(49,15) \\
 & + 0.082 \cdot \text{Hit}(49,18) + 0.056 \cdot \text{Hit}(49,29))
 \end{aligned}$$

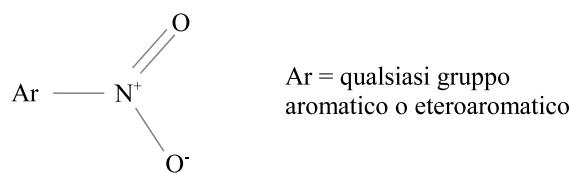
Dove  $\mathcal{C}(M)$  è la classificazione (+1 o -1) restituita dal modello per una molecola  $M$ , mentre la funzione  $\text{Hit}(m, n)$  ha valore 1 se la cella  $(m, n)$  viene attivata da almeno un vertice della molecola  $M$ , 0 altrimenti. Il coefficiente davanti alla funzione  $\text{Hit}(m, n)$  è il peso associato alla cella  $(m, n)$ .

In Figura 4.9 è riportata una molecola classificata dal modello. Sotto il simbolo di ogni atomo della molecola è riportato l'indice della cella che ha catturato il vertice corrispondente, mentre sopra il simbolo dell'atomo è riportato il peso associato a tale cella (se diverso da zero). Un asterisco accanto al peso indica che è un peso condiviso, cioè che all'interno della stessa molecola un altro vertice è finito nella stessa cella della SOM e questi vertici condividono quindi lo stesso peso. Nel calcolo della risposta i pesi condivisi (relativi ad una stessa cella) devono essere considerati una volta solamente. La somma dei pesi associati ad ogni atomo determina la classificazione della molecola. In altri termini, alla molecola è associata la seguente equazione



**Figura 4.9.** Esempio di una molecola classificata dal modello. Nella legenda in alto a destra sono descritte le informazioni riportate in corrispondenza di ogni vertice. Guardando i pesi associati ad ogni vertice è evidente che la sottostruttura evidenziata (con un cerchio) è determinante nella classificazione della molecola. Tale sottostruttura trova riscontro nella Structural Alert riportata in figura 4.10.

#### SA\_27: Nitro-aromatic



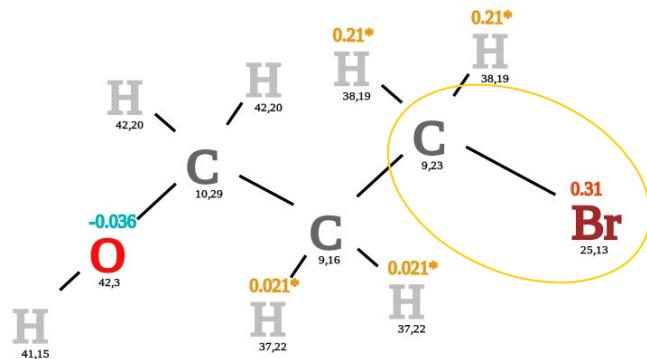
**Figura 4.10.** Structural Alert SA\_27: la presenza della struttura composta da un azoto con carica positiva ( $\text{N}^+$ ) legato a due ossigeni, uno dei quali con carica negativa ( $\text{O}^-$ ), legata ad un gruppo aromatico (Ar), è correlata ad un'attività mutagena del composto.

che ne determina la classificazione, ottenuta dalla equazione generale del modello mantenendo solo le Hit con valore 1:

$$\begin{aligned}
 \mathcal{C}(M) &= \text{sign}(-0.28 + 0.067 \cdot \text{Hit}(14, 3) + 0.041 \cdot \text{Hit}(3, 0) \\
 &\quad + 0.041 \cdot \text{Hit}(3, 0) + 0.089 \cdot \text{Hit}(7, 0) + 0.26 \cdot \text{Hit}(45, 11) \\
 &\quad + 0.20 \cdot \text{Hit}(49, 0)) \\
 &= \text{sign}(0.38) = +1
 \end{aligned}$$

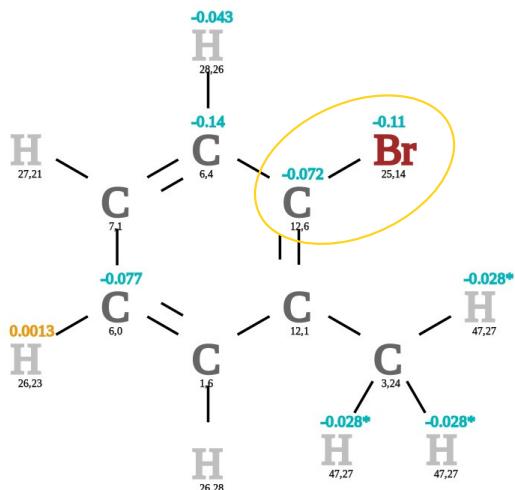
Nell'esempio della Figura 4.9 si può vedere che la sottostruttura composta da i due atomi di ossigeno (O) e l'atomo di azoto (N) che li collega (cerchiata in giallo nella figura) ha un peso complessivo che è fortemente positivo (di valore 0.46), che fa sì che la molecola sia classificata come tossica. La sottostruttura evidenziata trova conferma nella Structural Alert SA\_27 (si veda la sezione 2.2.2), riportata in figura 4.10, in cui viene indicato che la presenza di tale sottostruttura in un composto chimico, legata ad un gruppo aromatico (come infatti risulta anche nell'esempio), è correlata ad un'attività mutagenica della molecola. Molto importante è notare che il pruning permette di associare un numero limitato di pesi (non nulli) per ogni atomo della molecola, portando ad individuare facilmente le componenti strutturali determinanti per la risposta.

Un altro esempio di predizione di una molecola è riportato in Figura 4.11. Anche in questo caso si può notare una sottostruttura, composta da un atomo di carbonio (C) legato ad un atomo di bromo (Br) e a due atomi di idrogeno (H), con peso fortemente positivo che porta la molecola ad essere classificata come tossica. Di nuovo, questa sottostruttura trova conferma della sua correlazione con un'attività tossica nella SA\_8 riportata in figura 4.1. Con un cerchio giallo è evidenziata una sottostruttura carbonio-bromo che nella molecola in figura ha un peso (complessivo) positivo. La stessa sottostruttura si trova nella molecola in Figura 4.12, però, in questo caso, con un peso fortemente negativo che determina una classificazione negativa (non tossica) dell'intera molecola. Questo evidenzia la capacità di GraphESN-SOM di modificare un peso di un vertice (e più in generale di una sottostruttura) in base al contesto in cui si trova. Questa proprietà è aiutata ed amplificata dal training supervisionato della SOM che, come descritto meglio nella sezione 4.2, permette di separare vertici con contesti differenti in diverse unità della SOM se portano a target opposti. In questo modo il readout è in grado di assegnare un diverso peso alle unità che raccolgono vertici con contesti diversi. Si deve infine sottolineare che le tre molecole riportate come esempio di predizione sono prese dal test set di Bursi (esterne alla parte di training) e in tutti e tre i casi la predizione risulta corretta rispetto a quella annotata nel dataset.



Bias = -0.28  
Valore predetto = sign(+0.06) = +1 (TOX)

**Figura 4.11.** Esempio di una molecola classificata dal modello. Con un cerchio giallo è evidenziata una sottostruttura carbonio-bromo che in questo caso ha un peso positivo. La stessa sottostruttura, nella molecola in Figura 4.12, ha un peso negativo.



Bias = -0.28  
Valore predetto = sign(-0.75) = -1 (NON TOX)

**Figura 4.12.** La classificazione della molecola è negativa grazie anche alla sottostruttura bromo-carbonio che, in questo caso, assume un peso negativo.



# Capitolo 5

## Conclusioni

In questa tesi è stato introdotto un sistema di Reservoir Computing (RC) per grafi, chiamato GraphESN-SOM, in grado di fornire elementi utili all’analisi qualitativa della predizione, applicabile a problemi di regressione e classificazione. Per la realizzazione di questo sistema si è esteso il modello GraphESN con una nuova State Mapping Function (SMF), basata su Self-Organizing Map (SOM), e con un training del readout via Elastic Net. La SMF, innovativa sia nel contesto delle SMFs sia per i suoi aspetti supervisionati, permette di visualizzare, su una mappa bi-dimensionale ottenuta dalla SOM, una rappresentazione grafica di un modello allenato. La mappa facilita la comprensione del funzionamento del modello stesso supportando l’analisi qualitativa. In favore dell’interpretazione della risposta è stata inoltre introdotta una tecnica di discretizzazione degli stati associati alle celle della SOM. Allo stesso scopo, il training via Elastic Net consente di effettuare un pruning delle connessioni del readout, riducendo il numero di elementi che determinano la risposta restituita dal sistema. Tutto questo permette di visualizzare il modo in cui un modello ottiene una predizione direttamente sulla struttura da elaborare, associando un peso ad ogni vertice, in modo da evidenziare le componenti strutturali ritenute più importanti dal sistema. Inoltre si mantengono i vantaggi tipici del RC, con un approccio efficiente ed in grado di ottenere buone performance. Il sistema proposto si pone quindi come una valida scelta in ogni campo applicativo dove i dati sono rappresentati in strutture complesse (e.g. grafi), e si ha la necessità di analizzare e valutare qualitativamente la risposta. Si differenzia, invece, da metodi basati su descrittori, con i quali si devono scegliere a priori, ed in modo dipendente dal problema affrontato, un insieme di caratteristiche per la rappresentazione di un dato strutturato. La possibilità di trattare direttamente dati strutturati permette, infatti, a GraphESN-SOM di adattarsi in modo automatico al problema da affrontare.

Il sistema descritto trova un’importante applicazione in ambito tossicologico dove vi è un crescente interesse verso sistemi computazionali predittivi,

per l'analisi di composti chimici, per via di recenti regolamentazioni che ne incentivano l'utilizzo (come metodi alternativi a test su animali). In quest'ambito si devono poter elaborare grafi, con cui si rappresentano molecole chimiche, ed è necessario fornire informazioni qualitative di supporto alla predizione del modello. In questo contesto, il sistema proposto può essere utilizzato come supporto ad un utente esperto permettendogli di individuare le componenti molecolari ritenute più rilevanti per la predizione e di valutare quindi, in base alla conoscenza del problema, il tipo di meccanismi potenzialmente coinvolti. Il sistema permette inoltre di creare modelli che rispondono alle richieste REACH, che regolamenta, in ambito europeo, l'uso di questi sistemi per scopi di valutazione della pericolosità di prodotti chimici.

Il funzionamento è stato sperimentato in problemi di tossicologia, verificandone gli aspetti delle singole componenti e le prestazioni nel suo funzionamento globale, confrontandole con risultati allo stato dell'arte. Gli esperimenti effettuati mostrano che le scelte fatte a favore dell'interpretabilità della risposta non pregiudicano le prestazioni del modello. Le performance sono infatti comparabili con i migliori risultati ottenuti da sistemi più complessi e non dotati di componenti per l'analisi qualitativa. Per esempio si raggiungono risultati molto vicini allo stato dell'arte sul dataset Bursi (caratterizzato da una maggior quantità e qualità di dati).

La possibilità di lavorare direttamente sul dominio dei grafi permette di catturare aspetti topologici significativi, in modo adattivo, in base alla morfologia della struttura del grafo ed al valore della proprietà target. Questo è possibile grazie all'informazione contenuta in modo implicito nello stato prodotto dal reservoir per un vertice di un grafo, nel quale è codificata, con effetti Markoviani, l'intera struttura del grafo stesso. Questa informazione viene sfruttata dal training supervisionato della SOM che può portare ad una ripartizione dello spazio degli stati significativa rispetto al problema in esame. La mappatura dei vertici di un grafo sulle unità della SOM, che rappresenta la scomposizione del grafo da cui si ottiene una rappresentazione a dimensione fissata, viene quindi fatta in funzione del problema. Tutto questo si contrappone a metodi basati su descrittori, come il Metodo a Frammenti (MF) (introdotto ed utilizzato in questa tesi per un confronto con GraphESN-SOM), in cui un grafo è scomposto in un insieme di caratteristiche ed informazioni che devono essere esplicitate e decise a priori.

Infine, il lavoro svolto fornisce interessanti spunti per ulteriori sperimentazioni e sviluppi. Di sicuro interesse è l'approfondimento degli aspetti supervisionati della SMF, ad esempio provando procedure più sofisticate per il training supervisionato della SOM, o con analisi approfondite degli effetti sulla disposizione delle unità. Potendo disporre di buone risorse computazionali, ulteriori studi si potrebbero rivolgere verso la possibilità di mantenere l'intero stato del reservoir (mediato localmente ad ogni cluster) nella costruzione del vettore restituito dalla SMF, rinunciando ad una funzione di aggregazione. In tale modo ci sarebbe una minore perdita di informazione

da cui il readout potrebbe trarne vantaggi portando a migliori prestazioni predittive dei modelli creati. Sarebbe però necessario, in questo caso, valutare attentamente la regolarizzazione del readout. Infine, nello specifico dell’ambito tossicologico, dove è utile conoscere l’attività target di composti chimici simili al composto da analizzare, si può indagare la possibilità di effettuare una ricerca, tra i composti utilizzati per il training del modello, di composti simili sulla base delle unità attivate nella SOM come ulteriore elemento di analisi a supporto dell’interpretazione della risposta, al fine di creare strumenti automatici per la tossicologia computazionale.



# Appendice A

## Datasets

In questa appendice vengono sinteticamente descritti i 3 dataset di tossicologia, che portano a 6 problemi di classificazione. Tali problemi di classificazione sono utilizzati nelle prove sperimentali di questa tesi, descritte nel capitolo 4.

### A.1 Bursi

Il dataset Bursi [94, 87] è composto da 4204 molecole rappresentate in formato SDF [95]. Questo dataset si contraddistingue per un'ottima quantità e qualità dei dati. Per ogni molecola è riportata la mutagenicità su *salmonella typhimurium* ottenuta attraverso il *test di Ames*. Il dataset è suddiviso (in originale) in un training set composto da 3367 molecole ed un test set di 837 molecole. Per questo dataset è stato definito un problema di classificazione binaria. Il valore target è preso come +1 se la molecola è considerata attiva (mutagenica), -1 altrimenti.

### A.2 ISSCAN

Il dataset ISSCAN (versione 2a) [86, 83] è composto da 890 molecole rappresentate in formato SDF. Per ogni molecola è riportata la mutagenicità su *salmonella typhimurium* (SAL) ottenuta attraverso il *test di Ames*. Ogni molecola è classificata come mutagenica (valore 3), equivoca (valore 2), non mutagenica (valore 1) oppure ND nel caso non sia definita la classificazione. Le molecole classificate come mutageniche o equivoche sono state considerate come attive, mentre le molecole classificate come non mutageniche sono state considerate come non attive. Le molecole incomplete o con classificazione ND sono state scartate per un totale di 192 molecole (14 incomplete, 178 con valore ND). Per questo dataset è stato definito un problema di classificazione binaria. Il valore target è preso come +1 se la molecola è considerata attiva, -1 altrimenti. Il numero di molecole per le quali è stata definita una classe target è 698.

### A.3 Predictive Toxicology Challenge (PTC)

Il dataset PTC [78] è composto da 417 molecole rappresentate in formato SDF. Per ogni molecola è riportata la cancerogenicità per 4 tipi di roditori: topi maschi (MM), topi femmine (FM), ratti maschi (MR), ratti femmine (FR). Ogni molecola è classificata in una diversa categoria, a seconda della cancerogenicità, tra CE, SE, P, EE, NE, IS, E, N, per ogni tipo di roditore. Le molecole classificate come CE, SE o P sono state considerate attive, mentre le molecole classificate come N o NE sono state considerate non attive. Le molecole assegnate ad altre classi non state considerate. Per ogni tipo di roditore (FM, FR, MM, MR) è stato definito un problema di classificazione binaria. Il valore target è preso come +1 se il composto è considerato attivo, -1 se il composto è considerato non attivo. Il numero di molecole per le quali è stata definita una classe target è 349 per FM, 351 per FR, 336 per MM e 344 per MR.

# Bibliografia

- [1] T. Mitchell, *Machine Learning*. McGraw Hill, 1997.
- [2] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006.
- [3] S. Haykin, *Neural Networks: A Comprehensive Foundation*. Macmillan, 1994.
- [4] R. Rojas, *Neural Networks: A Systematic Introduction*. Springer, 1996.
- [5] G. Cybenko, “Approximation by Superpositions of a Sigmoidal Function,” *Mathematics of Control, Signals, and Systems (MCSS)*, vol. 2, no. 4, pp. 303–314, 1989.
- [6] H. T. Siegelmann and E. D. Sontag, “Turing Computability With Neural Nets,” *Applied Mathematics Letters*, vol. 4, no. 6, pp. 77–80, 1991.
- [7] B. Hammer, A. Micheli, and A. Sperduti, “Adaptive Contextual Processing of Structured Data by Recursive Neural Networks: A Survey of Computational Properties,” *Perspectives of Neural-Symbolic Integration*, vol. 77/2007, pp. 67–94, 2007.
- [8] J. Markoff, “Scientists See Promise in Deep-Learning Programs.” NY Times, 23 November 2012.
- [9] S. C. Kremer and J. F. Kolen, *Field Guide to Dynamical Recurrent Networks*. Wiley-IEEE Press, 2001.
- [10] A. Sperduti and A. Starita, “Supervised Neural Networks for the Classification of Structures,” *Transactions on Neural Networks*, vol. 8, no. 3, pp. 714–735, 1997.
- [11] P. Frasconi, M. Gori, and A. Sperduti, “A General Framework for Adaptive Processing of Data Structures,” *Transactions on Neural Networks*, vol. 9, pp. 768–786, 1998.
- [12] M. Lukoševičius and H. Jaeger, “Reservoir Computing Approaches to Recurrent Neural Network Training,” *Computer Science Review*, vol. 3, no. 3, pp. 127–149, 2009.

- [13] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt, “An Experimental Unification of Reservoir Computing Methods,” *Neural Networks*, vol. 20, no. 3, pp. 391–403, 2007.
- [14] B. Schrauwen, D. Verstraeten, and J. V. Campenhout, “An Overview of Reservoir Computing: Theory, Applications and Implementations,” in *Proceedings of the 15th European Symposium on Artificial Neural Networks (ESANN) 2007*, pp. 471–482, 2007.
- [15] C. Gallicchio and A. Micheli, “Architectural and Markovian Factors of Echo State Networks,” *Neural Networks*, vol. 24, no. 5, pp. 440–456, 2011.
- [16] H. Jaeger, “The “Echo State” Approach to Analysing and Training Recurrent Neural Networks,” GMD Technical Report 148, GMD - German National Research Institute for Computer Science, 2001.
- [17] H. Jaeger and H. Haas, “Harnessing Nonlinearity: Predicting Chaotic Systems and Saving Energy in Wireless Communication,” *Science*, vol. 304, no. 5667, pp. 78–80, 2004.
- [18] C. Gallicchio and A. Micheli, “Graph Echo State Networks,” in *Proceedings of the International Joint Conference on Neural Networks (IJCNN) 2010*, pp. 2159—2166, july 2010.
- [19] C. Gallicchio and A. Micheli, “Tree Echo State Networks,” *Neurocomputing*, vol. 101, pp. 319–337, 2013.
- [20] C. Gallicchio and A. Micheli, “Exploiting Vertices States in GraphESN by Weighted Nearest Neighbor,” 2011.
- [21] C. Gallicchio and A. Micheli, “Supervised State Mapping of Clustered GraphESN States,” in *Proceedings of the 21st Italian Workshop on Neural Networks (WIRN) 2011*, pp. 28–35, 2011.
- [22] “Guidance Document on the Validation of (Quantitative) Structure-Activity Relationship [(Q)SAR] Models,” Tech. Rep. 69, Organisation for Economic Co-Operation and Development (OECD), 2007.
- [23] T. Kohonen, “The Self-Organizing Map,” *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1464–1480, 1990.
- [24] H. Zou and T. Hastie, “Regularization and Variable Selection via the Elastic Net,” *Journal of the Royal Statistical Society*, vol. 67, no. 2, pp. 301–320, 2005.
- [25] J. Friedman, T. Hastie, and R. Tibshirani, *The Elements of Statistical Learning*. Springer, 2008.

- [26] R. Tibshirani, “Regression Shrinkage and Selection Via the Lasso,” *Journal of the Royal Statistical Society*, vol. 58, pp. 267–288, 1994.
- [27] A. M. Bianucci, A. Micheli, A. Sperduti, and A. Starita, “A Novel Approach to QSPR/QSAR Based on Neural Networks for Structures,” in *Soft Computing Approaches in Chemistry*, vol. 120 of *Studies in Fuzziness and Soft Computing*, pp. 265–296, Springer Berlin Heidelberg, 2003.
- [28] I. Gutman and O. E. Polansky, “Mathematical Concepts in Organic Chemistry,” *SIAM Review*, vol. 30, no. 2, pp. 348–350, 1988.
- [29] P. J. Werbos, “Backpropagation Through Time: What It Does and How To Do It,” *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550–1560, 1990.
- [30] R. J. Williams and D. Zipser, “A Learning Algorithm for Continually Running Fully Recurrent Neural Networks,” *Neural Computation*, vol. 1, no. 2, pp. 270–280, 1989.
- [31] R. J. Williams and D. Zipser, “Gradient-based Learning Algorithms for Recurrent Networks and Their Computational Complexity,” *Backpropagation: Theory, Architectures and Applications*, pp. 433–486, 1995.
- [32] A. Micheli, D. Sona, and A. Sperduti, “Contextual Processing of Structured Data by Recursive Cascade Correlation,” *IEEE Transactions on Neural Networks*, vol. 15, no. 6, pp. 1396–1410, 2004.
- [33] B. Hammer, A. Micheli, and A. Sperduti, “Universal Approximation Capability of Cascade Correlation for Structures,” *Neural Computation*, vol. 17, no. 5, pp. 1109–1159, 2005.
- [34] A. Micheli, “Neural Network for Graphs: a Contextual Constructive Approach,” *IEEE Transactions on Neural Networks*, vol. 20, no. 3, pp. 498–511, 2009.
- [35] A. M. Bianucci, A. Micheli, A. Sperduti, and A. Starita, “Application of Cascade Correlation Networks for Structures to Chemistry,” *Applied Intelligence*, vol. 12, no. 1-2, pp. 117–147, 2000.
- [36] C. Duce, A. Micheli, A. Starita, M. R. Tiné, and R. Solaro, “Prediction of Polymer Properties from Their Structure by Recursive Neural Networks,” *Macromolecular Rapid Communications*, vol. 27, no. 9, pp. 711–715, 2006.
- [37] F. Costa, P. Frasconi, V. Lombardo, and G. Soda, “Towards Incremental Parsing of Natural Language Using Recursive Neural Networks,” *Applied Intelligence*, vol. 19, no. 1-2, pp. 9–25, 2003.

- [38] P. Sturt, F. Costa, V. Lombardo, and P. Frasconi, “Learning First-pass Structural Attachment Preferences with Dynamic Grammars and Recursive Neural Networks,” *Cognition*, vol. 88, no. 2, pp. 133–169, 2003.
- [39] E. Francesconi, P. Frasconi, M. Gori, S. Marinai, J. Q. Sheng, G. Soda, and A. Sperduti, “Logo Recognition by Recursive Neural Networks,” *Graphics Recognition Algorithms and Systems*, pp. 104–117, 1998.
- [40] T. Gärtner, “A Survey of Kernels for Structured Data,” *ACM SIGKDD Explorations Newsletter*, vol. 5, no. 1, pp. 49–58, 2003.
- [41] J. Shawe-Taylor and N. Cristianini, *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.
- [42] L. Ralaivola, S. J. Swamidass, H. Saigo, and P. Baldi, “Graph Kernels for Chemical Informatics,” *Neural Networks*, vol. 18, no. 8, pp. 1093–1110, 2005.
- [43] B. Schölkopf and A. J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization and Beyond*. MIT Press, 2002.
- [44] V. N. Vapnik, “An Overview of Statistical Learning Theory,” *IEEE Transactions on Neural Networks*, vol. 10, no. 5, pp. 988–999, 1999.
- [45] V. N. Vapnik, “Pattern Recognition Using Generalized Portrait Method,” *Automation and Remote Control*, vol. 24, pp. 774–780, 1963.
- [46] B. E. Boser, I. M. Guyon, and V. N. Vapnik, “A Training Algorithm for Optimal Margin Classifiers,” *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, pp. 144–152, 1992.
- [47] C. Cortes and V. N. Vapnik, “Support-Vector Networks,” *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.
- [48] Q. Liao, J. Yao, and S. Yuan, “Prediction of Mutagenic Toxicity by Combination of Recursive Partitioning and Support Vector Machines,” *Molecular Diversity*, vol. 11, no. 2, pp. 59–72, 2007.
- [49] D. Haussler, “Convolution Kernels on Discrete Structures,” tech. rep., Department of Computer Science, University of California at Santa Cruz, 1999.
- [50] H. Kashima, K. Tsuda, and A. Inokuchi, “Marginalized Kernels Between Labeled Graphs,” vol. 20, p. 321, AAAI Press, 2003.
- [51] H. Fröhlich, J. Wegner, F. Sieker, and A. Zell, “Optimal Assignment Kernels for Attributed Molecular Graphs,” in *Proceedings of the 22nd*

- International Conference on Machine Learning*, pp. 225–232, ACM, 2005.
- [52] N. Lavrac and S. Dzeroski, *Inductive Logic Programming*. Ellis Horwood, 1994.
  - [53] A. Srinivasan, S. H. Muggleton, M. J. E. Sternberg, and R. D. King, “Theories for Mutagenicity: A Study in First-order and Feature-based Induction,” *Artificial Intelligence*, vol. 85, no. 1, pp. 277–299, 1996.
  - [54] J. R. Koza, “Evolution of a Computer Program for Classifying Protein Segments as Transmembrane Domains Using Genetic Programming,” pp. 244—252, AAAI Press, 1994.
  - [55] T. Washio and H. Motoda, “State of the Art of Graph-based Data Mining,” *Sigkdd Explorations Newsletter*, vol. 5, no. 1, pp. 59–68, 2003.
  - [56] C. Helma, T. Cramer, S. Kramer, and L. D. Raedt, “Data Mining and Machine Learning Techniques for the Identification of Mutagenicity Inducing Substructures and Structure Activity Relationships of Noncongeneric Compounds,” *Journal of Chemical Information and Computer Sciences*, vol. 44, no. 4, pp. 1402–1411, 2004.
  - [57] W. Maass, T. Natschläger, and H. Markram, “Real-time Computing Without Stable States: A New Framework for Neural Computation Based on Perturbations,” *Neural Computation*, vol. 14, no. 11, pp. 2531–2560, 2002.
  - [58] G. B. Huang, Q. Y. Zhu, and C. K. Siew, “Extreme Learning Machine: Theory and Applications,” *Neurocomputing*, vol. 70, no. 1, pp. 489–501, 2006.
  - [59] J. J. Steil, “Backpropagation-Decorrelation: Online Recurrent Learning with O(N) Complexity,” in *Proceedings of the International Joint Conference on Neural Networks (IJCNN) 2004*, vol. 2, pp. 843–848, IEEE, 2004.
  - [60] H. Jaeger, “Adaptive Nonlinear System Identification with Echo State Networks,” *Advances in Neural Information Processing Systems (NIPS)*, no. 15, pp. 593–600, 2002.
  - [61] P. Tiňo, M. Černansky, and L. Benuskova, “Markovian Architectural Bias of Recurrent Neural Networks,” *Transactions on Neural Networks*, vol. 15, no. 1, pp. 6–15, 2004.
  - [62] B. Hammer, P. Tiňo, and A. Micheli, “A Mathematical Characterization of the Architectural Bias of Recursive Models,” Tech. Rep. 252, 2004.

- [63] T. M. Martinetz, S. G. Berkovich, and K. J. Schulten, “Neural-Gas Network for Vector Quantization and its Application to Time-Series Prediction,” *Transactions on Neural Networks*, vol. 4, no. 4, pp. 558–569, 1993.
- [64] X. Dutoit, B. Schrauwen, J. V. Campenhout, D. Stroobandt, H. V. Brussel, and M. Nuttin, “Pruning and Regularization in Reservoir Computing,” *Neurocomputing*, vol. 72, no. 7–9, pp. 1534–1546, 2009.
- [65] J. B. Butcher, C. R. Day, P. W. Haycock, D. Verstraeten, and B. Schrauwen, “Pruning Reservoirs with Random Static Projections,” pp. 250–255, IEEE, 2010.
- [66] H. U. Kobialka and U. Kayani, “Echo State Networks with Sparse Output Connections,” *20th International Conference on Artificial Neural Networks (ICANN) 2010*, vol. 6352, pp. 356–361, 2010.
- [67] B. Efron, H. T. L. Johnstone, and R. Tibshirani, “Least Angle Regression,” *Annals of Statistics*, vol. 32, pp. 407–499, 2004.
- [68] L. Breiman, “Better Subset Regression Using the Nonnegative Garrote,” *Technometrics*, vol. 37, no. 4, pp. 373–384, 1995.
- [69] T. Kohonen and T. Honkela, “Kohonen Network,” *Scholarpedia*, vol. 2, no. 1, p. 1568, 2007.
- [70] J. Vesanto, “SOM-based Data Visualization Methods,” *Intelligent data analysis*, vol. 3, no. 2, pp. 111–126, 1999.
- [71] S. Kaski, J. Kangas, and T. Kohonen, “Bibliography of Self-Organizing Map (SOM) Papers: 1981-1997,” *Neural Computing Surveys*, vol. 1, no. 3&4, pp. 1–176, 1998.
- [72] M. Oja, S. Kaski, T. Kohonen, *et al.*, “Bibliography of Self-Organizing Map (SOM) Papers: 1998-2001 addendum,” *Neural Computing Surveys*, vol. 3, no. 1, pp. 1–156, 2003.
- [73] T. Kohonen, J. Hynninen, J. Kangas, J. Laaksonen, *et al.*, “Som Pak: The Self-Organizing Map Program Package,” tech. rep., Helsinki University of Technology, Laboratory of Computer and Information Science, 1996.
- [74] K. Kiviluoto, “Topology Preservation in Self-Organizing Maps,” in *Proceedings of the International Joint Conference on Neural Networks (IJCNN) 1996*, vol. 1, pp. 294–299, IEEE, 1996.
- [75] A. Ultsch and H. P. Siemon, “Kohonen’s Self Organizing Feature Maps for Exploratory Data Analysis,” vol. 49, pp. 305—308, Dordrecht, The Netherlands, 1990.

- [76] T. Kohonen, *Self-Organizing Maps*, vol. 30. Springer, 1995.
- [77] M. Hagenbuchner and A. C. Tsoi, “A Supervised Self-Organizing Map for Structures,” in *Proceedings of the International Joint Conference on Neural Networks (IJCNN) 2004*, vol. 3, pp. 1923–1928, IEEE, 2004.
- [78] C. Helma, R. D. King, S. Kramer, and A. Srinivasan, “The Predictive Toxicology Challenge 2000–2001,” *Bioinformatics*, vol. 17, no. 1, pp. 107–108, 2001.
- [79] R. J. Kavlock, G. Ankley, J. Blancato, M. Breen, R. Conolly, D. Dix, K. Houck, E. Hubal, R. Judson, J. Rabinowitz, *et al.*, “Computational Toxicology — A State of the Science Mini Review,” *Toxicological Sciences*, vol. 103, no. 1, pp. 14–27, 2008.
- [80] H. Kubinyi, “From Narcosis to Hyperspace: The History of QSAR,” *Quantitative Structure-Activity Relationships*, vol. 21, no. 4, pp. 348–356, 2002.
- [81] R. Benigni and C. Bossa, “Predictivity of QSAR,” *Journal of Chemical Information and Modeling*, vol. 48, no. 5, pp. 971–980, 2008.
- [82] R. Benigni and C. Bossa, “Structural Alerts of Mutagens and Carcinogens,” *Current Computer-Aided Drug Design*, vol. 2, no. 2, pp. 169–176, 2006.
- [83] R. Benigni and C. Bossa, “Structure Alerts for Carcinogenicity, and the Salmonella Assay System: A Novel Insight Through the Chemical Relational Databases Technology,” *Mutation Research/Reviews in Mutation Research*, vol. 659, no. 3, pp. 248–261, 2008.
- [84] J. Ashby, “Fundamental Structural Alerts to Potential Carcinogenicity or Noncarcinogenicity,” *Environmental Mutagenesis*, vol. 7, no. 6, pp. 919–921, 1985.
- [85] R. Benigni, C. Bossa, N. Jeliazkova, T. I. Netzeva, and A. P. Worth, “The Benigni/Bossa Rulebase for Mutagenicity and Carcinogenicity - A Module of Toxtree,” Tech. Rep. EUR 23241 EN, 2008.
- [86] R. Benigni, C. Bossa, A. M. Richard, and C. Yang, “A Novel Approach: Chemical Relational Databases, and the Role of the ISSCAN Database on Assessing Chemical Carcinogenicity,” *Annali dell'Istituto Superiore di Sanità*, vol. 44, pp. 48–56, 2008.
- [87] T. Ferrari and G. Gini, “An Open Source Multistep Model to Predict Mutagenicity from Statistical Analysis and Relevant Structural Alerts,” *Chemistry Central Journal*, vol. 4, no. Suppl 1, p. S2, 2010.

- [88] J. Vesanto, J. Himberg, E. Alhoniemi, and J. Parhankangas, “SOM Toolbox for Matlab 5,” Tech. Rep. A57, Helsinki University of Technology, apr. 2000.
- [89] M. Grbovic and S. Vucetic, “Regression Learning Vector Quantization,” in *ICDM '09: Proceedings of the Ninth IEEE International Conference on Data Mining*, pp. 788–793, IEEE, 2009.
- [90] H. Fröhlich, J. Wegner, and A. Zell, “Assignment Kernels For Chemical Compounds,” pp. 913–918, 2005.
- [91] W. W. Piegorsch and E. Zeiger, “Measuring Intra-assay Agreement for the Ames Salmonella Assay,” in *Statistical Methods in Toxicology*, pp. 35–41, Springer, 1991.
- [92] D. Weininger, “SMILES, a Chemical Language and Information System. 1. Introduction to Methodology and Encoding Rules,” *Journal of Chemical Information and Computer Sciences*, vol. 28, no. 1, pp. 31–36, 1988.
- [93] T. Ferrari, G. Gini, and E. Benfenati, “Support Vector Machines in the Prediction of Mutagenicity of Chemical Compounds,” pp. 1–6, IEEE, 2009.
- [94] J. Kazius, R. McGuire, and R. Bursi, “Derivation and Validation of Toxiphores for Mutagenicity Prediction,” *Journal of Medicinal Chemistry*, vol. 48, no. 1, pp. 312–320, 2005.
- [95] A. Dalby, J. G. Nourse, W. D. Hounshell, A. K. I. Gushurst, D. L. Grier, B. A. Leland, and J. Laufer, “Description of Several Chemical Structure File Formats Used by Computer Programs Developed at Molecular Design Limited,” *Journal of Chemical Information and Computer Sciences*, vol. 32, no. 3, pp. 244–255, 1992.