

UNIVERSITÀ DI PISA



FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI  
CORSO DI LAUREA SPECIALISTICA IN INFORMATICA

Tesi di laurea

**Modelli neurali costruttivi di tipo Reservoir  
Computing per domini strutturati**

Giulio Visco

Relatori

Controrelatore

Prof. Alessio Micheli

Prof. Stefano Chessa

Dott. Claudio Gallicchio

Anno Accademico 2010/2011

In God we trust, all others bring data.

— William Edwards Deming

## Sommario

La tesi introduce e discute nuovi modelli di Reti Neurali Ricorsive per l'apprendimento su domini strutturati che comprendano grafi indiretti e grafi diretti ciclici. Due sono i contributi maggiori realizzati: l'adozione di un approccio costruttivo e l'introduzione di un meccanismo stabile di output-feedback, entrambi innovativi nell'ambito del Reservoir Computing a cui si rifanno i modelli considerati. La combinazione di una strategia costruttiva e dell'utilizzo di modelli di tipo Reservoir Computing ha inoltre permesso la realizzazione di modelli estremamente efficienti dal punto di vista computazionale.

I modelli e le strategie individuate si configurano dunque come uno strumento utile e flessibile nel trattamento di domini complessi, collocandosi alla frontiera della ricerca nell'ambito del Machine Learning riguardante l'apprendimento su domini strutturati tramite il paradigma neurale.

L'analisi sperimentale svolta riguarda l'apprendimento di trasduzioni strutturali da dataset reali appartenenti all'ambito della Chemioinformatica.

# Indice

<b>Introduzione</b>	<b>vii</b>
<b>1 Background</b>	<b>1</b>
1.1 Reti Neurali Costruttive . . . . .	2
1.1.1 Cascade Correlation . . . . .	4
1.2 Reservoir Computing . . . . .	7
1.2.1 Echo State Networks . . . . .	10
1.3 Modelli per domini strutturati . . . . .	15
1.3.1 Terminologia . . . . .	17
1.3.2 Graph Echo State Networks . . . . .	19
<b>2 Modelli</b>	<b>27</b>
2.1 Introduzione ai modelli . . . . .	27
2.2 GraphESN costruttive . . . . .	29
2.2.1 Costruzione di una rete . . . . .	32
2.2.2 Output-feedback . . . . .	35

2.3	Modelli . . . . .	39
2.4	Costo computazionale . . . . .	42
2.4.1	Considerazioni . . . . .	45
<b>3</b>	<b>Risultati Sperimentali</b>	<b>48</b>
3.1	Dataset . . . . .	48
3.2	Esperimenti . . . . .	51
3.2.1	PTC . . . . .	54
3.2.2	Mutagenesis . . . . .	55
3.3	Considerazioni . . . . .	56
<b>4</b>	<b>Conclusioni</b>	<b>61</b>

## Elenco delle figure

1.1	Cascade Correlation. . . . .	6
1.2	Echo State Network. . . . .	11
1.3	Funzione di transizione locale di stato . . . . .	21
1.4	Encoding di una GraphESN. . . . .	24
1.5	GraphESN. . . . .	26
2.1	Dinamiche del Reservoir esteso. . . . .	37
2.2	GraphESN-ZERO. . . . .	39
2.3	GraphESN-FW. . . . .	41
2.4	GraphESN-FOF. . . . .	41
2.5	Approccio costruttivo: vantaggio computazionale . . . . .	46
3.1	PCA dei reservoir delle sotto-reti . . . . .	57
3.2	Confronto fra i modelli: errore di training . . . . .	58

## Elenco delle tabelle

3.1	Model selection: iperparametri per PTC . . . . .	54
3.2	Accuratezza media su PTC . . . . .	54
3.3	Model selection: iperparametri per Mutagenesis . . . . .	55
3.4	Accuratezza media su Mutagenesis . . . . .	56

## Introduzione

In molti settori applicativi i dati trattati trovano una propria rappresentazione naturale in forma di grafi. Per citarne solo alcuni, si hanno esempi in questo senso nella Proteomica [1], nel trattamento e riconoscimento di immagini, nell'Ingegneria del Software o nella Chimica [6]. Estendere le metodologie per trattare domini strutturati rappresenta quindi un modo per dare risposte efficaci a problemi esistenti, aprendo spazio per lo sviluppo di nuove applicazioni.

L'ampliamento della gamma dei problemi trattabili è d'altra parte una delle ragioni d'essere del Machine Learning: laddove non risulti applicabile o efficace un approccio classico, algoritmico o analitico, l'Apprendimento Automatico si configura infatti come uno strumento utile per spostare la frontiera che delimita i problemi affrontabili.

A questi due ambiti guarda quanto verrà discusso nel seguito, proponendo nuovi modelli neurali che possano apprendere funzioni definite su domini strutturati.

Dei vari approcci possibili nel contesto del Machine Learning per il trattamento di domini strutturati, quello neurale ha subito una notevole evoluzione



negli ultimi anni ed è ad oggi una frontiera della ricerca stimolante, che non manca di mostrare criticità e margini di miglioramento. Fra gli aspetti critici certamente più rilevanti è possibile individuarne due particolarmente importanti in relazione al lavoro svolto: il costo computazionale e la classe degli input trattabili. Il costo computazionale, che si configura come un elemento critico nei modelli neurali in generale, acquista in effetti un ruolo particolarmente determinante nel trattamento di domini complessi (e.g. sequenze, alberi o grafi): in una disciplina fortemente orientata alle applicazioni come il Machine Learning, il costo computazionale rappresenta infatti un vincolo tutt'altro che teorico sull'applicabilità dei modelli. Il secondo aspetto rappresenta invece un limite specifico delle Reti Neurali Ricorsive, adibite al trattamento dei domini strutturati, per le quali è talvolta necessario ricorrere ad assunzioni circa la struttura degli input (i.e. Grafi Diretti Aciclici) che riducono la gamma dei task affrontabili (si veda il paragrafo 1.3). Con i modelli proposti si è dunque cercato di ricondurre la soluzione di entrambi i problemi ad un'unica strategia che permettesse di apprendere, con buone performance e ad un costo computazionale vantaggioso, funzioni su un dominio di input che comprendesse i grafi non diretti ed i grafi ciclici.

La soluzione individuata consiste nell'introduzione dell'approccio costruttivo (paragrafo 1.1) nell'ambito del paradigma emergente del Reservoir Computing (paragrafo 1.2), applicato al trattamento di domini strutturati (paragrafo 1.3). Ne risulta una strategia efficiente e flessibile che ha permesso la formulazione di tre nuovi modelli di reti neurali ricorsive (capitolo 2). L'aver intrapreso un percorso di integrazione fra due mondi ad oggi separati, quello delle Reti Neurali Costruttive e quello del Reservoir Computing, ha inoltre

permesso di mettere in luce caratteristiche originali dei modelli proposti, come l'introduzione di un meccanismo stabile di output-feedback (paragrafo 2.2.2).

Le capacità dei modelli proposti sono state discusse e testate in maniera rigorosa su task di Chemioinformatica relativi a dataset reali noti in letteratura (capitolo 3), evidenziando come l'approccio proposto permetta in molti casi di migliorare le performance raggiunte tramite l'applicazione di modelli non costruttivi. Parte dei risultati riportati nel seguito è inoltre discussa nell'articolo scientifico *Constructive Reservoir Computation with Output Feedbacks for Structured Domains*, inviato all'European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning<sup>1</sup> ed attualmente in fase di revisione.

L'esposizione degli argomenti nel testo è articolata come segue:

**Nel capitolo 1** si introducono i temi, le tecniche ed i modelli alla base del lavoro svolto. Nel corso di questa rassegna dello stato dell'arte vengono in particolare presentati i principi dell'approccio costruttivo e del Reservoir Computing, descrivendo anche modalità ed obiettivi nel trattamento dei domini strutturati.

**Nel capitolo 2** vengono introdotti i modelli originali, oggetto del lavoro di tesi, ne vengono presentate le caratteristiche e discussi gli aspetti specifici.

**Nel capitolo 3** sono descritti i risultati sperimentali ottenuti testando le capacità dei modelli su una serie di problemi del mondo reale appartenenti all'ambito della Chemioinformatica.

---

<sup>1</sup><http://www.dice.ucl.ac.be/esann/>

**Nel capitolo 4** la tesi si conclude con una valutazione del lavoro svolto nonché l'individuazione di quegli aspetti che potrebbero essere oggetto di un ulteriore, specifico, lavoro di indagine e sperimentazione.

Prima di procedere oltre, un ultimo aspetto merita di essere sottolineato: discutendo temi che vanno dalle basi dell'Apprendimento Automatico fino alle frontiere della ricerca, la trattazione non può avere la pretesa né la presunzione di risultare esaustiva in ogni sua parte. Il testo, seppur corredato di richiami e riferimenti bibliografici, non può dunque fare a meno di assumere nel lettore una conoscenza anche superficiale almeno degli strumenti di base, della terminologia, delle motivazioni del Machine Learning e delle Reti Neurali [14, 28, 13].

Il capitolo introduce teorie e tecniche alla base del lavoro svolto. È opportuno premettere che quanto verrà descritto nel seguito rappresenta un sottoinsieme eterogeneo all'interno del mondo del Machine Learning, sia dal punto di vista cronologico che da quello modellistico, che però trova una sintesi nei modelli sviluppati (si veda il capitolo 2 a pagina 27). La trattazione procede dunque secondo una suddivisione concettuale che guarda principalmente ai modelli ed ai domini applicativi.

Il paragrafo 1.1 introduce l'approccio costruttivo nell'allenamento delle Reti Neurali e descrive l'algoritmo Cascade Correlation per lo sviluppo di reti feedforward costruttive.

Nel paragrafo 1.2 viene presentato il paradigma del Reservoir Computing, delineandone caratteristiche e motivazioni, e vengono introdotte le Echo State Networks nella loro formulazione classica di Reti Neurali Ricorrenti utilizzate per processare sequenze di input.

Il paragrafo 1.3 descrive il problema dell'apprendimento di dati strutturati, indicando alcune delle soluzioni offerte dal paradigma neurale in questo campo.

## 1.1 Reti Neurali Costruttive

La maggior parte degli algoritmi per Reti Neurali prevede l'uso di modelli con architetture statiche. Più esattamente, reti con una topologia prefissata vengono create e successivamente allenate: i pesi sulle connessioni vengono fatti variare, ma non l'architettura in sé.

Uno degli evidenti svantaggi di un simile approccio sta nella necessità di dover evitare i casi in cui la rete risulti essere, per la propria struttura e quindi al netto delle modifiche ai pesi, troppo o troppo poco complessa per il task che si vuole affrontare.

Le *Reti Neurali Costruttive* [34] offrono una valida soluzione al problema di dover stabilire a priori la topologia della rete affinché possa ben adattarsi al task che si vuole risolvere. L'allenamento di una rete neurale costruttiva inizia tipicamente con una rete di piccole dimensioni — anche senza alcuna unità nascosta, nel caso delle reti feedforward — e procede aggiungendo nuove unità alla rete finché questa non abbia raggiunto una complessità compatibile con il problema in questione.

Al di là dei dettagli implementativi possiamo dunque individuare le due caratteristiche alla base dell'approccio costruttivo.

- Una rete viene vista come formata da più *unità computazionali* con una capacità limitata, che vengono allenate per risolvere un sotto-problema rispetto al task affrontato.
- La rete, nel suo complesso, *aumenta la propria complessità* nel corso del training, adattandosi al task che le viene sottoposto.

È importante sottolineare come questi due principi, benché sviluppati nell'ambito delle reti feedforward, possano essere applicabili ad un'ampia classe di problemi o modelli.

I principali vantaggi offerti dall'adozione di un approccio costruttivo sono:

- Il superamento della necessità di dover fissare a priori la topologia della rete, che diventa dunque adattiva e viene dinamicamente “appresa” dalla rete stessa.
- La possibilità di definire per le sotto-reti dei task specifici, che possano essere trattati in maniera più efficace o più efficiente rispetto al problema affrontato (si veda ad esempio il paragrafo 1.1.1 nella pagina successiva).
- L'adozione di una politica locale nell'aggiornamento dei pesi, con il duplice vantaggio di evitare i problemi legati all'adattamento dei pesi dell'intera rete (e.g. vanish del gradiente) e di consentire l'implementazione di meccanismi di caching/memoization per le porzioni di rete non direttamente interessate dal learning locale.
- La possibilità di circoscrivere il learning ad unità computazionali, o sotto-reti, più semplici della rete nel suo complesso. Questo consente in particolare l'impiego di algoritmi di apprendimento specifici e meno onerosi dal punto di vista computazionale rispetto a quelli necessari ad allenare l'intera rete.

A fronte dei vantaggi offerti, le Reti Neurali Costruttive presentano tuttavia alcune criticità legate alla capacità della rete di crescere. Poiché generalmente ogni nuova unità è connessa alle precedenti, infatti, le reti di grandi dimensioni

tendono ad avere molti layer ed unità con un numero di connessioni in input molto elevato, il che può avere un grosso impatto sul processo di learning e compromettere la scalabilità del modello. Oltre a questo, il fatto che la rete possa aumentare il proprio numero di unità indefinitamente, guidata unicamente dall'input, espone i modelli costruttivi al verificarsi di situazioni di overfitting. In altri termini, la capacità accrescere la propria complessità per adattarsi agli esempi di training può portare la rete a “focalizzarsi” eccessivamente su questi, perdendo invece la capacità di generalizzare, ovvero di poter funzionare correttamente anche su input mai visti in precedenza.

### 1.1.1 Cascade Correlation

L'algoritmo *Cascade Correlation* [4, 21, 29] rappresenta probabilmente uno dei più diffusi casi di applicazione dell'approccio costruttivo nell'allenamento di reti neurali feedforward. L'intuizione alla base dell'algoritmo sta nell'idea di allenare nuove unità computazionali perché possano (i) contribuire alla risoluzione del task affrontato risolvendo dei sotto-problemi di natura diversa e semplificata e (ii) avvalersi delle informazioni precedentemente apprese dalla rete nel corso del processo costruttivo. In particolare ad ogni nuova unità viene affidato il compito di *massimizzare la correlazione* fra il proprio output e l'errore commesso dalla rete, con lo scopo di correggerlo.

L'evoluzione dell'algoritmo è la seguente. Inizialmente la rete non ha unità nascoste; le connessioni input-output vengono dunque allenate sul training-set attraverso un algoritmo di apprendimento adatto a reti con un singolo strato (e.g. delta-rule o algoritmo di apprendimento del Perceptron) e senza necessità

di utilizzare back-propagation.

Dopo aver effettuato l'allenamento viene calcolato l'errore commesso dalla rete: se si è soddisfatti della performance raggiunta, in termini di fitting, allora l'algoritmo termina, altrimenti si procede nel tentativo di ridurre l'errore.

Per ridurre l'errore, una nuova unità nascosta, chiamata *candidata*, viene aggiunta alla rete: collegata sia all'input che ad ogni altra unità nascosta esistente, viene allenata perché il suo output abbia correlazione massima con l'errore residuo commesso dalla rete. La correlazione (non normalizzata)  $S$  fra l'uscita della rete  $V$  e l'errore residuo  $E_o$  osservato all'unità di output  $o$ -esima è definita come

$$S = \sum_o \left| \sum_p (V_p - \bar{V})(E_{p,o} - \bar{E}_o) \right| \quad (1.1)$$

dove  $p$  indica i pattern in input e  $\bar{V}$  ed  $\bar{E}_o$  sono i valori medi, dell'uscita e dell'errore rispettivamente, calcolati su tutti i pattern del training-set.

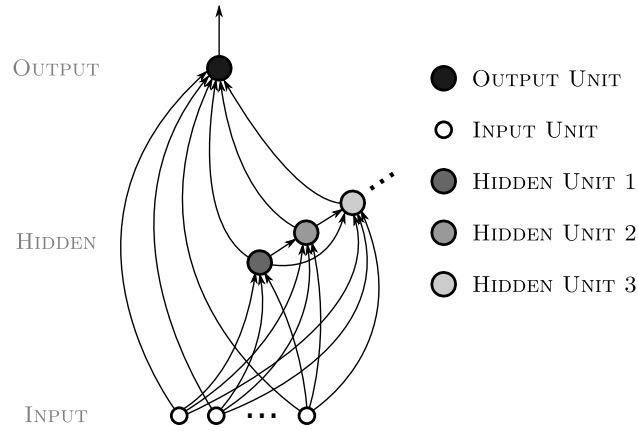
L'allenamento della candidata avviene attraverso una *ascesa del gradiente* che sfrutta la derivata parziale di  $S$  rispetto al generico peso  $w_i$

$$\frac{\partial S}{\partial w_i} = \sum_{p,o} \sigma_o (E_{p,o} - \bar{E}_o) f'_p I_{i,p} \quad (1.2)$$

dove  $\sigma_o$  è il segno della correlazione fra l'output della candidata e l'output  $o$ ,  $f'_p$  è la derivata della funzione di attivazione della candidata (e.g. tangente iperbolica) applicata ai suoi input per il pattern  $p$ -esimo ed  $I_{i,p}$  è l' $i$ -esimo input che la candidata riceve per il pattern  $p$ .

In questa fase è inoltre possibile ricorrere all'uso di un *pool* di candidate,





**Figura 1.1:** Cascade Correlation. Architettura di una rete con 1 unità di output e 3 unità nascoste.

ognuna con pesi iniziali random, in modo da variare le condizioni iniziali dell'algoritmo di apprendimento e scegliere poi l'unità che abbia raggiunto il massimo valore di  $S$ .

Ad apprendimento ultimato la nuova unità viene stabilmente aggiunta alla rete: i pesi sulle sue connessioni in ingresso vengono “congelati”, di modo che rimarranno inalterati per il resto del processo di costruzione della rete, ed il suo output viene collegato allo strato di output della rete. Il “congelamento” dei pesi sulle connessioni in input alla nuova unità ha risvolti importanti sulle caratteristiche dell'algoritmo. Una volta allenata, infatti, l'unità candidata agirà permanentemente come un *feature detector* e la sua uscita potrà essere presentata allo strato di output o ad altre unità nascoste esattamente come un input aggiuntivo. Benché la rete evolva secondo una architettura a più layer (i.e. ogni unità rappresenta un layer a sé stante, figura 1.1), dunque, gli input e le uscite delle unità nascoste possono essere considerati come appartenenti ad un unico strato ai fini dell'allenamento, il che permette l'uso di algoritmi

di apprendimento semplici e poco onerosi dal punto di vista computazionale.

Aggiunta una nuova unità nascosta si procede nuovamente con l'allenamento delle connessioni verso l'output della rete, con la valutazione dell'errore commesso ed eventualmente con l'aggiunta di nuove unità, finché il fitting non sia ritenuto soddisfacente. L'algoritmo evolve dunque in maniera greedy, aggiungendo unità nascoste finché non siano rispettati determinati criteri di stop.

Per chiarire meglio in che modo evolve la topologia di una rete allenata con Cascade Correlation, la figura 1.1 nella pagina precedente mostra l'architettura di una rete con 3 unità nascoste ed un'unica unità di output.

## 1.2 Reservoir Computing

Il *Reservoir Computing* [23, 40] è un paradigma emergente, nell'ambito delle *Reti Neurali Ricorrenti*, che ha origine da due classi di modelli: *Echo State Networks* (ESN) [17] e *Liquid State Machines* (LSM) [24]. Benché entrambi i modelli condividano — e contribuiscano a definire — i tratti distintivi del Reservoir Computing, nel seguito si farà riferimento in maniera specifica alle ESN, maggiormente legate ad un approccio computazionale<sup>1</sup> e vere progenitrici dei modelli oggetto del lavoro svolto.

Prima di descrivere il funzionamento di una ESN, è utile delineare motivazioni, intuizioni e metodi che caratterizzano il Reservoir Computing come paradigma a sé stante per il trattamento di dati strutturati<sup>2</sup>.

---

<sup>1</sup>Le LSM nascono infatti nell'ambito delle neuroscienze e mantengono caratteristiche di forte ispirazione biologica.

<sup>2</sup>Il termine *dati strutturati* è in questo contesto volutamente generico. Benché in questo paragrafo ci si riferisca a dati in forma di sequenze, dominio proprio delle Reti

Una generica Rete Neurale Ricorrente viene usata per elaborare dati sequenziali apprendendo una *trasduzione di sequenza*, ovvero una funzione da un dominio di sequenze di input ad un dominio di sequenze di output. Tale compito è realizzato modellando un sistema dinamico in cui lo stato interno  $\mathbf{x}(n)$  e l'output  $\mathbf{y}(n)$  si possono descrivere per l'input  $n$ -esimo,  $\mathbf{u}(n)$ , come

$$\begin{aligned}\mathbf{x}(n) &= \tau(\mathbf{u}(n), \mathbf{x}(n-1)) \\ \mathbf{y}(n) &= g(\mathbf{x}(n))\end{aligned}\tag{1.3}$$

Dalla formula risulta evidente come lo stato interno e l'output giochino un ruolo differente: la *funzione di transizione di stato*  $\tau$  implementa un processo di codifica ricorsivo della sequenza in input, che sfrutta informazioni sul contesto ed ha quindi una propria memoria, mentre  $\mathbf{y}(n)$  è il risultato di una funzione pura, senza memoria, della codifica in  $\mathbf{x}(n)$ .

Secondo l'approccio classico, l'allenamento delle Reti Neurali Ricorrenti usa tecniche di discesa del gradiente in cui tutte le connessioni della rete vengono modificate per minimizzare l'errore: nessuna suddivisione concettuale fra stato interno ed output viene realizzata benché le differenze emergano dal punto di vista algoritmico (l'output, a differenza dello stato interno, è infatti direttamente confrontabile con il target). Gli algoritmi di apprendimento più comuni, *Back Propagation Through Time* (BPTT) [41] e *Real-Time Recurrent Learning* (RTRL) [42], soffrono tuttavia di alcuni svantaggi. In particolare:

- L'aggiornamento graduale dei parametri può modificare le dinamiche della rete fino a far degenerare le informazioni del gradiente, di modo

---

Neurali Ricorrenti e quindi delle ESN, è vero infatti che le caratteristiche del paradigma possono essere riportate a domini strutturati anche più complessi, come i grafi (si veda il paragrafo 1.3.2 a pagina 19).

che la convergenza possa non essere garantita [3].

- Il costo computazionale per l'aggiornamento è molto alto —  $O(N^2)$  per BPTT e  $O(N^4)$  per RTRL, in una rete con  $N$  unità — e riduce la possibilità di utilizzare reti di grandi dimensioni.
- Il gradiente decresce esponenzialmente nel tempo, per cui risulta difficile apprendere dipendenze a lungo termine [2].

Il paradigma del Reservoir Computing nasce dunque con l'idea di evitare almeno parzialmente questi problemi adottando un approccio radicalmente differente:

- Una rete neurale ricorrente, chiamata *reservoir*, viene generata *in maniera casuale* ed ha lo scopo di espandere il segnale in input, mantenendo nel proprio stato interno una trasformazione non lineare del passato. I pesi del reservoir rimangono *inalterati* durante il training.
- L'output viene generato come una *combinazione lineare* dei segnali provenienti dalle unità del reservoir, passivamente eccitate dall'input. Il *readout* lineare viene adattato, in fase di learning, utilizzando il segnale del teacher come target.

La suddivisione fra stato interno della rete ed output è quindi in questo caso resa esplicita e si sfrutta la capacità del reservoir di mantenere una trasformazione non lineare del passato, anche senza dover effettuare l'apprendimento [39], per limitare l'azione del learning al solo readout.

Benché il Reservoir Computing si caratterizzi come un paradigma a sé, è giusto osservare che l'idea di gestire in maniera specifica e separata readout e

stato interno trova altri esempi nell'ambito del Machine Learning: è il caso, ad esempio, dell'algoritmo di apprendimento *Backpropagation-Decorrelation*<sup>3</sup> (BPDC) [38] o delle *Extreme Learning Machines* (ELM) [16]. In termini ancor più generali, l'approccio del Reservoir Computing è inoltre riconducibile all'utilizzo di un kernel, come sottolineato anche in [20].

### 1.2.1 Echo State Networks

Definiamo formalmente le ESN [17, 18, 19], fissando anche parte della terminologia che verrà adottata nel seguito.

Consideriamo una rete con  $N_U$  unità di input,  $N_R$  unità ricorrenti interne (reservoir), ed  $N_Y$  unità di output. Indichiamo con  $\mathbf{u}(n) \in \mathbb{R}^{N_U}$  l'input all'istante  $n$ , appartenente alla sequenza  $s(\mathbf{u}) = [\mathbf{u}(1), \mathbf{u}(2), \dots, \mathbf{u}(k)]$ , con  $\mathbf{x}(n) \in \mathbb{R}^{N_R}$  le attivazioni delle unità del reservoir e con  $\mathbf{y}(n) \in \mathbb{R}^{N_Y}$  l'output. Usiamo le seguenti matrici per i pesi delle connessioni:  $\mathbf{W}_{\text{in}} \in \mathbb{R}^{N_R \times (N_U+1)}$  per l'input,  $\mathbf{W} \in \mathbb{R}^{N_R \times N_R}$  per le connessioni interne e  $\mathbf{W}_{\text{out}} \in \mathbb{R}^{N_Y \times (N_R+1)}$  per le connessioni verso le unità di output, ovvero verso il readout.

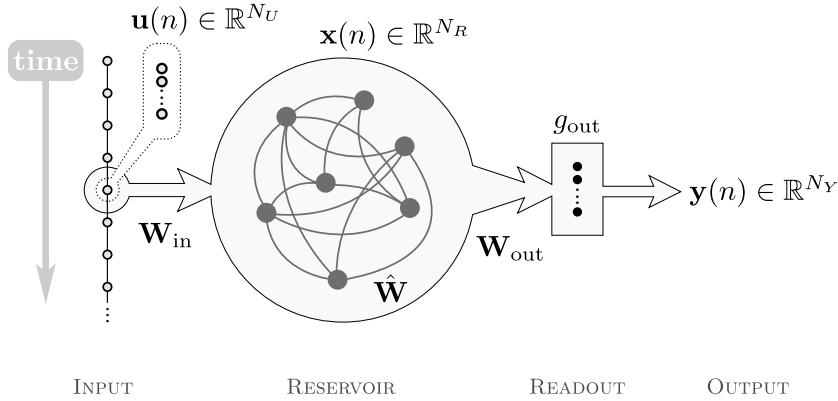
Le equazioni di base che determinano l'evoluzione del sistema sono le seguenti:

$$\begin{aligned} \mathbf{x}(n) &= \tau(\mathbf{u}(n), \mathbf{x}(n-1)) = f(\mathbf{W}_{\text{in}}\mathbf{u}(n) + \hat{\mathbf{W}}\mathbf{x}(n-1)) \\ \mathbf{y}(n) &= g(\mathbf{x}(n)) = g_{\text{out}}(\mathbf{W}_{\text{out}}\mathbf{x}(n)) \end{aligned} \tag{1.4}$$

dove  $f$  ed  $g_{\text{out}}$  sono funzioni applicate elemento per elemento e corrispondono rispettivamente alle funzioni di attivazione delle unità del reservoir, tipicamente sigmoidali (e.g. *tanh*), e delle unità di output.

---

<sup>3</sup>Per le sue caratteristiche, l'algoritmo BPDC viene talvolta indicato a tutti gli effetti come appartenente all'ambito del Reservoir Computing. Si veda, ad esempio, [35].



**Figura 1.2:** Schematizzazione grafica di una Echo State Network (ESN).

Varianti comuni prevedono inoltre l'esistenza di connessioni dirette dall'input al readout o all'indietro, dal readout al reservoir (si veda ad esempio [17, 18]). La presenza di queste ultime in particolare implica considerazioni specifiche. Connessioni di questo tipo, che possiamo chiamare *output-feedback* secondo una terminologia che verrà ripresa in seguito, offrono infatti l'opportunità di modificare le dinamiche interne del reservoir in maniera supervisionata ma hanno impatto sulla stabilità complessiva del modello [22, 43].

La figura 1.2 mostra in maniera schematica la struttura di una ESN come quella descritta dall'equazione (1.4).

L'allenamento di una ESN avviene, in maniera supervisionata, sulla base dei valori target  $\mathbf{y}_{target}(n)$ . A differenza di quanto accade nell'approccio classico, solo  $\mathbf{W}_{out}$  è interessata dall'aggiornamento dei pesi in fase di learning, mentre le altre matrici dei pesi rimangono invariate. Proprio per questa caratteristica è tuttavia necessario che le matrici  $\mathbf{W}_{in}$  e  $\hat{\mathbf{W}}$  soddisfino determinati requisiti: informalmente possiamo dire che è necessario che lo stato interno della rete sia un'*eco* della sequenza in input.

In [17, 18] viene definita la *echo state property*, che descrive la condizione basilare per il corretto funzionamento del reservoir. Intuitivamente la echo state property implica che l'effetto dello stato  $\mathbf{x}(n)$  e dell'input  $\mathbf{u}(n)$  sugli stati futuri,  $\mathbf{x}(n+t)$ , svanisca con il passare del tempo ( $t \rightarrow \infty$ ), senza persistere né essere amplificato. Di conseguenza le attivazioni della rete, dopo una fase transitoria, dipenderanno unicamente dalla sequenza in input e non dallo stato iniziale, che può dunque essere arbitrario.

Assumendo che una rete abbia funzioni di attivazione sigmoidali è possibile individuare una condizione sufficiente per il verificarsi di una simile situazione ed una condizione sufficiente perché invece non si verifichi. Sia la matrice  $\hat{\mathbf{W}}$  tale che

$$\sigma_{max} < 1 \quad (1.5)$$

con  $\sigma_{max}$  massimo valore singolare, allora la rete possiede la echo state property per ogni input ammissibile  $s(\mathbf{u})$ .

Sia la matrice  $\hat{\mathbf{W}}$  tale da avere raggio spettrale

$$\rho(\hat{\mathbf{W}}) = |\lambda_{max}| > 1 \quad (1.6)$$

dove  $\lambda_{max}$  è l'autovalore di modulo massimo, allora la rete non ha la echo state property se la sequenza nulla è un input ammissibile.

In [10] la presenza della echo state property viene messa in relazione con la *contrattività* della funzione di transizione di stato  $\tau$ . In particolare si dimostra

che se  $\tau$  è contrattiva con parametro  $C < 1$ , ovvero

$\exists C \in [0, 1)$  tale che  $\forall \mathbf{u} \in \mathbb{R}^{N_U}, \forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^{N_R}$  :

$$\|\tau(\mathbf{u}, \mathbf{x}) - \tau(\mathbf{u}, \mathbf{x}')\| \leq C \|\mathbf{x} - \mathbf{x}'\| \quad (1.7)$$

per una qualsiasi norma  $\|\cdot\|$  nello spazio degli stati  $\mathbb{R}^{N_R}$ , allora la echo state property è garantita. Oltre ad assicurare la stabilità della rete, la contrattività della funzione di transizione di stato del reservoir determina dei vincoli sull'evoluzione degli stati della rete. Lo spazio degli stati del reservoir risulta infatti avere una *natura Markoviana*: gli stati corrispondenti a due diverse sequenze di input che abbiano un suffisso comune saranno tanto più vicini quanto maggiore sarà la lunghezza del suffisso comune. Questo cosiddetto bias Markoviano ha forte influenza sulla capacità computazionale del modello: in [12, 39] si mostra come una Rete Neurale Ricorrente che abbia una funzione di transizione di stato contrattiva e spazio degli stati limitato possa essere approssimata arbitrariamente bene dalla classe dei modelli su sequenze con dinamiche di stato Markoviane (e.g. Variable Memory Length Markov Models [31]). Ne risulta che, anche senza alcun learning, gli stati interni corrispondenti a sequenze di input con suffissi comuni tendano ad essere naturalmente raggruppati o, in altri termini, che la rete abbia per una propria caratteristica architetturale la capacità di discriminare le sequenze di input sulla base del loro suffisso.

Questo bias architetturale Markoviano, come visto strettamente legato alla echo state property, rappresenta di fatto l'essenza del modello e ne giustifica l'intuizione di base: sfruttare le caratteristiche strutturali della rete, in grado



di per sé di discriminare sequenze con suffissi diversi, per limitare l'azione del learning alla sola, semplice, funzione di output  $g$ .

Nonostante l'apprendimento richieda un basso costo in termini computazionali, le ESN sono state applicate con successo a molti problemi ottenendo risultati migliori rispetto ad altri approcci precedenti (si veda ad esempio [19], [33, pag. 8], o [23, pag. 5] per un elenco dettagliato). Il modello è inoltre semplice e segue un paradigma molto generale: questo offre ampi margini di scelta e sperimentazione nell'implementazione del learning, nella topologia, nella scelta delle funzioni utilizzate e addirittura nell'implementazione fisica della rete<sup>4</sup>. Fra gli aspetti positivi delle ESN è infine opportuno menzionare l'alta plausibilità biologica, che caratterizza il Reservoir Computing in generale e che continua a rappresentare un fattore importante nell'ambito delle Reti Neurali.

Le ESN mostrano tuttavia anche dei limiti: alcuni di natura intrinseca al modello, come l'alto numero di unità impiegate che rende difficile l'implementazione fisica con risorse hardware limitate [30], ed altri legati al fatto che la ricerca in materia sia ancora in una fase poco avanzata. Gli effetti del rumore nei dati in input o la replicabilità delle condizioni di buon funzionamento della rete sono ad esempio fattori determinanti ma non completamente spiegati, che possono mettere in dubbio l'applicabilità del modello ad ampie classi di problemi [30].

Per contestualizzare meglio quanto verrà descritto in seguito è inoltre opportuno sottolineare come la ricerca sulle ESN sia ad oggi in larghissima parte

---

<sup>4</sup>Si veda [5] per un esempio in cui la rete — basata però su LSM — viene implementata attraverso hardware “poco convenzionale”.

orientata all'apprendimento di trasduzioni di sequenze e come l'applicazione dei principi del Reservoir Computing nell'ambito del trattamento di domini più complessi, come alberi o grafi, sia da considerarsi di vivissima attualità.

### 1.3 Modelli per domini strutturati

La possibilità di trattare domini di input in cui le informazioni siano strutturate (e.g. grafi) rappresenta una delle sfide che attualmente dominano e danno impulso ad una parte del mondo del Machine Learning. Molti problemi dell'informatica, della chimica, relativi all'elaborazione del linguaggio naturale, all'analisi di immagini o documenti, trovano infatti una propria naturale formulazione all'interno di domini più complessi della semplice rappresentazione tramite vettori numerici o sequenze.

Nell'ambito neurale, le *Reti Neurali Ricorsive* [6, 36] generalizzano le Reti Neurali Ricorrenti affinché possano gestire domini di input strutturati. Reti di questo tipo sono basate su un processo di encoding in cui gli stati vengono calcolati ricorsivamente in accordo con le relazioni topologiche fra i vertici dell'input. La possibilità di tenere in considerazione la topologia durante l'encoding e di far sì che l'intero processo sia adattivo, delimita la separazione, sia concettuale che pratica, tra le Reti Neurali Ricorsive e l'approccio classico [36] che prevede invece l'uso di conoscenza pregressa per codificare i dati.

Il processo di encoding rappresenta dunque la chiave dell'approccio che caratterizza i modelli neurali ricorsivi: ne determina i vantaggi e, tuttavia, ne mette anche in luce i limiti. In particolare, per garantire che ogni input abbia una rappresentazione appropriata, ovvero per garantire la convergenza del

sistema dinamico che implementa il processo di encoding, si ricorre all’assunzione di *causalità* del sistema di transizione [36]. Essendo la codifica “guidata” dalla topologia dell’input, si assume infatti che lo stato corrispondente ad un vertice dipenda unicamente dall’etichetta del vertice stesso e dagli stati corrispondenti ai vertici discendenti da esso. Se questo ha il vantaggio di garantire la convergenza ha, d’altra parte, forte impatto sulla classe degli input trattabili: l’assunzione di causalità impone infatti la presenza di un ordinamento topologico fra i vertici, limitando il dominio di input ai soli Grafi Diretti Aciclici (e.g. alberi radicati).

Nella storia recente delle Reti Neurali Ricorsive si ritrovano tuttavia dei modelli tesi a superare le limitazioni imposte dal vincolo di causalità. Ai fini di una completa comprensione del lavoro svolto, siamo particolarmente interessati a menzionare due approcci che, seppur non direttamente legati fra loro, trovano una sintesi nei modelli proposti (si veda il capitolo 2).

Prima con la *Contextual Recursive Cascade Correlation* (CRCC) [27, 11] poi con il modello *Neural Network for Graphs* (NN4G) [26], si fa leva sull’approccio costruttivo per limitare e poi superare gli effetti dell’assunzione di causalità. In particolare si sfrutta il processo incrementale di costruzione della rete per comporre una rete feedforward, che non risenta dunque delle eventuali dipendenze cicliche nell’input. I modelli citati mettono anche l’accento sull’importanza del fatto che, durante il processo di costruzione della rete, le nuove unità possano usufruire delle informazioni rese disponibili dalle unità precedenti (i.e. le unità “congelate”). Questa caratteristica, vera anche nel caso delle reti feedforward, acquisisce particolare rilevanza nel caso di input strutturati: lo stato di ogni unità è infatti funzione anche della

topologia dell'input. Per questo le informazioni trasmesse alle unità successive contribuiscono alla formazione di un *contesto*, che cresce con il crescere della rete [26]. L'approccio costruttivo permette dunque a questi modelli di sfruttare informazioni contestuali altrimenti inaccessibili.

Un secondo approccio per garantire la stabilità del processo di encoding anche in presenza di dipendenze cicliche nell'input fa invece affidamento su un setting contrattivo della funzione di transizione di stato (si veda il paragrafo 1.2.1). Rientrano in questo filone i modelli *Graph Neural Network* (GNN) [32] e le *Graph Echo State Network* (GraphESN). In quanto diretta progenitrice dei modelli proposti, quest'ultima tipologia di reti verrà discussa in dettaglio nel seguito (si veda il paragrafo 1.3.2), non prima di aver introdotto formalmente il problema dell'apprendimento su domini strutturati.

### 1.3.1 Terminologia

Nel corso di questo paragrafo sarà formulato il problema dell'apprendimento su domini strutturati, introducendo terminologia e notazione che verranno utilizzate nel corso della tesi.

Un grafo  $\mathbf{g}$ , appartenente ad un insieme di grafi  $\mathcal{G}$ , è una coppia  $(V(\mathbf{g}), E(\mathbf{g}))$ , dove  $V(\mathbf{g})$  denota l'insieme dei vertici di  $\mathbf{g}$  ed  $E(\mathbf{g}) = \{(u, v) \mid u, v \in V(\mathbf{g})\}$  denota l'insieme di archi di  $\mathbf{g}$ . Per semplicità nel seguito gli insiemi  $V(\mathbf{g})$  e  $E(\mathbf{g})$  saranno indicati come  $V$  ed  $E$  rispettivamente, mantenendo implicito il riferimento al grafo  $\mathbf{g}$ . Nel caso di grafi indiretti definiamo i *vicini* del vertice  $v$  di un grafo come l'insieme dei vertici ad esso adiacenti<sup>5</sup>,

---

<sup>5</sup>Nel caso di grafi diretti è possibile un'ulteriore suddivisione dei vertici adiacenti in *predecessori* e *successori*. Per semplicità e per una maggiore aderenza ai task affrontati questa formulazione viene tralasciata.

$\mathcal{N}(v) = \{u \in V \mid (u, v) \in E\}$ . Sia il grado di un vertice  $v$  il numero dei suoi vicini,  $\text{degree}(v) = |\mathcal{N}(v)|$ , indichiamo con  $k$  il *grado massimo* riscontrabile sull'insieme  $\mathcal{G}$ .

Consideriamo che ad ogni vertice di un grafo sia associata un'etichetta numerica  $\mathbf{u}(v) \in \mathbb{R}^{N_U}$ , dove  $\mathbb{R}^{N_U}$  denota uno spazio di input di etichette vettoriali. Indichiamo l'insieme dei grafi con etichette sui vertici in  $\mathbb{R}^{N_U}$  con  $(\mathbb{R}^{N_U})^\#$ .

Obiettivo di un modello per domini strutturati è l'apprendimento di una *trasduzione strutturale*  $\mathcal{T}$ , ovvero di una funzione da un dominio di grafi in input  $(\mathbb{R}^{N_U})^\#$  ad un dominio di grafi in output  $(\mathbb{R}^{N_Y})^\#$ :

$$\mathcal{T} : (\mathbb{R}^{N_U})^\# \rightarrow (\mathbb{R}^{N_Y})^\# \quad (1.8)$$

Indichiamo con  $\mathbf{y}(v) \in \mathbb{R}^{N_Y}$  l'etichetta vettoriale associata al vertice  $v$  del grafo nel dominio di output.

Distinguiamo due tipi di trasduzioni: in una trasduzione *structure-to-structure* il grafo in output è isomorfo a quello in input, ovvero  $\mathcal{T}$  associa un vertice in output ad ogni vertice dell'input, mentre in una trasduzione *structure-to-element* un singolo output vettoriale è associato all'intero grafo.

Nel caso dell'apprendimento supervisionato, l'allenamento avviene avvalendosi di un training-set  $\mathfrak{T} = \{(\mathbf{g}, \mathbf{y}_{\text{target}}(g)) \mid \mathbf{g} \in \mathcal{G}, \mathbf{y}_{\text{target}}(\mathbf{g}) \in (\mathbb{R}^{N_Y})^\#\}$ , dove  $\mathbf{y}_{\text{target}}(\mathbf{g})$  indica il target associato ad uno specifico input e può essere un grafo etichettato (isomorfo a  $\mathbf{g}$ ) nel caso di trasduzioni structure-to-structure o un vettore di reali a dimensione fissa nel caso di trasduzioni structure-to-element.

### 1.3.2 Graph Echo State Networks

Le *Graph Echo State Network* (GraphESN) [8] rappresentano un'estensione delle ESN (si veda il paragrafo 1.2.1) — e delle TreeESN [9] — per il trattamento del dominio dei grafi.

In maniera simile ad una ESN, in una GraphESN sono distinguibili tre layer: uno di *input*, uno nascosto detto *reservoir* e formato da unità ricorsive e non lineari ed infine un *readout* feedforward. Anche in questo caso è richiesto che la funzione di transizione di stato del reservoir sia contrattiva. Il reservoir viene dunque inizializzato affinché rispetti tale vincolo e rimane poi inalterato, mentre il solo readout viene allenato.

È importante sottolineare come la contrattività della funzione di transizione di stato assuma in questo caso un significato specifico di grossa rilevanza, ampliando la classe delle strutture supportate dal modello. La contrattività garantisce infatti la stabilità del processo di codifica anche nel caso di dipendenze cicliche fra le variabili di stato (si veda il paragrafo 1.3), determinando l'applicabilità del modello ad una classe di strutture che comprende grafi non diretti e/o ciclici.

Passiamo ora a caratterizzare più formalmente una GraphESN come modello per l'apprendimento di *trasduzioni strutturali* su un dominio di grafi.

Riprendendo quanto descritto in precedenza (si veda il paragrafo 1.3.1), una trasduzione strutturale (1.8) può essere efficacemente decomposta come

$$\mathcal{T} = \mathcal{T}_{\text{out}} \circ \mathcal{T}_{\text{enc}} \quad (1.9)$$

dove  $\mathcal{T}_{\text{enc}} : (\mathbb{R}^{N_U})^\# \rightarrow (\mathbb{R}^{N_R})^\#$  è una *funzione di encoding*, che mappa l'input

in un dominio strutturato di features, e  $\mathcal{T}_{\text{out}} : (\mathbb{R}^{N_R})^\# \rightarrow (\mathbb{R}^{N_Y})^\#$  la *funzione di output*.

Secondo l'approccio neurale — ricorsivo e ricorrente — la funzione di encoding  $\mathcal{T}_{\text{enc}}$  viene realizzata da un sistema dinamico; lo spazio delle features  $(\mathbb{R}^{N_R})^\#$  è composto da variabili di stato, associate ad ogni vertice dell'input. Chiamiamo  $\mathbf{x}(v) \in \mathbb{R}^{N_R}$  l'informazione di stato associata al vertice  $v$  del grafo in input  $\mathbf{g}$ . Indichiamo inoltre con  $\mathbf{x}(\mathbf{g}) \in \mathbb{R}^{|V(\mathbf{g})|N_R}$  la concatenazione degli stati associati a tutti i vertici dell'input, secondo un ordine arbitrario, e con  $\mathbf{x}(\mathcal{N}(v)) \in \mathbb{R}^{|\mathcal{N}(v)|N_R}$  la concatenazione degli stati associati ai vicini di un vertice  $v$ . La funzione  $\mathcal{T}_{\text{enc}}$  associa dunque ad ogni vertice dell'input  $v \in V(\mathbf{g})$  una corrispondente informazione di stato secondo una *funzione locale di encoding*  $\tau$

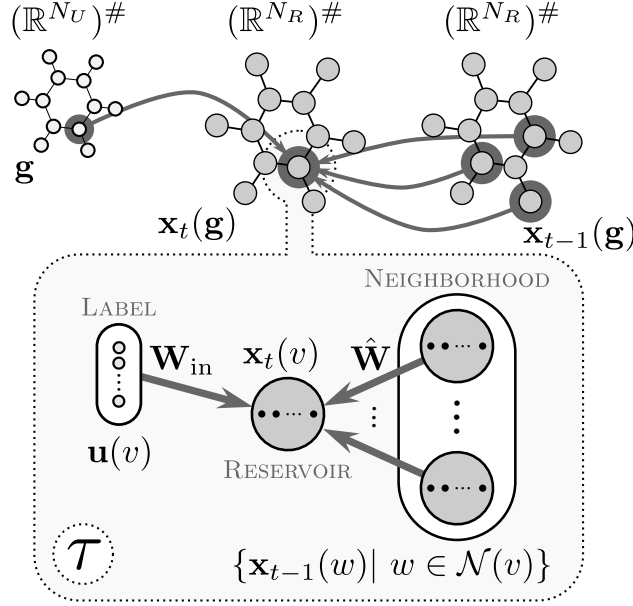
$$\mathbf{x}(v) = \tau(\mathbf{u}(v), \mathcal{N}(v)) \quad (1.10)$$

che esprime la dipendenza dello stato associato al vertice  $v$  da quelli associati ai suoi vicini. Equivalentemente possiamo dire che l'applicazione dell'equazione (1.10) ad ogni vertice dell'input realizza la *funzione globale di encoding*  $\hat{\tau}$

$$\mathbf{x}(\mathbf{g}) = \hat{\tau}(\mathbf{g}, \mathbf{x}(\mathbf{g})) \quad (1.11)$$

Dato un grafo in input  $\mathbf{g}$ , dunque, l'output della funzione di encoding  $\mathcal{T}_{\text{enc}}$  corrisponde alla soluzione dell'equazione (1.11).

In una GraphESN la funzione  $\mathcal{T}_{\text{enc}}$  viene realizzata dal reservoir, che calcola iterativamente una *funzione di transizione locale di stato*, versione iterativa dell'equazione (1.10). Al passo  $t$ , la funzione associa ad ogni vertice  $v$  una



**Figura 1.3:** Schema di applicazione della funzione di transizione locale di stato ad un vertice  $v$  dell'input  $\mathbf{g}$ .

codifica, o valore di stato,  $\mathbf{x}_t(v)$  secondo la seguente equazione

$$\mathbf{x}_t(v) = \tau(\mathbf{u}(v), \mathbf{x}_{t-1}(\mathcal{N}(v))) = f(\mathbf{W}_{\text{in}}\mathbf{u}(v) + \sum_{w \in \mathcal{N}(v)} \hat{\mathbf{W}}\mathbf{x}_{t-1}(w)) \quad (1.12)$$

dove  $\mathbf{W}_{\text{in}} \in \mathbb{R}^{N_R \times (N_U + 1)}$  è la matrice dei pesi sulle connessioni tra input e reservoir,  $\hat{\mathbf{W}} \in \mathbb{R}^{N_R \times N_R}$  è la matrice dei pesi ricorrenti tra i vicini di un vertice ed  $f$  è la funzione di attivazione, tipicamente sigmoidale, delle unità del reservoir. La figura 1.3 mostra un passo del procedimento iterativo di encoding, evidenziando come lo stato calcolato per ogni vertice  $v$  dipenda dall'etichetta numerica di input corrispondente,  $\mathbf{u}(v)$ , nonché dallo stato in corrispondenza dei vertici vicini calcolato al passo precedente,  $\mathbf{x}(\mathcal{N}(v))$ .

Per poter garantire che l'encoding converga, e quindi che l'equazione (1.11) abbia una soluzione, si ricorre ad un setting contrattivo della funzione  $\tau$ .



Perché  $\tau$  sia *contrattiva* rispetto allo stato, è necessario che valga la seguente condizione:

$$\exists C \in [0, 1) \text{ tale che } \forall \mathbf{u} \in \mathbb{R}^{N_U}, \forall \mathbf{x}_1, \dots, \mathbf{x}_k, \mathbf{x}'_1, \dots, \mathbf{x}'_k \in \mathbb{R}^{N_R} : \\ \|\tau(\mathbf{u}, \mathbf{x}_1, \dots, \mathbf{x}_k) - \tau(\mathbf{u}, \mathbf{x}'_1, \dots, \mathbf{x}'_k)\| \leq C \max_{i=1, \dots, k} \|\mathbf{x}_i - \mathbf{x}'_i\| \quad (1.13)$$

dove  $\|\cdot\|$  è una qualsiasi norma su  $\mathbb{R}^{N_R}$ . Considerando per  $\tau$  l'implementazione dell'equazione (1.12), con  $f = \tanh$  usata come funzione di attivazione delle unità del reservoir, e scegliendo la distanza euclidea come norma, risulta che la contrattività è garantita per

riferimento  
al draft

$$\sigma = \|\hat{\mathbf{W}}\|_2 k < 1 \quad (1.14)$$

dove  $k$  è il grado massimo calcolato su tutti i grafi in  $\mathcal{G}$  e  $\sigma$ , che controlla il grado di contrattività delle dinamiche del reservoir, è chiamato *coefficiente di contrazione*. Una volta che la contrattività di  $\tau$  sia assicurata, la convergenza del calcolo iterativo del reservoir è garantita dal *Principio di Contrazione di Banach* [25] per ogni stato iniziale  $\mathbf{x}_0(\mathbf{g})$ ; nella pratica questo permette di calcolare la codifica di un grafo in maniera iterativa, interrompendo l'elaborazione una volta che la distanza tra due stati successivi  $\mathbf{x}_t(\mathbf{g})$  e  $\mathbf{x}_{t+1}(\mathbf{g})$  sia inferiore ad una soglia prefissata  $\epsilon$ . L'algoritmo 1 nella pagina successiva mostra il processo di codifica ricorsivo implementato dal reservoir di una GraphESN.

È opportuno sottolineare come la contrattività della funzione di transizione di stato caratterizza le GraphESN sotto diversi importanti aspetti, che vanno

---

**Algoritmo 1** GraphESN: algoritmo iterativo di encoding.

---

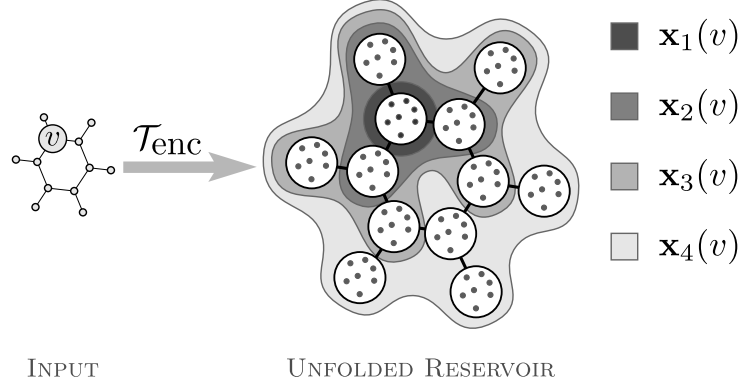
```

for all  $\mathbf{g} \in \mathcal{G}$  do
   $t = 0$ 
  for all  $v \in V(\mathbf{g})$  do
     $\mathbf{x}_0(v) = 0$ 
  end for
  repeat
     $t = t + 1$ 
    for all  $v \in V(\mathbf{g})$  do
       $\mathbf{x}_t(v) = \tau(\mathbf{u}(v), \mathbf{x}_{t-1}(\mathcal{N}(v)))$ 
    end for
  until  $\forall v \in V(\mathbf{g}) : \|\mathbf{x}_t(v) - \mathbf{x}_{t-1}(v)\|_2 \leq \epsilon$ 
end for
return  $\mathbf{x}(\mathbf{g})$ 

```

---

oltre quello algoritmico. Innanzi tutto la stabilità del processo di codifica rende le GraphESN in grado di gestire grafi ciclici, il che rappresenta invece un problema per le reti ricorrenti classiche. Inoltre la contrattività implica la echo state property (si veda il paragrafo 1.2.1), garantendo che gli stati calcolati dal reservoir dipendano (asintoticamente) unicamente dal grafo in input e non dallo stato iniziale. Infine il fatto che  $\tau$  sia contrattiva determina, come nel caso delle ESN, una caratterizzazione Markoviana delle dinamiche del reservoir, con il concetto di *suffisso comune* esteso, rispetto al caso di sequenze [10] o alberi [9], al concetto di *vicinato comune* [8]. La figura 1.4 nella pagina seguente mostra quali siano i vertici che contribuiscono al calcolo dello stato corrispondente ad un vertice  $v$ , evidenziando come nel corso di ogni iterazione la “frontiera” dei vertici coinvolti si allarghi. È bene sottolineare come, nella sua schematizzazione, la figura non renda conto del fatto che uno stesso vertice possa contribuire al calcolo dello stato di  $v$  sia direttamente (i.e. il vertice appartiene a  $\mathcal{N}(v)$ ) sia in maniera indiretta (i.e. per ogni cammino



**Figura 1.4:** Vertici coinvolti nel calcolo di  $\mathbf{x}_t(v)$  durante quattro passi del processo di encoding.

che partendo da  $v$  si estenda, vicino dopo vicino ed iterazione dopo iterazione, fino al vertice in questione).

Nel caso di trasduzioni structure-to-element (si veda il paragrafo 1.3.1) si ricorre ad un'ulteriore funzione  $\mathcal{X} : (\mathbb{R}^{N_R})^\# \rightarrow \mathbb{R}^{N_R}$ , detta *state mapping function*, che applicata al risultato dell'encoding nello spazio strutturato di features restituisce una rappresentazione in uno spazio vettoriale a dimensione fissa per l'intero grafo. In questo caso l'equazione (1.9) si può dunque estendere come

$$\mathcal{T} = \mathcal{T}_{\text{out}} \circ \mathcal{X} \circ \mathcal{T}_{\text{enc}} \quad (1.15)$$

Benché la scelta della funzione  $\mathcal{X}$  sia arbitraria, si distinguono principalmente due alternative. Con *supersource state mapping* [9], lo stato dell'intero grafo  $\mathcal{X}(\mathbf{x}(\mathbf{g}))$  viene mappato nello stato di un solo vertice *supersource*, qualora per la natura dei dati o del problema sia individuabile un vertice che dipenda da tutti gli altri<sup>6</sup>. In alternativa, con *mean state mapping*,  $\mathcal{X}(\mathbf{x}(\mathbf{g}))$  viene

<sup>6</sup>Questa caratteristica, naturale nel caso di alberi radicati e sequenze, è comunque realizzabile “artificialmente” per qualsiasi grafo aggiungendo un vertice che sia collegato a tutti gli altri.

calcolato come la media degli stati corrispondenti ai vertici di  $\mathbf{g}$ :

$$\mathcal{X}(\mathbf{x}(\mathbf{g})) = \frac{1}{|V(\mathbf{g})|} \sum_{v \in V(\mathbf{g})} \mathbf{x}(v) \quad (1.16)$$

Come in una ESN, la funzione di output  $\mathcal{T}_{\text{out}}$  è realizzata da un *readout* formato da  $N_Y$  unità che ricevono input dal reservoir (o dalla state mapping function).

Nel caso di trasduzioni structure-to-structure, la *funzione di output locale* applicata agli stati corrispondenti ai singoli vertici è la seguente:

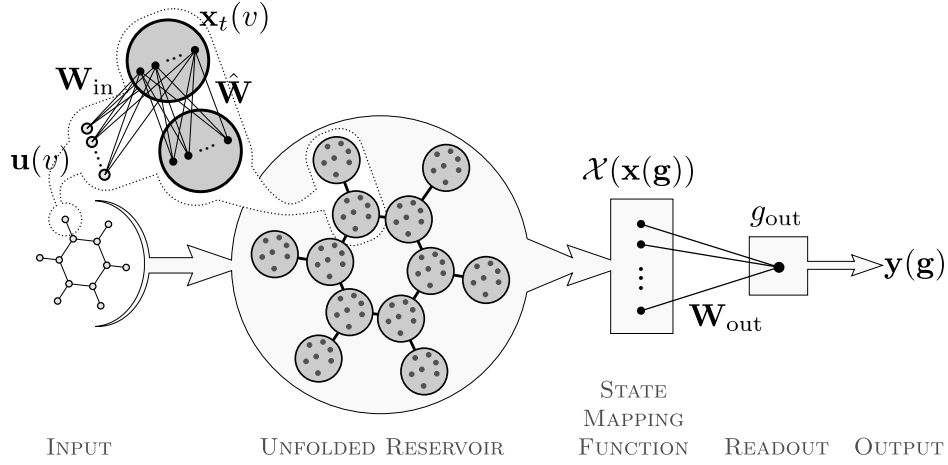
$$\mathbf{y}(v) = g(\mathbf{x}(v)) = g_{\text{out}}(\mathbf{W}_{\text{out}}\mathbf{x}(v)) \quad (1.17)$$

dove  $\mathbf{y}(v) \in \mathbb{R}^{N_Y}$  è il vettore dei valori di output per il vertice  $v$ ,  $\mathbf{W}_{\text{out}} \in \mathbb{R}^{N_Y \times (N_R+1)}$  è la matrice dei pesi delle connessioni tra reservoir e readout e  $g_{\text{out}}$  è la funzione di attivazione delle unità di output, tipicamente lineare. L'applicazione di  $g$  all'encoding ottenuto per ogni vertice dell'input definisce il calcolo della funzione di output  $\mathcal{T}_{\text{out}}$ .

Per trasduzioni structure-to-element, l'output a dimensione fissa relativo all'intero grafo,  $\mathbf{y}(\mathbf{g}) \in \mathbb{R}^{N_Y}$ , è invece ottenuto applicando la funzione di output locale al risultato della state mapping function:

$$\mathbf{y}(\mathbf{g}) = g(\mathcal{X}(\mathbf{x}(\mathbf{g}))) = g_{\text{out}}(\mathbf{W}_{\text{out}}\mathcal{X}(\mathbf{x}(\mathbf{g}))) \quad (1.18)$$

In entrambi i casi, ed in maniera simile a quanto accade in una ESN standard, l'allenamento di una GraphESN prevede l'adattamento dei pesi della matrice



**Figura 1.5:** Schema di una GraphESN per l'apprendimento di trasduzioni structure-to-element.

$\mathbf{W}_{\text{out}}$ , realizzabile attraverso regressione lineare<sup>7</sup> sulla base dei valori di target  $\mathbf{y}_{\text{target}}(v)$ , o  $\mathbf{y}_{\text{target}}(\mathbf{g})$  nel caso di trasduzioni structure-to-element.

La figura 1.5 riassume schematicamente la struttura ed il funzionamento complessivo di una GraphESN che modelli traduzioni structure-to-element. Da sinistra verso destra: l'input  $\mathbf{g}$  viene codificato dal reservoir, che lo mappa in uno spazio delle features strutturato; in seguito l'encoding ottenuto viene mappato, dalla state mapping function  $\mathcal{X}$ , in un vettore di features a dimensione fissa; il risultato dell'applicazione della state mapping function viene infine usato come input del readout, unica componente adattiva del modello, ottenendo in output il vettore a dimensione fissa  $\mathbf{y}(\mathbf{g})$ . In alto a sinistra è invece mostrata la dipendenza che lega uno stato nello spazio delle features, sia all'etichetta di input del vertice corrispondente che agli stati calcolati in corrispondenza dei suoi vicini (i.e. un unico vicino nella figura).

<sup>7</sup>Purché  $g_{\text{out}}$  sia invertibile, è possibile ricondurre  $g$  ad una combinazione lineare semplicemente modificando i valori target:  $\mathbf{y}'_{\text{target}} = g_{\text{out}}^{-1}(\mathbf{y}_{\text{target}})$ .

Il capitolo descrive i modelli realizzati, definendone sia le caratteristiche strutturali e topologiche che quelle computazionali.

Il paragrafo 2.1 fornisce un'introduzione ai modelli proposti, definendone i principi di base in relazione a quanto descritto nel capitolo 1.

Il paragrafo 2.2 introduce e discute i dettagli dell'estensione delle Graph-ESN al caso costruttivo.

Nel paragrafo 2.3 vengono descritti nello specifico i modelli oggetto della sperimentazione svolta, definendone le caratteristiche strutturali.

Infine, nel paragrafo 2.4, viene presentata un'analisi del costo computazionale relativo all'uso dei modelli proposti.

## **2.1 Introduzione ai modelli**

Tecniche, paradigmi e modelli descritti nel capitolo 1, benché appartenenti a periodi ed ambiti diversi, contribuiscono a formare i tratti distintivi dei modelli che saranno discussi nel corso del capitolo. Nel seguito saranno infatti introdotti dei modelli neurali, appartenenti all'ambito del Reservoir Compu-

ting, che permettano di adottare una strategia costruttiva nel trattamento di dati strutturati.

Prima ancora di introdurre i modelli originali, oggetto del lavoro svolto, è quindi possibile identificarne alcune caratteristiche di base, rifacendosi a quanto già descritto in precedenza, in modo da meglio comprenderne i tratti distintivi e, soprattutto, le motivazioni.

L'adozione di un approccio costruttivo, innovativo nell'ambito del Reservoir Computing, ha impatto sui modelli dal punto di vista computazionale ed algoritmico: permette la suddivisione di un task in più sotto-task, l'impiego di singole unità computazionali poco complesse e di algoritmi di apprendimento poco onerosi (si veda il paragrafo 1.1). Applicata al caso del trattamento dei domini strutturati, inoltre, la strategia costruttiva offre l'opportunità di sfruttare localmente delle informazioni contestuali e supervisionate (si veda il paragrafo 1.3), tramite gli output-feedback, che possono essere utilizzate per influenzare le dinamiche del reservoir.

Come appartenenti all'ambito del Reservoir Computing, inoltre, i modelli sperimentati fanno affidamento su una fase di learning estremamente vantaggiosa dal punto di vista computazionale, che interessa solo una parte delle connessioni della rete (si veda il capitolo 1.2).

Infine, essendo estensioni delle GraphESN, i modelli proposti risultano in grado di apprendere trasduzioni strutturali generiche, che abbiano come input anche grafi non diretti o grafi diretti che presentino dei cicli (si veda il paragrafo 1.3.2).

## 2.2 GraphESN costruttive

Stando anche a quanto descritto in precedenza (si veda il paragrafo 1.1 a pagina 2), l'approccio costruttivo prevede l'uso di singole unità computazionali, o sotto-reti, che interconnesse fra loro formano una rete nella sua totalità. Prima di poter descrivere i modelli sviluppati e definirne la topologia è dunque necessario dare una formulazione esatta di quali siano le unità di cui sono composti.

Come avviene in generale per i modelli costruttivi, consideriamo il caso in cui ogni nuova unità computazionale prenda in input gli output delle sotto-reti precedenti. Ogni sotto-rete agisce quindi come una sorta di feature-detector adattivo e la sua uscita può essere trattata come un input aggiuntivo dalle sotto-reti seguenti. Per far sì che più reti possano essere interconnesse fra loro, estendiamo reservoir e readout di una GraphESN standard in modo da rendere possibile la presenza di nuove connessioni (i.e. gli output di altre sotto-reti) che chiamiamo *output-feedback*, provenienti da altre unità computazionali che operano indipendentemente da quella considerata.

Nel caso di trasduzioni structure-to-structure, gli output-feedback rappresentano valori calcolati in corrispondenza di ogni vertice. Il reservoir esteso della sotto-rete  $i$ -esima calcola, per un grafo in input  $\mathbf{g}$  ed al passo  $t$  del processo di codifica, un valore di stato  $\mathbf{x}_t^{(i)}(v) \in \mathbb{R}^{N_R^{(i)}}$  per ogni vertice  $v \in V(\mathbf{g})$  secondo la seguente equazione

$$\begin{aligned} \mathbf{x}_t^{(i)}(v) &= \tau(\mathbf{u}(v), \mathbf{x}_{t-1}^{(i)}(\mathcal{N}(v)), \mathbf{z}^{(1)}(v), \dots, \mathbf{z}^{(i-1)}(v)) \\ &= f(\mathbf{W}_{\text{in}}^{(i)} \mathbf{u}(v) + \sum_{w \in \mathcal{N}(v)} \hat{\mathbf{W}}^{(i)} \mathbf{x}_{t-1}^{(i)}(w) + \sum_{j=1}^{i-1} \mathbf{W}_{\text{fof}}^{(ij)} \mathbf{z}^{(j)}(v)) \end{aligned} \quad (2.1)$$



dove  $\mathbf{z}^{(j)}(v) \in \mathbb{R}^{N_Z^{(j)}}$  rappresenta l'uscita della sotto-rete  $j$ -esima e  $\mathbf{W}_{\text{fof}}^{(ij)} \in \mathbb{R}^{N_R^{(i)} \times N_Z^{(j)}}$  è la matrice dei pesi per le connessioni tra il reservoir  $i$ -esimo e l'uscita della  $j$ -esima sotto-rete.

Nel caso di task structure-to-element cambia la natura degli output delle singole sotto-reti e l'equazione di transizione di stato del reservoir viene modificata di conseguenza come

$$\begin{aligned} \mathbf{x}_t^{(i)}(v) &= \tau(\mathbf{u}(v), \mathbf{x}_{t-1}^{(i)}(\mathcal{N}(v)), \mathbf{z}^{(1)}(\mathbf{g}), \dots, \mathbf{z}^{(i-1)}(\mathbf{g})) \\ &= f(\mathbf{W}_{\text{in}}^{(i)} \mathbf{u}(v) + \sum_{w \in \mathcal{N}(v)} \hat{\mathbf{W}}^{(i)} \mathbf{x}_{t-1}^{(i)}(w) + \sum_{j=1}^{i-1} \mathbf{W}_{\text{fof}}^{(ij)} \mathbf{z}^{(j)}(\mathbf{g})) \end{aligned} \quad (2.2)$$

con  $\mathbf{z}^{(j)}(\mathbf{g}) \in \mathbb{R}^{N_Z^{(j)}}$ .

Al netto della modifica effettuata, il processo di codifica dell'input da parte del reservoir rimane inalterato in entrambi i casi (si veda il paragrafo 1.3.2 e l'algoritmo 1 a pagina 23).

Il readout delle sotto-reti viene invece modificato considerando gli output-feedback come parte della codifica del grafo in input. In altri termini, la codifica viene arricchita da nuove informazioni o features che, calcolate dalle precedenti sotto-reti, vengono concatenate al vettore risultato del processo di encoding del reservoir.

Nel caso di trasduzioni structure-to-structure, l'uscita  $\mathbf{z}^{(i)}(v) \in \mathbb{R}^{N_Z^{(i)}}$  della sotto-rete  $i$ -esima è quindi data da

$$\begin{aligned} \mathbf{z}^{(i)}(v) &= g(\mathbf{x}^{(i)}(v), \mathbf{z}^{(1)}(v), \dots, \mathbf{z}^{(i-1)}(v)) \\ &= g_{\text{out}}(\mathbf{W}_{\text{out}}^{(i)} [\mathbf{x}^{(i)}(v), \mathbf{z}^{(1)}(v), \dots, \mathbf{z}^{(i-1)}(v)]) \end{aligned} \quad (2.3)$$

con  $[\mathbf{x}^{(i)}(v), \mathbf{z}^{(1)}(v), \dots, \mathbf{z}^{(i-1)}(v)]$  concatenazione della codifica corrispondente al vertice  $v$  e degli output-feedback e con la matrice dei pesi del readout opportunamente modificata per adattarsi alla nuova dimensione dello spazio delle features,  $\mathbf{W}_{\text{out}}^{(i)} \in \mathbb{R}^{N_Z^{(i)} \times (N_R^{(i)} + \sum_{j=1}^{i-1} N_Z^{(j)} + 1)}$ .

Per le trasduzioni structure-to-element si procede in maniera simile, calcolando l'output relativo all'intero grafo  $\mathbf{z}^{(i)}(\mathbf{g})$  come

$$\begin{aligned} \mathbf{z}^{(i)}(\mathbf{g}) &= g(\mathcal{X}(\mathbf{x}^{(i)}(\mathbf{g})), \mathbf{z}^{(1)}(\mathbf{g}), \dots, \mathbf{z}^{(i-1)}(\mathbf{g})) \\ &= g_{\text{out}}(\mathbf{W}_{\text{out}}^{(i)} [\mathcal{X}(\mathbf{x}^{(i)}(\mathbf{g})), \mathbf{z}^{(1)}(\mathbf{g}), \dots, \mathbf{z}^{(i-1)}(\mathbf{g})]) \end{aligned} \quad (2.4)$$

Poiché la natura del readout rimane sostanzialmente invariata rispetto al caso delle GraphESN, non sono necessari particolari accorgimenti per la fase di learning, che può essere eseguita con le tecniche standard.

È importante sottolineare alcuni aspetti relativi alla formulazione data delle singole unità computazionali.

- La presenza degli output-feedback è opzionale (e.g.  $\mathbf{W}_{\text{fof}}^{(ij)} = \mathbf{0}$ ). Questo fornisce la possibilità di implementare diverse varianti architetturali (si vedano i paragrafi 2.2.1 e 2.3) e permette di caratterizzare le singole unità computazionali come vere e proprie estensioni delle GraphESN standard.
- Non è stato posto alcun vincolo particolare sulle caratteristiche di una singola unità computazionale in relazione al fatto che più sotto-reti debbano interagire fra loro. Benché ognuna sia predisposta per ricevere gli output-feedback dalle precedenti è dunque possibile che ciascuna sotto-rete differisca dalle altre nelle dimensioni del reservoir,

del readout o dell'output. Questa caratteristica consente quindi l'impiego di sotto-reti eterogenee sia nelle dimensioni che nel sotto-task affrontato.

- I pesi dei singoli output-feedback sono tutti indipendenti l'uno dall'altro. Ne risulta un'ampia gamma di possibilità nell'implementare politiche di feedback specifiche, ad esempio facendo in modo che alcune sotto-reti abbiano maggiore influenza sulle altre (i.e. abbiano pesi maggiori sulle connessioni di output-feedback).
- Gli output-feedback, agendo come input ausiliari, non modificano le condizioni per la contrattività del reservoir.

Nel seguito verrà descritto il funzionamento della rete nella sua totalità e sarà discusso in dettaglio il ruolo degli output-feedback.

### 2.2.1 Costruzione di una rete

Dopo aver definito come sono fatte le singole unità che formano la rete nel suo complesso, è possibile indicare quale sia l'approccio usato per combinarle.

Consideriamo una rete come formata da un *readout globale* al quale vengono collegate in ingresso le connessioni corrispondenti alle uscite di una o più sotto-reti, aggiunte incrementalmente. Al passo  $i$ -esimo, ovvero dopo aver aggiunto  $i$  sotto-reti, l'output della rete è dunque dato, nel caso di trasduzioni structure-to-structure, da:

$$\mathbf{y}^{(i)}(v) = g_{\text{out}}(\mathbf{W}_{\text{out}} [\mathbf{z}^{(1)}(v), \dots, \mathbf{z}^{(i)}(v)]) \quad (2.5)$$

con  $\mathbf{W}_{\text{out}} \in \mathbb{R}^{N_Y \times (\sum_{j=1}^i N_Z^{(j)} + 1)}$  matrice dei pesi delle connessioni del readout globale. Nel caso dell'apprendimento di trasduzioni structure-to-element l'output è calcolato, in maniera simile, come

$$\mathbf{y}^{(i)}(\mathbf{g}) = g_{\text{out}}(\mathbf{W}_{\text{out}} [\mathbf{z}^{(1)}(\mathbf{g}), \dots, \mathbf{z}^{(i)}(\mathbf{g})]) \quad (2.6)$$

In entrambi i casi siamo in presenza di un unico livello di connessioni che può essere allenato attraverso tecniche efficienti, in analogia con quanto avviene nelle sotto-reti ed in accordo con l'approccio del Reservoir Computing.

Internamente alla rete, ogni nuova sotto-rete può essere collegata a tutte le altre sotto-reti esistenti: gli output-feedback possono andare al reservoir della nuova sotto-rete, al suo readout o ad entrambi, in accordo con quanto descritto nel paragrafo 2.2. Il modo in cui le sotto-reti sono connesse fra loro determina la differenza fra i modelli sperimentati, le cui topologie verranno descritte in seguito (si veda il paragrafo 2.3 a pagina 39).

Ogni sotto-rete viene allenata per *correggere l'errore* commesso dalla rete (i.e. dal readout globale). Riferendoci al caso di un task structure-to-element<sup>1</sup>, indichiamo con  $\mathbf{e}^{(i)}(\mathbf{g}) \in \mathbb{R}^{N_Y}$  l'errore commesso dalla rete al passo  $i$ -esimo (i.e. dopo aver aggiunto  $i$  sotto-reti) sull'input  $\mathbf{g}$ . La sotto-rete  $(i+1)$ -esima utilizza dunque  $\mathbf{e}^{(i)}(\mathbf{g})$  come target nella propria fase di apprendimento. Una volta effettuato il training, la sotto-rete viene “congelata” ed aggiunta alla rete: il suo output diventa un nuovo input per il readout globale e può, eventualmente, essere usato come output-feedback per le sotto-reti successive. Ogni volta che una nuova unità computazionale allenata viene aggiunta alla rete si esegue un

---

<sup>1</sup>Il passaggio al caso di task structure-to-structure è banale e viene quindi tralasciato.

**Algoritmo 2** GraphESN costruttiva. Caso structure-to-element.

---


```

init sub-network counter:  $i = 0$ 
init error:  $\mathbf{e}^{(0)}(\mathbf{g}) = -\mathbf{y}_{\text{target}}(\mathbf{g})$ 
repeat
  for all  $1 \leq j \leq \text{pool\_size}$  do
    init  $j$ -th candidate
    connect candidate's output-feedback
    train candidate using  $\mathbf{e}^{(i)}(\mathbf{g})$  as target
  end for
  select the winning candidate
  update sub-network counter:  $i = i + 1$ 
  connect the winner (i.e.  $i$ -th sub-network) to the global readout
  train the global readout
  update error  $\mathbf{e}^{(i)}(\mathbf{g})$ 
until stop adding sub-networks

```

---

nuovo allenamento del readout globale, necessario a valutare se interrompere l'algoritmo o, in alternativa, a determinare i nuovi errori da correggere tramite ulteriori sotto-reti. L'algoritmo procede dunque iterativamente aggiungendo nuove sotto-reti e modificando l'errore, finché non venga verificato un criterio di stop prefissato (e.g. l'errore di training non scenda al di sotto di una soglia prefissata).

L'algoritmo 2 riassume il  cedimento iterativo di costruzione della rete, mostrando anche come la fase di training delle sotto-reti possa avvalersi di un pool di candidate come avviene per la Cascade Correlation (si veda il paragrafo 1.1.1). In questo caso il pool di candidate può contenere sotto-reti che si differenziano, oltre che per l'esito dell'apprendimento, anche nell'iperparametrizzazione o nell'inizializzazione dei pesi, in particolare dei pesi del reservoir che non vengono modificati dal training.

Poiché al primo passo non è disponibile alcuna informazione sull'errore commesso, la prima sotto-rete viene allenata considerando l'errore massimo:

$\mathbf{e}^{(0)}(\mathbf{g}) = -\mathbf{y}_{\text{target}}(\mathbf{g}), \forall \mathbf{g} \in \mathcal{G}$ . Tale scelta risponde ad un'esigenza ben precisa. Il segnale appreso da una sotto-rete gioca infatti un ruolo ambivalente rispetto alla risoluzione del task: da una parte serve al readout globale come indicazione degli errori da correggere, dall'altra sposta gli stati delle sotto-reti successive in modo da raggruppare i pattern potenzialmente già corretti (si veda il paragrafo 2.2.2). In quest'ottica è quindi importante che la prima sotto-rete agisca in maniera analoga alle successive, seppur riferendosi ad un errore per certi versi artificiale.

È infine opportuno sottolineare che la strategia incrementale adottata permette di allenare il readout globale attraverso metodi iterativi (e.g. Least Mean Squares). Tale pratica, comune nell'ambito delle reti neurali, risulta infatti generalmente inapplicabile nel caso del Reservoir Computing a causa dell'alto numero di condizionamento delle matrici di input utilizzate per allenare quei readout che prendano i propri input direttamente da un reservoir.

rifer?

### 2.2.2 Output-feedback

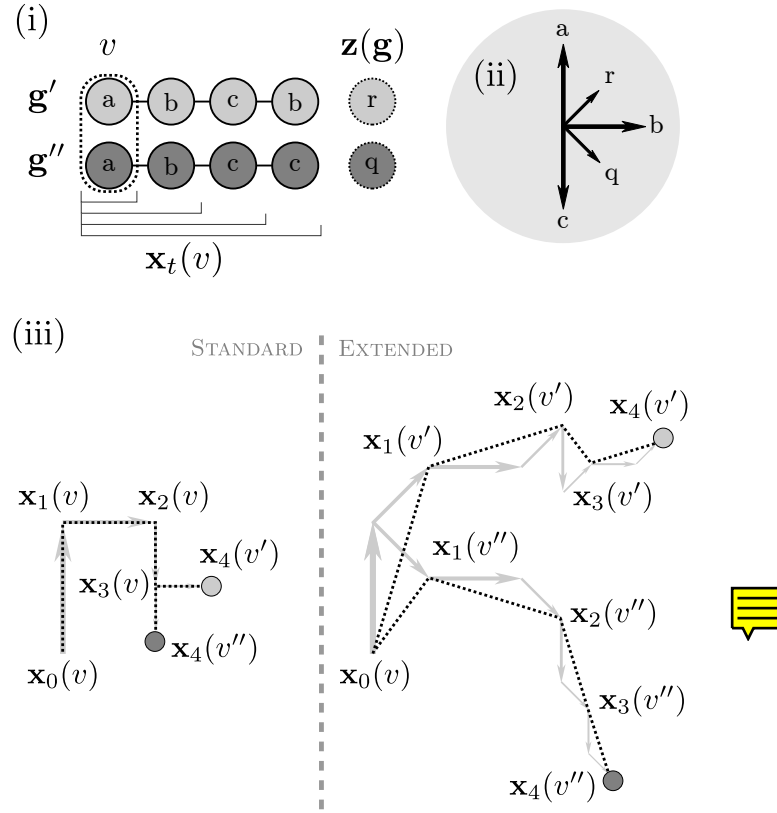
Il ruolo degli output-feedback è di rilevanza centrale per i modelli proposti, in particolare rispetto al modo in cui questi possono influenzare le dinamiche del reservoir. Con riferimento al reservoir esteso dell'equazione (2.2) è infatti importante sottolineare come i valori in  $\mathbf{z}^{(j)}(\mathbf{g})$  introducano *localmente* un'informazione *globale*, riferita all'intero grafo in input<sup>2</sup>, e *supervisionata*, ottenuta cioè tramite il processo di apprendimento (i.e. gli output-feedback

<sup>2</sup>Nel caso di un task structure-to-structure, equazione (2.1), è forse più corretto parlare di informazione *contestuale* piuttosto che globale. Tale differenza nella terminologia è tuttavia irrilevante ai fini del discorso affrontato se non addirittura fuorviante se si considera che il processo di encoding di una GraphESN sfrutta di per sé informazioni contestuali, anche se di natura diversa da quelle qui riferite.

corrispondono agli output di altre sotto-reti, precedentemente allenate). Se per la sua natura Markoviana un reservoir ha la capacità di discriminare i vertici in base ai loro vicini (ed i vicini dei vicini, e così via, iterativamente), il reservoir esteso usufruisce quindi anche di un'informazione che gli permette di “vedere oltre” la frontiera che determina lo stato corrente (si veda la figura 1.4 a pagina 24) ed in base a questo di spostare gli stati in maniera consistente con il task affrontato.

L'idea di sfruttare le caratteristiche dell'output per influenzare le dinamiche del reservoir non è d'altra parte originale se si pensa che già nella prima formulazione delle ESN [17] era prevista la presenza di connessioni dal readout verso il reservoir. È tuttavia opportuno sottolineare come i modelli qui proposti possano fare affidamento, grazie all'approccio costruttivo, su uno schema stabile di output-feedback, in netta contrapposizione con gli scenari in cui gli output-feedback siano stati applicati alle ESN standard [22, 43, 30].

Per comprenderne meglio la natura, la figura 2.1 nella pagina successiva mostra un esempio di come gli output-feedback possano modificare le dinamiche della rete. Con riferimento alla figura, si considerino (i) due sequenze in input  $\mathbf{g}'$  e  $\mathbf{g}''$ , che differiscono unicamente nell'ultimo vertice ed a cui sono associati valori di output-feedback distinti, indicati con  $r$  e  $q$  rispettivamente. Supponendo che lo spazio delle feature sia bidimensionale (i.e.  $N_R = 2$ ), è possibile dare un'interpretazione geometrica ai vari input (e.g.  $\mathbf{W}_{\text{in}} \mathbf{a} = (0, 1)$ ), che supponiamo essere come è indicato in (ii). In (iii) sono mostrate le traiettorie corrispondenti all'encoding iterativo del primo vertice della sequenza, indicato con  $v'$  e  $v''$  rispettivamente per i due input o semplicemente con  $v$  quando le traiettorie nello spazio degli stati coincidono per entrambi i casi.



**Figura 2.1:** Effetto degli output-feedback sul reservoir esteso: esempio di encoding in uno spazio bidimensionale.

(i) Input; (ii) interpretazione geometrica degli input; (iii) traiettorie nel feature space, (sx) reservoir standard, (dx) reservoir esteso.

Nel caso di un reservoir standard, a sinistra nella figura, i due input seguono la stessa traiettoria per le prime tre iterazioni. Le due sequenze sono dunque indistinguibili fino al quarto passo, ovvero fino a quando l'insieme dei vertici che determina lo stato  $\mathbf{x}(v)$  è sufficientemente ampio da includere l'ultimo vertice. Si nota inoltre come, per effetto della natura Markoviana del reservoir, gli spostamenti delle variabili di stato si riducano con l'aumentare delle iterazioni di modo che l'ultima iterazione, quella determinante per discriminare  $g'$  e  $g''$  nel reservoir standard, mantenga comunque le due codifiche

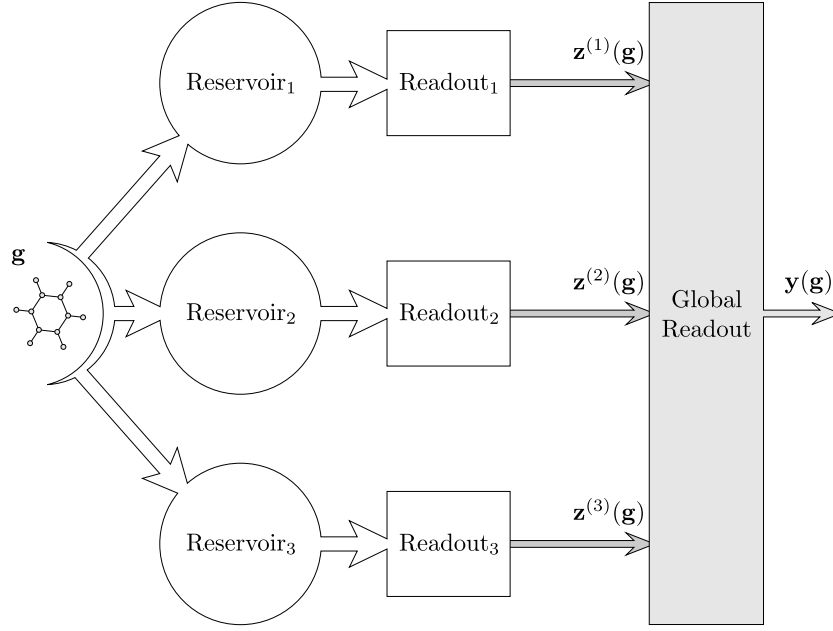




$\mathbf{x}_4(v')$  e  $\mathbf{x}_4(v'')$  molto vicine nello spazio degli stati.

Sulla destra sono invece mostrate le traiettorie nel caso in cui ai due input sia associato un output-feedback differente, processabile ricorrendo ad un reservoir esteso. La figura evidenzia come in questo caso le traiettorie si differenzino sin dalla prima iterazione, permettendo di discriminare le due sequenze ben prima che l'encoding di  $v'$  e  $v''$  arrivi ad includere l'etichetta dell'ultimo vertice e con l'effetto di portare  $\mathbf{x}_4(v')$  e  $\mathbf{x}_4(v'')$  in punti molto più lontani all'interno del feature space rispetto a quanto avverrebbe con un reservoir standard.

Per effetto dello spostamento appena descritto, dunque, i pattern in input tenderanno a raggrupparsi nello spazio degli stati in maniera consistente con i propri valori degli output-feedback. È importante notare che, quando le varie sotto-reti sono allenate per riprodurre un segnale legato all'errore residuo commesso dalla rete (e.g. emulando l'errore o massimizzando la correlazione fra il proprio output e l'errore, si veda il paragrafo 1.1.1 a pagina 4), allora si avrà uno spostamento maggiore in quegli input per cui sia stato precedentemente riconosciuto un alto errore residuo, e che hanno dunque maggiore probabilità di essere stati corretti dalla rete nei passi precedenti (si veda anche il paragrafo 2.2.1). Informalmente potremmo dire che gli output-feedback, provenienti da altre unità computazionali, hanno in questo caso l'effetto di “mettere da parte” gli input i cui errori siano già stati corretti da altre sotto-reti, permettendo dunque al learning di concentrarsi maggiormente su una porzione di pattern per i quali la rete commette gli errori maggiori.



**Figura 2.2:** Modello GraphESN-ZERO. Topologia con tre sotto-reti.

## 2.3 Modelli

Come detto in precedenza, i modelli sperimentati si distinguono nel modo in cui le varie sotto-reti vengono collegate fra loro.

Mantenendo il denominatore comune dell'approccio costruttivo, secondo i principi e le modalità già descritte, i tre modelli proposti si inseriscono in un percorso incrementale che guarda all'arricchimento della struttura nelle connessioni in modo da sfruttare l'introduzione di informazione supervisionata attraverso la presenza degli output-feedback.

### GraphESN-ZERO



Il primo modello sperimentato, chiamato *GraphESN-ZERO*, rappresenta la semplice estensione delle GraphESN al caso costruttivo: le singole sotto-reti

vengono aggiunte incrementalmente ma non vi è alcuna connessione fra i loro output. Il modello è quindi formato da un unico strato di sotto-reti e dal readout globale, senza che sia presente alcun output-feedback.

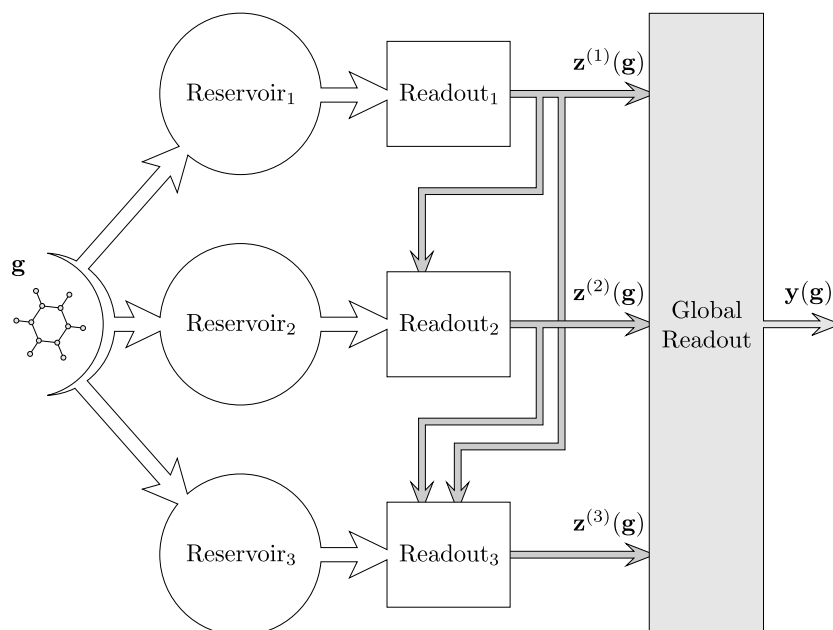
La figura 2.2 nella pagina precedente mostra schematicamente la topologia del modello GraphESN-ZERO.

### GraphESN-FW

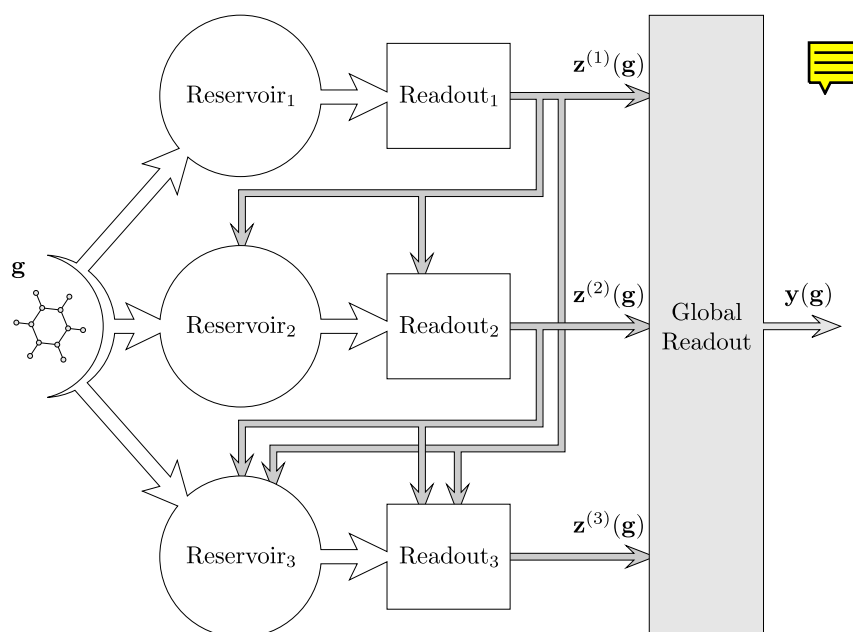
Il secondo modello proposto arricchisce il precedente attraverso una struttura più complessa. In questo caso, infatti, il readout di ogni sotto-rete riceve in input gli output-feedback provenienti dalle sotto-reti precedenti. Il risultato è una struttura in cascata in cui, in maniera simile a quanto accade per la Cascade Correlation (si veda il paragrafo 1.1.1), ogni sotto-rete forma un livello a sé stante dipendente dai precedenti. Poiché gli output-feedback vanno da una sotto-rete a tutte le successive, le connessioni si sviluppano “in avanti”, motivo per cui il modello è chiamato *GraphESN-FW* o *GraphESN-Forward*.

La figura 2.3 nella pagina seguente mostra la topologia del modello GraphESN-FW, con le connessioni in avanti.

Alcune osservazioni sono opportune riguardo alla presenza di output-feedback ed alla particolare scelta di una struttura a cascata in avanti. L'esistenza di output-feedback, infatti, determina implicitamente una relazione di dipendenza fra le varie unità computazionali (i.e. per poter svolgere la propria elaborazione, ogni sotto-rete necessita dell'output delle sotto-reti dalle quali riceve output-feedback). La struttura a cascata in avanti individua dunque un ordinamento parziale che mette il modello nelle condizioni di poter funzionare, determinando anche una strategia di esecuzione specifica (i.e. gli



**Figura 2.3:** Modello GraphESN-FW. Topologia con tre sotto-reti.



**Figura 2.4:** Modello GraphESN-FOF. Topologia con tre sotto-reti.

output vengono calcolati in sequenza, a partire dalla prima sotto-rete in poi). Il fatto che gli output-feedback siano solo in avanti ha anche il vantaggio di mantenere inalterate tutte quelle sotto-reti per cui siano già state svolte delle elaborazioni, con l'effetto pratico di poter salvare i risultati ottenuti (e.g. l'encoding dei singoli input da parte dei reservoir) e risparmiare dunque risorse di calcolo.

### GraphESN-FOF

L'ultimo modello proposto prende il nome di *GraphESN-FOF* (i.e. GraphESN Forward Output-Feedback) e combina l'approccio incrementale, la struttura a cascata in avanti e l'uso degli output-feedback per spostare gli stati del reservoir come discusso nel paragrafo 2.2.2. In questo caso, dunque, le connessioni in avanti sono mandate sia al readout che al reservoir di tutte le sotto-reti successive.

La struttura delle connessioni del modello GraphESN-FOF è mostrata nella figura 2.4.

## 2.4 Costo computazionale



Consideriamo una rete di tipo GraphESN-FOF formata da  $d$  sotto-reti, ognuna con  $N_R$  unità nel reservoir. Assumiamo inoltre, secondo un'ipotesi tutt'altro che irrealistica, che la dimensione dell'output della rete e delle sotto-reti sia trascurabile. Di seguito viene riportato il costo computazionale delle fasi principali che caratterizzano l'evoluzione della rete (si veda l'algoritmo 2 a pagina 34).

## Encoding

Ogni sotto-rete deve far convergere il proprio reservoir per ogni input  $\mathbf{g} \in \mathcal{G}$  e quindi ottenere l'encoding corrispondente nello spazio delle features. Grazie alle caratteristiche dei modelli sperimentati è sufficiente che ogni sotto-rete esegua questa fase una sola volta per ogni input.

Chiamando  $v$  il numero massimo di vertici di un grafo del dataset  $\mathcal{G}$  (i.e.  $v = \max(\{|V(\mathbf{g})|, \mathbf{g} \in \mathcal{G}\})$ ) ed  $i$  il numero massimo di iterazioni consentite per la convergenza di un singolo reservoir, il costo computazionale complessivo del processo di encoding è

$$O(d \, i \, v \, |\mathcal{G}| \, N_R^2) \quad (2.7)$$

Come nel caso delle GraphESN standard questa fase è identica sia per gli input usati nel training della rete che per il test.

## Training

Distinguiamo due distinte fasi di training, in accordo con la strategia costruttiva descritta in precedenza: il training di una singola sotto-rete e l'allenamento del readout-globale. Poiché il training riguarda in ogni caso l'adattamento dei pesi di un unico livello di connessioni, verso il readout, vari algoritmi possono essere efficacemente applicati. Nel seguito saranno considerate solo alcune delle strategie possibili, ed in particolare quelle implementate e sperimentate.

Una sotto-rete può essere allenata per emulare l'errore commesso dal readout-globale. In questo caso, ricorrendo all'algoritmo di Ridge Regression

il costo computazionale complessivo è

$$O(d^4 + N_R d^3 + N_R^2 d^2 + N_R^3 d + d^3 |\mathcal{G}| + d^2 N_R |\mathcal{G}| + d N_R^2 |\mathcal{G}|) \quad (2.8)$$

In alternativa è possibile allenare la sotto-rete affinché il suo output massimizzi la correlazione con l'errore commesso dal readout-globale, in maniera analoga a quanto avviene per l'algoritmo di Cascade Correlation (si veda il paragrafo 1.1.1). Il learning è in questo caso realizzato attraverso un algoritmo iterativo. Chiamando  $q$  il numero massimo di iterazioni necessarie all'algoritmo per convergere, il costo complessivo dell'allenamento delle sotto-reti è

$$O(q d |\mathcal{G}| N_R + q |\mathcal{G}| d^2) \quad (2.9)$$

Il readout-globale della rete può a sua volta essere allenato tramite Ridge Regression, con un costo complessivo di

$$O(d^4 + d^3 |\mathcal{G}|) \quad (2.10)$$

È tuttavia possibile allenare il readout-globale con Least Mean Squares. Sia  $r$  il numero massimo di iterazioni necessarie alla convergenza dell'algoritmo, il costo complessivo è in questo caso

$$O(r |\mathcal{G}| d^2) \quad (2.11)$$

### Calcolo dell'output

Ogni volta che una nuova sotto-rete viene allenata ed aggiunta alla rete è necessario procedere con il calcolo dell'output relativo ad ognuno dei grafi nel dataset  $\mathcal{G}$ . Nell'intero processo di costruzione della rete il costo del calcolo dell'output, a livello delle sotto-reti, risulta

$$O(dN_R|\mathcal{G}| + d^2|\mathcal{G}|) \quad (2.12)$$

È opportuno sottolineare come, per una generica sotto-rete, il calcolo dell'output dipenda dagli output-feedback delle sotto-reti precedenti, i cui valori sono precalcolati, e dal risultato della fase di encoding, anch'esso già calcolato durante la fase di allenamento.

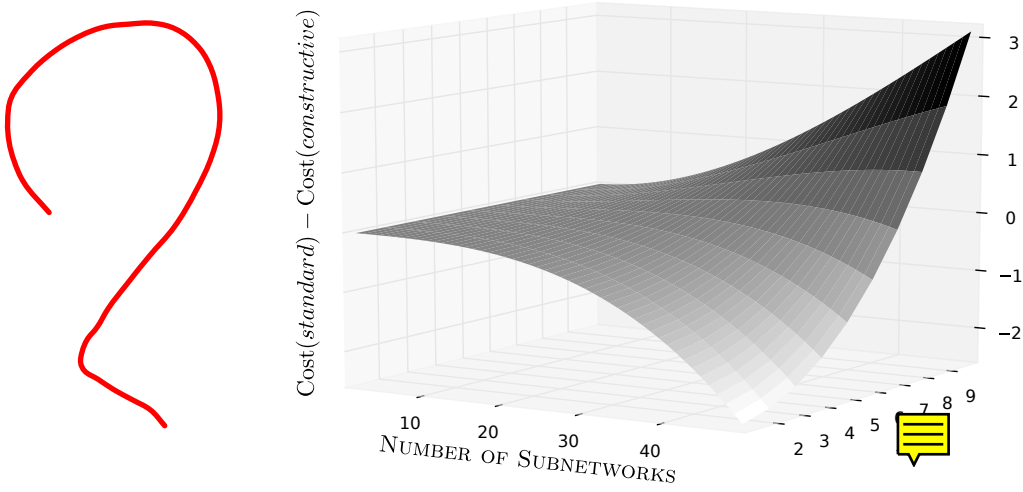
Ad ogni incremento delle dimensioni della rete è inoltre necessario ottenere l'errore commesso dal readout-globale e quindi calcolare l'output della rete. Poiché il readout-globale prende in input unicamente gli output delle singole sotto-reti il costo complessivo è

$$O(d^2|\mathcal{G}|) \quad (2.13)$$

#### 2.4.1 Considerazioni

Il Reservoir Computing in generale si caratterizza per l'efficienza computazionale dei modelli. Per poter efficacemente riflettere sul costo computazionale dei modelli proposti prenderemo dunque come riferimento il confronto con la GraphESN standard (si veda il paragrafo 1.3.2).





**Figura 2.5:** Vantaggio computazionale nell'uso dell'approccio costruttivo al variare di  $N_R$  e del numero di sotto-reti. Assi orizzontali: numero di sotto-reti e dimensioni del reservoir. Asse verticale: differenza fra costo del modello standard e costo del modello costruttivo.

La figura 2.5 dà un'intuizione di quale sia il vantaggio dal punto di vista computazionale derivato dall'adozione dell'approccio costruttivo. Al variare delle dimensioni dei reservoir e del numero delle sotto-reti, la figura mostra come varia la differenza fra il costo di una GraphESN standard equivalente<sup>3</sup> e di una GraphESN-FOF. I valori positivi, riportati in scala logaritmica, corrispondono dunque ai casi in cui l'approccio costruttivo risulta più efficiente. La figura evidenzia come l'approccio costruttivo risulti sconveniente nel caso in cui si usino molte sotto-reti di dimensioni estremamente ridotte. La motivazione risiede nel fatto che in un simile scenario il costo computazionale complessivo è dominato dal processo iterativo di costruzione della rete piuttosto che dalla dimensione dei reservoir trattati. In condizioni più realistiche — il Reservoir Computing in generale si caratterizza per l'utilizzo di reservoir

<sup>3</sup>Per *equivalente* si intende in questo caso una GraphESN con un unico reservoir di dimensioni pari alla somma dei reservoir delle sotto-reti del caso costruttivo.

ricchi di unità — è tuttavia evidente come i modelli costruttivi risultino meno onerosi della GraphESN standard.

In merito al costo computazionale risulta particolarmente importante un'ulteriore osservazione. Benché il confronto con una singola GraphESN lasci intravedere i vantaggi dei modelli proposti è infatti opportuno sottolineare come la strategia costruttiva comporti particolari benefici in termini di selezione del modello (e.g. tramite cross-validation). Eliminando la necessità di dover fissare la dimensione del reservoir a priori, infatti, il costo speso per la costruzione incrementale di una singola rete corrisponde all'allenamento ed il test di svariate reti diverse nel caso delle GraphESN standard.



## Risultati Sperimentali

Il capitolo illustra e discute i risultati sperimentali ottenuti applicando i modelli proposti su task relativi a dataset reali.

Il paragrafo 3.1 descrive i dataset utilizzati ed i corrispondenti task affrontati.

Il paragrafo 3.2 descrive i setting sperimentali e raccoglie i risultati ottenuti.

Il paragrafo 3.3 espone un'analisi critica dei risultati sperimentali.

### 3.1 Dataset

Le potenzialità dei modelli proposti sono state sperimentate su problemi reali appartenenti all'ambito della Chimica.

Pur senza addentrarsi nei dettagli è interessante sottolineare come il legame fra il Machine Learning e la Chimica vada recentemente rafforzandosi, in particolar modo in relazione all'apprendimento di dati su domini strutturati. L'enorme varietà di molecole naturalmente rappresentabili come grafi, i costi sperimentali elevati, l'esistenza di conoscenze non sempre codificabili o prone

a numerose eccezioni, fanno infatti della Chimica il banco di prova ideale per dei modelli in grado di gestire domini strutturati.

È dunque alla sfera della Chemioinformatica che vanno ascritti i task affrontati ed i dataset che verranno descritti nel seguito del paragrafo.

Prima di procedere, è importante sottolineare che nessuna specifica conoscenza pregressa è stata necessaria per preprocessare i dati, modificarne la struttura o estrapolarne informazioni numeriche. In altri termini, le informazioni utilizzate per allenare i modelli sono esattamente quelle riportate nei dataset: diversamente da quanto accade ricorrendo ad altri approcci, nessuna particolare assunzione riguardante il dominio trattato è stata fatta (e.g. la scelta di una metrica nell'uso di un kernel) né è stata praticata alcuna forma di feature-selection manuale (e.g. la selezione di indici topologici).

#### **PTC**

Il Predictive Toxicology Challenge (PTC) [15] ha fornito i dati per quattro task distinti. Il dataset consta di 417 molecole di cui è riportata la carcinogenicità riferita a diversi tipi di roditori: topi maschi (MM), topi femmine (FM), ratti maschi (MR), ratti femmine (FR).

Ogni molecola è stata rappresentata come un grafo indiretto, con i vertici corrispondenti agli atomi e gli archi ai legami atomici. Il grado massimo riscontrato sull'intero dataset è  $k = 4$ . Il numero di vertici nei grafi in input risulta mediamente 25.7, variando da un minimo di 2 ad un massimo di 109.

L'etichetta numerica associata ad ogni vertice è stata ottenuta concatenando l'encoding binario 1-of- $m$  del simbolo atomico associato all'atomo

corrispondente e la sua carica parziale. La dimensione totale delle etichette di input è dunque 24.

Quattro task di classificazione sono stati definiti per PTC, uno per ogni tipo di roditore [7], assegnando target +1 alle molecole attive e target -1 a quelle inattive.

### Mutagenesis

La seconda serie di task affrontati si riferisce al dataset Mutagenesis [37], contenente 230 molecole nitroaromatiche, con l'obiettivo di classificare correttamente fra queste quelle mutagene. Ogni composto nel dataset è descritto dalla sua struttura atomo-legame (AB), da due valori corrispondenti a misurazioni di proprietà chimiche della molecola (C) e due attributi strutturali precalcolati (PS). Riprendendo quanto fatto in [8], sono state dunque considerate le tre possibili descrizioni: AB, AB+C, AB+C+PS.

Ogni molecola del dataset è stata rappresentata come un grafo indiretto, con vertici ed archi corrispondenti ad atomi e legami rispettivamente, entrambi riportati nella descrizione AB dei singoli composti. Il grado massimo riscontrato sul dataset risulta essere  $k = 4$ . Il numero di vertici per input varia tra un minimo di 13 ad un massimo di 40, con una media di 25.6 vertici per ogni molecola.

L'etichetta numerica associata ad ogni vertice dell'input è stata ricavata codificando il simbolo atomico corrispondente attraverso un encoding binario 1-of- $m$  e poi concatenando la carica parziale dell'atomo, il tipo atomico normalizzato in  $[-1, 1]$  più gli altri valori globali contenuti nella descrizione C e PS, in accordo alla rappresentazione via via adottata. La dimensione

delle etichette di input è dunque di 11, 13 e 15 per le descrizioni AB, AB+C e AB+C+PS rispettivamente.

Il task sul dataset è stato definito come una classificazione binaria, con il target a +1 per i grafi corrispondenti a composti mutageni e  $-1$  altrimenti.

## 3.2 Esperimenti

Sui dataset descritti in precedenza sono stati svolti esperimenti che permettessero di confrontare i tre modelli proposti (si veda il paragrafo 2.3) con le GraphESN (paragrafo 1.3.2).

È bene precisare che, data la flessibilità dell’approccio introdotto, sono molte le diverse strategie applicabili per realizzare un singolo modello fra quelli descritti. Variazioni sono possibili nell’inizializzazione dei pesi, nella scelta delle candidate, nella tipologia di allenamento delle sotto-reti o nel task loro assegnato. Tutte le reti a cui fanno riferimento i risultati riportati nel seguito condividono dunque un setting sperimentale comune, che individua solo un piccolo frammento delle possibili varianti implementative. Prima di presentare i risultati sperimentali è dunque necessario descrivere il setting sperimentale adottato.

Come appartenenti all’ambito del Reservoir Computing, le reti considerate presentano una serie di connessioni i cui pesi non vengono modificati durante il learning: le connessioni input-reservoir e reservoir-reservoir, a cui si aggiungono, nei modelli proposti, anche le connessioni per gli output-feedback.

Nei modelli sperimentati i pesi sulle connessioni di input, contenuti nella matrice  $\mathbf{W}_{\text{in}}^{(i)}$  per la sotto-rete  $i$ -esima, hanno valori random con distribuzione

uniforme nell'intervallo  $[-\lambda_{\text{in}}, \lambda_{\text{in}}]$ , con  $\lambda_{\text{in}}$  parte degli iperparametri del modello. I pesi sulle connessioni del reservoir, matrice  $\hat{\mathbf{W}}^{(i)}$ , sono invece stati inizializzati secondo una distribuzione uniforme in  $[-1, 1]$  e poi ridimensionati per ottenere il coefficiente di contrattività  $\sigma$  desiderato (si veda l'equazione (1.14) a pagina 22).

Alle connessioni di output-feedback, in  $\mathbf{W}_{\text{fof}}^{(ij)}$  per ogni coppia di sotto-reti  $(i, j)$ , sono stati assegnati pesi fissi con valore dato dall'iperparametro  $\lambda_{\text{fof}}$ .

Poiché entrambi i dataset trattati riguardano trasduzioni structure-to-element, i modelli sperimentati ricorrono all'uso di una *state mapping function*,  $\mathcal{X}$ . In tutti i casi è stata utilizzata una funzione *mean state mapping* (si veda l'equazione (1.16) a pagina 25).

Come si evince dall'algoritmo 2 a pagina 34 i modelli costruttivi proposti sono caratterizzati da due distinte fasi di apprendimento per ogni iterazione: una riguardante le singole sotto-reti e l'altra relativa al readout globale.

Nel corso degli esperimenti svolti le sotto-reti sono state allenate attraverso l'algoritmo di Ridge Regression per emulare l'errore residuo commesso dalla rete. Il parametro di regolarizzazione  $\lambda_r$  utilizzato, uguale per ogni sotto-rete, rappresenta uno degli iperparametri dei modelli. La possibilità di usare un algoritmo di apprendimento che non dipendesse dal valore iniziale dei pesi e che non non rischiasse di convergere a minimi o massimi locali ha inoltre fatto propendere per tralasciare l'impiego di un pool di sotto-reti candidate. Il readout globale è invece stato allenato tramite Least Mean Squares (LMS) con l'iperparametro  $\lambda_{\text{wd}}$  a regolare il *weight decay*.

Per tutte le unità, sia nei reservoir che nei readout, la funzione di attivazione utilizzata è la *tangente iperbolica*. L'errore residuo, utilizzato per

riferimento?



riferimento

riferimento

l'allenamento delle sotto-reti, è dunque quello ottenuto dalla differenza fra l'attivazione delle unità di output ed il valore del target

$$\begin{aligned}\mathbf{e}^{(i)}(\mathbf{g}) &= \mathbf{y}^{(i)}(\mathbf{g}) - \mathbf{y}_{\text{target}}(\mathbf{g}) \\ &= \tanh(\mathbf{W}_{\text{out}} [\mathbf{z}^{(1)}(\mathbf{g}), \dots, \mathbf{z}^{(i)}(\mathbf{g})]) - \mathbf{y}_{\text{target}}(\mathbf{g})\end{aligned}\tag{3.1}$$

Ne risulta un errore distribuito nell'intervallo  $(-2, 2)$ . Due fattori influenzano la scelta di adottare tale criterio nel determinare l'errore, e dunque il target delle sotto-reti: da una parte la pratica ha lasciato emergere una minore efficacia dell'uso di errori discreti (e.g.  $\mathbf{e}^{(i)}(\mathbf{g}) \in \{-1, 1\}$  oppure  $\mathbf{e}^{(i)}(\mathbf{g}) \in \{-1, 0, 1\}$ ), dall'altra la necessità di usare un'arcotangente iperbolica,  $\tanh^{-1}$ , per poter applicare la Ridge Regression nelle sotto-reti ha reso indispensabile la presenza di un limite, sia superiore che inferiore, sui valori di errore. Per poter essere efficacemente usato come target per le sotto-reti, dunque, il segnale di errore è stato ad ogni iterazione dimezzato e portato nell'intervallo  $(-1, 1)$ .

Nel corso degli esperimenti svolti le reti sono state fatte crescere finché non si sia verificata una delle seguenti condizioni: l'errore di misclassificazione sul training-set sia risultato inferiore ad una soglia prefissata, specifica per ogni task, oppure la variazione dell'errore di training sia risultata, in due iterazioni successive, inferiore all'1% in valore assoluto.

Per confrontare i modelli introdotti con le GraphESN, sono stati svolti su queste ultime esperimenti che prevedessero l'impiego degli stessi iperparametri. Per ovviare al fatto che in questo caso il numero di unità totali dovesse essere fissato a priori, sono state prese in considerazione diverse GraphESN facendone variare la dimensione del reservoir,  $N_R \in \{50, 100, 200, 500\}$ .



**Tabella 3.1:** Iperparametri usati per la model-selection sui 4 task PTC.

$\lambda_{in}$	$\lambda_{fof}$	$\lambda_r$	$\lambda_{wd}$
$\{1.0, 0.1\}$	$\{1.0, 2.0\}$	$\{0.01, 0.1, 0.2\}$	$\{0.0, 0.01\}$

**Tabella 3.2:** Accuratezza media dei modelli e deviazione standard, in percentuale, sui 4 task PTC.

Model	$N_R$	FR	FM	MR	MM
GraphESN		67.7 ( $\pm 1.4$ )	60.7 ( $\pm 4.2$ )	56.7 ( $\pm 3.6$ )	67.1 ( $\pm 3.3$ )
GraphESN-ZERO	50	67.3 ( $\pm 1.9$ )	62.8 ( $\pm 4.2$ )	57.9 ( $\pm 2.1$ )	65.0 ( $\pm 2.4$ )
GraphESN-ZERO	30	67.2 ( $\pm 1.8$ )	63.2 ( $\pm 4.4$ )	58.1 ( $\pm 1.9$ )	65.8 ( $\pm 2.3$ )
GraphESN-FW	50	67.1 ( $\pm 1.0$ )	63.6 ( $\pm 4.4$ )	58.4 ( $\pm 1.7$ )	65.4 ( $\pm 3.2$ )
GraphESN-FW	30	67.2 ( $\pm 0.4$ )	63.3 ( $\pm 3.8$ )	57.4 ( $\pm 2.1$ )	64.6 ( $\pm 2.7$ )
GraphESN-FOF	50	68.3 ( $\pm 1.5$ )	62.5 ( $\pm 3.6$ )	57.2 ( $\pm 2.1$ )	66.6 ( $\pm 3.0$ )
GraphESN-FOF	30	67.9 ( $\pm 1.3$ )	62.8 ( $\pm 3.5$ )	57.4 ( $\pm 1.6$ )	65.4 ( $\pm 2.3$ )

### 3.2.1 PTC

La valutazione dei modelli sui quattro task PTC (si veda il paragrafo 3.1 a pagina 49) è stata fatta attraverso una doppia *cross-fold validation* con un ciclo esterno di 5 fold ed un ciclo interno di 5 fold. Ogni iperparametrizzazione è stata testata su 5 istanziazioni del modello, in modo da variare i valori dei pesi random assegnati alle connessioni.

La tabella 3.1 mostra in che modo sono stati fatti variare gli iperparametri dei modelli. In tutti i casi si è mantenuto un coefficiente di contrattività prefissato,  $\sigma = 1$ .

I risultati sperimentali ottenuti sono invece indicati nella tabella 3.2, che riporta l'accuratezza media dei modelli e la deviazione standard per ciascun task. Si riscontra come i modelli introdotti risultino in grado, in tre dei quattro casi, di migliorare la performance ottenuta tramite un approccio non costruttivo. In due dei quattro casi esaminati (i.e. FM, FR) il miglioramento

offerto dai modelli costruttivi risulta consistente e sistematico.

Le performance ottenute tramite i modelli proposti risultano inoltre comparabili con quelle raggiunte applicando sugli stessi task dei modelli basati su kernel allo stato dell'arte nel trattamento dei domini strutturati [7].

### 3.2.2 Mutagenesis

Le performance dei modelli sui tre task relativi al dataset Mutagenesis (paragrafo 3.1 a pagina 50) sono state calcolate ricorrendo ad una doppia *cross-fold validation* con un ciclo esterno di 10 fold ed uno interno di 5 sotto-fold. Per avere una stima più attendibile al variare dell'assegnazione iniziale dei pesi, 5 distinte ripetizioni sono state effettuate per ogni iperparametrizzazione testata.

La tabella 3.3 mostra i valori degli iperparametri utilizzati per la selezione dei modelli, mentre nella tabella 3.4 nella pagina seguente sono riportati i risultati sperimentali ottenuti sui tre task.

Benché le performance ottenute dai modelli proposti risultino inferiori rispetto a quelle raggiungibili con l'utilizzo di altri approcci (e.g. GNN [32]), i risultati sperimentali evidenziano come i modelli costruttivi rappresentino una valida alternativa ed un miglioramento rispetto alle GraphESN.

**Tabella 3.3:** Iperparametri usati per la model-selection sui 3 task Mutagenesis.

$\lambda_{\text{in}}$	$\lambda_{\text{fof}}$	$\lambda_{\text{r}}$	$\lambda_{\text{wd}}$	$\sigma$
{1.0, 0.1}	{1.0, 2.0}	{0.001, 0.01, 0.1}	{0.0, 0.01}	{1.0, 2.0}

**Tabella 3.4:** Accuratezza media dei modelli e deviazione standard, in percentuale, sui 3 task Mutagenesis.

Model	$N_R$	AB	AB+C	AB+C+PS
GraphESN		75.2 ( $\pm 13.0$ )	76.5 ( $\pm 9.4$ )	80.3 ( $\pm 6.6$ )
GraphESN-ZERO	50	79.0 ( $\pm 10.3$ )	78.1 ( $\pm 8.5$ )	79.4 ( $\pm 8.5$ )
GraphESN-ZERO	30	79.6 ( $\pm 10.2$ )	76.0 ( $\pm 8.3$ )	79.7 ( $\pm 8.2$ )
GraphESN-FW	50	80.5 ( $\pm 9.9$ )	76.3 ( $\pm 8.4$ )	79.3 ( $\pm 8.3$ )
GraphESN-FW	30	79.7 ( $\pm 10.6$ )	76.7 ( $\pm 8.3$ )	80.6 ( $\pm 7.2$ )
GraphESN-FOF	50	79.3 ( $\pm 5.3$ )	76.0 ( $\pm 7.5$ )	79.9 ( $\pm 7.3$ )
GraphESN-FOF	30	76.8 ( $\pm 7.9$ )	77.0 ( $\pm 6.0$ )	80.0 ( $\pm 8.0$ )



### 3.3 Considerazioni

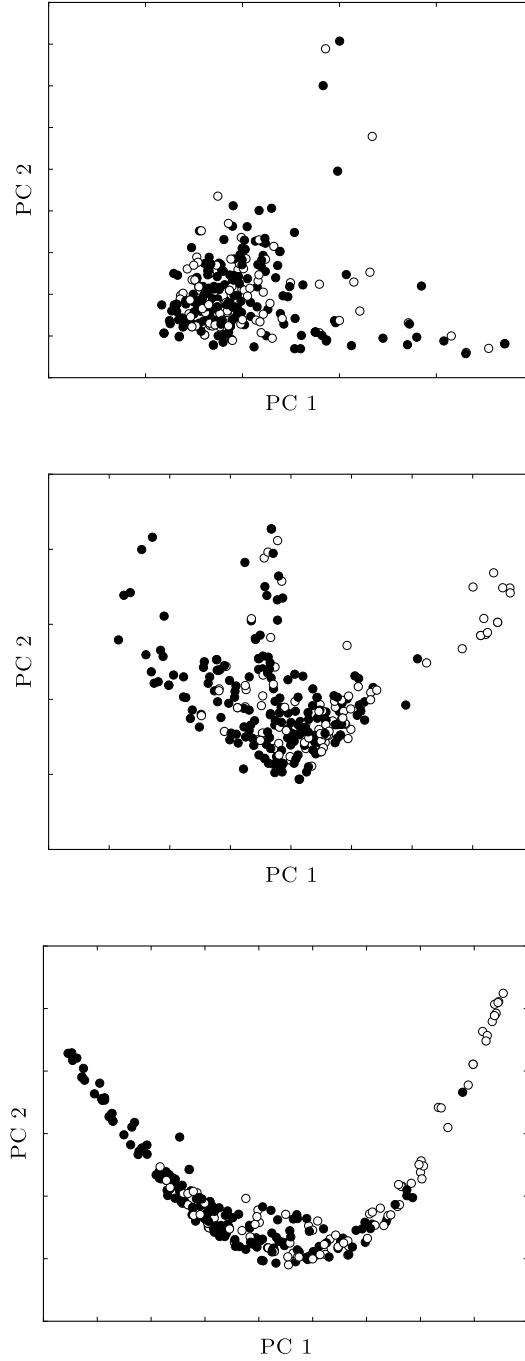
Guardando ai risultati emerge piuttosto chiaramente come la strategia costruttiva, seppur meno onerosa dal punto di vista computazionale, possa rappresentare un potenziamento delle GraphESN. In quest'ottica i modelli proposti si configurano come uno strumento utile nel trattamento dei domini strutturati, efficace e compatibile con scenari in cui le risorse di calcolo o di tempo risultino limitate.



L'analisi delle performance mostra tuttavia alcune zone d'ombra. Si nota ad esempio come l'introduzione e lo sfruttamento di un meccanismo stabile di output-feedback, in particolare nel modello GraphESN-FOF, non corrisponda ad un miglioramento sistematico della capacità di generalizzazione della rete. Ulteriori analisi sperimentali hanno tuttavia permesso di verificare l'effetto degli output-feedback sui reservoir.

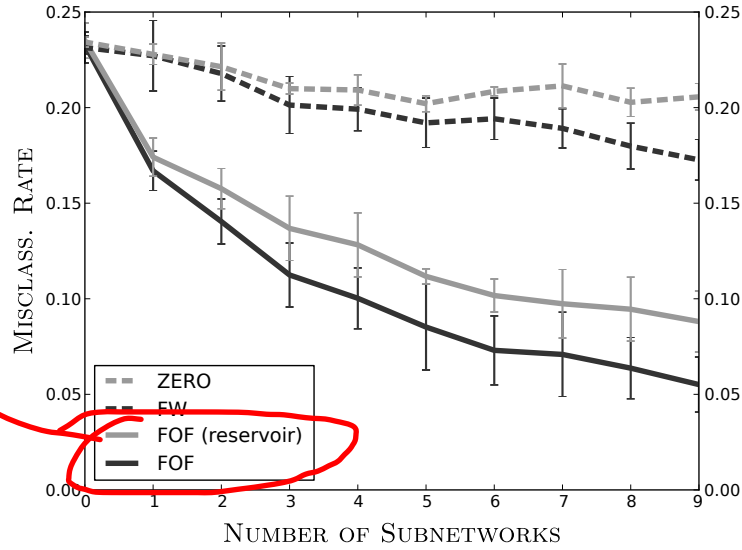
L'organizzazione dello spazio degli stati dei reservoir è stata investigata attraverso la Principal Component Analysis (PCA). La figura 3.1 nella pagina successiva mostra le prime due componenti principali dello spazio degli stati dei reservoir di sotto-reti appartenenti ad una GraphESN-FOF. I punti dei

riferimento



**Figura 3.1:** Plot delle prime due componenti principali dello spazio degli stati dei reservoir nelle sotto-reti. Modello FOF con  $N_R = 50$  applicato al task PTC-FR.

In nero: gli input con target  $-1$ . In bianco: gli input con target  $+1$ . Reservoir riportati: (*alto*) sotto-rete 1; (*centro*) sotto-rete 5; (*basso*) sotto-rete 9.



**Figura 3.2:** Confronto fra i modelli: errore di training (i.e. misclassification rate) e deviazione standard al crescere della rete. Task PTC-FM,  $N_R = 100$ .

plot corrispondono ai grafi in input, colorati in base al target loro assegnato e disposti nello spazio delle features secondo l'encoding corrispondente. Ricordando che tutte le sotto-reti utilizzate condividono lo stesso tipo di iperparametrizzazione, è evidente come lo spazio degli stati di sotto-reti successive venga modificato per effetto degli output-feedback. Dalla figura emerge chiaramente che il meccanismo degli output-feedback modifica le dinamiche del processo di encoding organizzando gli input nello spazio degli stati in maniera consistente con il task affrontato ed aumentando progressivamente la distanza fra input appartenenti a classi distinte. Risulta quindi ben evidente come la presenza degli output-feedback introduca effettivamente dell'informazione supervisionata all'interno di reservoir non adattivi.

L'efficacia dell'adozione di uno schema di output-feedback è inoltre confermata, secondo un approccio diverso, dall'analisi della capacità di fitting dei modelli. La figura 3.2 mostra il confronto fra l'errore commesso, a parità di

iperparametrizzazione ed al crescere della rete, dai tre modelli proposti più un quarto, corrispondente ad una GraphESN-FOF con le connessioni in avanti dirette al solo reservoir. Tenendo in considerazione il solo errore commesso sul training-set, ovvero la semplice capacità di fitting dei modelli, risulta evidente dal grafico come l'introduzione di informazione supervisionata attraverso gli output-feedback determini una accresciuta capacità di approssimazione da parte del modello.

Dalle analisi supplementari risulta dunque come l'introduzione degli output-feedback rappresenti effettivamente una importante risorsa tesa a potenziare l'encoding delle reti, superando in parte i limiti legati all'uso di reservoir non adattivi. A fronte di questo, gli esperimenti svolti suggeriscono l'impiego di forti meccanismi di regolarizzazione e di criteri di stop sofisticati, che impediscano alla rete di accrescere eccessivamente la propria complessità.

Sul piano pratico, gli esperimenti lasciano invece emergere un altro aspetto critico. Come si evince dai risultati riportati, infatti, l'adozione di un approccio costruttivo non è sufficiente a determinare la topologia della rete in maniera completamente automatica. Poiché per la natura del Reservoir Computing la dimensione del reservoir è estremamente importante ai fini del funzionamento della rete, è infatti necessario stabilire a priori quali siano le dimensioni dei reservoir delle sotto-reti. Nel farlo ci si trova di fronte ad una sorta di trade-off:

- Con  $N_R$  troppo piccolo i reservoir delle sotto-reti perdono la capacità di espandere l'input su uno spazio degli stati ad alta dimensionalità, riducendo la capacità di apprendimento delle sotto-reti e della rete nel suo complesso. In questo scenario, inoltre, si rischia di perdere il vantaggio computazionale, pagando il prezzo di un numero molto alto

di iterazioni necessarie alla costruzione della rete (si veda il paragrafo 2.4 a pagina 42). D'altra parte l'utilizzo di sotto-reservoir di piccole dimensioni offre l'opportunità di avere maggiore controllo sulla crescita della rete, che avverrà in maniera poco discontinua.

- Con  $N_R$  troppo grande si ottengono al contrario singole sotto-reti con una grande capacità di apprendimento che però più facilmente metteranno la rete nella condizione di incappare in situazioni di overfitting. L'aggiunta di una singola sotto-rete determina infatti in questo scenario un aumento elevato, e poco controllabile, della complessità del modello, risultando nell'impossibilità di apprendere dinamicamente una topologia che sia davvero adeguata al task affrontato.

In quest'ottica l'introduzione dell'approccio costruttivo rappresenta dunque una notevole semplificazione, ma non una definitiva soluzione, al problema proprio del Reservoir Computing di dover stabilire a priori una dimensione ottimale per il reservoir.



# 4

## Conclusioni

Nel corso della tesi sono stati presentati dei nuovi modelli di Reti Neurali Ricorrenti per l'apprendimento di trasduzioni strutturali sul dominio dei grafi. Un duplice contributo caratterizza i modelli proposti: l'adozione di un approccio costruttivo e l'introduzione di uno schema di output-feedback stabile nell'ambito del Reservoir Computing.

L'aver adottato una strategia costruttiva ha permesso la formulazione di modelli molto efficienti dal punto di vista computazionale. L'allenamento ed il processo di model selection per le reti proposte risultano infatti vantaggiosi anche se si guarda al solo ambito del Reservoir Computing, di per sé caratterizzato da oneri computazionali estremamente ridotti. In quest'ottica, quanto realizzato si configura come uno strumento pratico ed efficace, unico nel suo genere, nel trattamento di domini strutturati ed in presenza di risorse di calcolo o di tempo limitate.

Con l'introduzione di un sistema stabile di output-feedback si è invece implementato un meccanismo che permetta di sfruttare la presenza di informazione supervisionata all'interno del processo di encoding del Reservoir



Computing. L'uso di informazioni di questo tipo è stato possibile grazie alla fusione di due mondi ad oggi separati, quello del Reservoir Computing e quello delle Reti Neurali Costruttive, e dà risposta, anche se in maniera parziale, ad un'esigenza già emersa nella sfera del Reservoir Computing.

L'efficacia dei modelli realizzati è stata testata su problemi reali appartenenti all'ambito della Chemioinformatica. Gli esperimenti svolti hanno mostrato come i modelli proposti, nonostante richiedano un impiego inferiore di risorse computazionali, siano in grado in molti casi di migliorare le performance raggiunte attraverso l'utilizzo di modelli non costruttivi. L'analisi sperimentale ha inoltre messo in luce gli effetti degli output-feedback sugli stati dei reservoir, evidenziando come il loro impiego possa effettivamente arricchirne le dinamiche, influenzandole in maniera consistente con il task affrontato.

Il lavoro svolto ha anche messo in luce alcune criticità. Il ruolo positivo degli output-feedback, ad esempio, non è completamente supportato dai risultati sperimentali, almeno in termini di performance. Non sempre, infatti, il loro impiego ha comportato un miglioramento della capacità di generalizzazione della rete. Oltre a questo, il numero di unità nel reservoir rimane comunque un iperparametro critico, limitando i benefici derivanti dall'approccio costruttivo. Entrambi questi aspetti possono essere oggetto di ulteriori ricerche.

L'aver indirizzato un approccio innovativo nel campo del Reservoir Computing lascia inoltre emergere ulteriori spunti di sperimentazione. La strategia costruttiva proposta è infatti estremamente flessibile ed è ragionevole ipotizzare che molte altre varianti topologiche, oltre a quelle proposte, possano essere oggetto di investigazione. Altrettanto importante e variegata è anche

la gamma degli algoritmi di learning o dei sotto-task utilizzabili per l'allenamento delle sotto-reti, anch'essa possibile oggetto di ulteriori indagini così come l'impiego di sotto-reti eterogenee selezionate tramite un pool.

È infine opportuno sottolineare come parte dei contenuti della tesi siano oggetto dell'articolo scientifico *Constructive Reservoir Computation with Output Feedbacks for Structured Domains*, attualmente in fase di revisione per essere presentato nel corso del XX European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning.

## Bibliografia

- [1] Pierre Baldi e Gianluca Pollastri. The principled design of large-scale recursive neural network architectures–DAG-RNNs and the protein structure prediction problem. *J. Mach. Learn. Res.*, 4:575–602, December 2003.
- [2] Yoshua Bengio, Paolo Frasconi, e Patrice Simard. The problem of learning long-term dependencies in recurrent networks. In *Neural Networks, 1993., IEEE International Conference on*, pp. 1183–1188 vol.3, 1993.
- [3] Kenji Doya. Bifurcations of recurrent neural networks in gradient descent learning. *IEEE Transactions on Neural Networks*, 1:75–80, 1993.
- [4] Scott E. Fahlman e Christian Lebiere. The Cascade-Correlation learning architecture. In *Advances in Neural Information Processing Systems 2*, pp. 524–532. Morgan Kaufmann, 1990.
- [5] Chrisantha Fernando e Samipsa Sojakka. Pattern recognition in a bucket. In *Advances in Artificial Life, Lecture Notes In Computer Science*, pp.

- 588–597. Springer Berlin / Heidelberg, 2003.
- [6] Paolo Frasconi, Marco Gori, e Alessandro Sperduti. A general framework for adaptive processing of data structures. *IEEE Transactions on Neural Networks*, 9:768–786, 1998.
- [7] Holger Fröhlich, Jörg K. Wegner, Florian Sieker, e Andreas Zell. Optimal assignment kernels for attributed molecular graphs. In *Proceedings of the 22nd international conference on Machine learning*, ICML '05, pp. 225–232, New York, NY, USA, 2005. ACM.
- [8] Claudio Gallicchio e Alessio Micheli. Graph Echo State Networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN) 2010*, pp. 1–8, July 2010.
- [9] Claudio Gallicchio e Alessio Micheli. TreeESN: a preliminary experimental analysis. In *Proceedings of the ESANN 2010*, pp. 333–338, d-side, 2010.
- [10] Claudio Gallicchio e Alessio Micheli. Architectural and markovian factors of Echo State Networks. *Neural Networks*, 24(5):440–456, 2011.
- [11] Barbara Hammer, Alessio Micheli, e Alessandro Sperduti. Universal approximation capability of cascade correlation for structures. *Neural Computation*, 17(5):1109–1159, 2005.
- [12] Barbara Hammer e Peter Tiño. Recurrent neural networks with small weights implement definite memory machines. *Neural Computation*, 15:1897–1929, 2003.

- 
- [13] Trevor Hastie, Robert Tibshirani, e Jerome H. Friedman. *The Elements of Statistical Learning*. Springer, corrected edizione, July 2003.
- [14] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Macmillan, New York, 1994.
- [15] Christoph Helma, Ross D. King, Stefan Kramer, e Ashwin Srinivasan. The predictive toxicology challenge 2000-2001. *Bioinformatics*, 17(1):107–108, 2001.
- [16] Guang-Bin Huang, Qin-Yu Zhu, e Chee-Kheong Siew. Extreme Learning Machine: theory and applications. *Neurocomputing*, 70(1-3):489–501, December 2006.
- [17] Herbert Jaeger. The “echo state” approach to analyzing and training recurrent neural networks. Technical Report GMD Report 148, German National Research Center for information Technology, 2001. <http://www.faculty.iu-bremen.de/hjaeger/pubs/EchoStatesTechRep.pdf>.
- [18] Herbert Jaeger. Short term memory in Echo State Networks. Technical Report GMD Report 152, German National Research Center for information Technology, 2001. <http://www.faculty.iu-bremen.de/hjaeger/pubs/STMEchoStatesTechRep.pdf>.
- [19] Herbert Jaeger e Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, April 2004.

- [20] Herbert Jaeger, Wolfgang Maass, e Jose Principe. Editorial: Special issue on Echo State Networks and Liquid State Machines. *Neural Networks*, 20(3):287–289, 2007.
- [21] Enno Littmann e Helge Ritter. Learning and generalization in cascade network architectures. *Neural Comput.*, 8:1521–1539, October 1996.
- [22] Mantas Lukoševičius. Echo State Networks with trained feedbacks. Technical Report No. 4, Jacobs University Bremen, 2007. [http://wwwback.jacobs-university.de/imperia/md/content/groups/research/techreports/tfbesn\\_iubtechreport.pdf](http://wwwback.jacobs-university.de/imperia/md/content/groups/research/techreports/tfbesn_iubtechreport.pdf).
- [23] Mantas Lukoševičius e Herbert Jaeger. Reservoir Computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, August 2009.
- [24] Wolfgang Maass, Thomas Natschläger, e Henry Markram. Real-time computing without stable states: a new framework for neural computation based on perturbations. *Neural Comput*, 14(11):2531–60, 2002.
- [25] Mario Martelli. *Introduction to Discrete Dynamical Systems and Chaos*. Wiley, 1999.
- [26] Alessio Micheli. Neural Network for Graphs: a contextual constructive approach. *IEEE Transactions on Neural Networks*, 20(3):498–511, 2009.
- [27] Alessio Micheli, Diego Sona, e Alessandro Sperduti. Contextual processing of structured data by recursive cascade correlation. *IEEE Transactions on Neural Networks*, 15:1396–1410, 2003.

- 
- [28] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.
- [29] Lutz Prechelt. Investigation of the CasCor family of learning algorithms. *Neural Networks*, 10:885–896, 1996.
- [30] Danil Prokhorov. Echo State Networks: appeal and challenges. In *Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference on*, volume 3, pp. 1463–1466 vol. 3, 2005.
- [31] Dana Ron, Yoram Singer, e Naftali Tishby. The power of amnesia: Learning probabilistic automata with variable memory length. In *Machine Learning*, pp. 117–149, 1996.
- [32] Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, e Gabriele Monfardini. The Graph Neural Network Model. *IEEE Transactions on Neural Networks*, 20(1):61–80, January 2009.
- [33] Benjamin Schrauwen, David Verstraeten, e Jan Van Campenhout. An overview of Reservoir Computing: theory, applications and implementations. In *Proceedings of the 15th European Symposium on Artificial Neural Networks*, pp. 471–482, April 2007.
- [34] Frank J. Śmieja. Neural network constructive algorithms: trading generalization for learning efficiency? *Circuits Syst. Signal Process.*, 12:331–374, February 1993.
- [35] Qingsong Song e Zuren Feng. Effects of connectivity structure of complex Echo State Network on its prediction performance for nonlinear time series. *Neurocomputing*, 73(10-12):2177 – 2185, 2010.

- 
- [36] Alessandro Sperduti e Antonina Starita. Supervised neural networks for the classification of structures. *IEEE Transactions on Neural Networks*, 8:714–735, 1997.
- [37] Ashwin Srinivasan, Stephen H. Muggleton, Ross D. King, e Michael J. E. Sternberg. Mutagenesis: ILP experiments in a non-determinate biological domain. In *Proceedings of the 4th International Workshop on Inductive Logic Programming*, volume 237, pp. 217–232, 1994.
- [38] Jochen J. Steil. Backpropagation-Decorrelation: online recurrent learning with  $O(N)$  complexity. In *Proc. IJCNN*, volume 1, pp. 843–848, July 2004.
- [39] Peter Tiño, Michal Čerňanský, e Ľubica Beňušková. Markovian architectural bias of recurrent neural networks. *IEEE Transactions on Neural Networks*, 15(1):6–15, 2004.
- [40] David Verstraeten, Benjamin Schrauwen, Michiel D’Haene, e Dirk Stroobandt. An experimental unification of reservoir computing methods. *Neural Networks*, 20(3):391–403, April 2007.
- [41] Paul J. Werbos. Backpropagation Through Time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, August 2002.
- [42] Ronald J. Williams e David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.*, 1(2):270–280, 1989.



- 
- [43] Francis Wyffels, Benjamin Schrauwen, e Dirk Stroobandt. Stable output feedback in reservoir computing using ridge regression In *Proceedings of the 18th International Conference on Artificial Neural Networks*. A cura di V. Kurkova, R. Neruda, e J. Koutník, pp. 808–817, Prague, 2008. Springer.