

Capitolo 1

Background

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 ?? a pagina ??). 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* [20] 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 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.
- 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/merging per le porzioni di rete non direttamente interessate dal learning locale.
- 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).

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 complessità del modello possa crescere indefinidamente espone i modelli costruttivi al verificarsi di situazioni di overfitting.

1.1.1 Cascade Correlation

L'algoritmo *Cascade Correlation* rappresenta probabilmente uno dei più diffusi e comuni 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 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 σ_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, 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 layer¹, 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.

¹Essendo le unità nascoste in cascata, ogni unità rappresenta di fatto un layer a sé stante. Si veda anche la Figura 1.1.

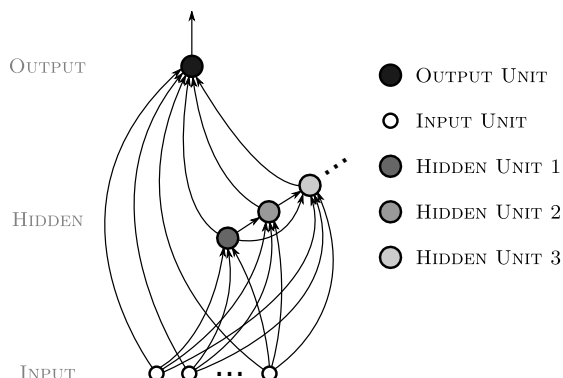



Figura 1.1: Cascade Correlation. Architettura di una rete con 1 unità di output e 3 unità nascoste.


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 a fronte mostra l'architettura di una rete con 3 unità nascoste ed un'unica unità di output.

1.2 Reservoir Computing

Il *Reservoir Computing*  è un paradigma emergente, nell'ambito delle *Reti Neurali Ricorrenti*, che ha origine da due classi di modelli: *Echo State Networks* (ESN) [10] e *Liquid State Machines* (LSM) [16]. 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 computazionale² 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 strutturati³.

Una ~~generica~~ Rete Neurale Ricorrente viene usata per  cessare dati sequenziali calcolando 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

²Le LSM nascono infatti nell'ambito delle neuroscienze e mantengono caratteristiche di forte ispirazione biologica.

³Il 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 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 1.3.1 a pagina 15).

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

Dalla formula risulta evidente come lo stato interno e l'output giochino un ruolo differente: la *funzione di transizione di stato* τ 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) [24] e *Real-Time Recurrent Learning* (RTRL) [25], 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 che la convergenza possa non essere garantita [2].
- 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 [1].

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 [23], 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*⁴

⁴Per le sue caratteristiche, l'algoritmo BPDC viene talvolta indicato a tutti gli effetti come appartenente all'ambito del Reservoir Computing. Si veda, ad esempio, [21].

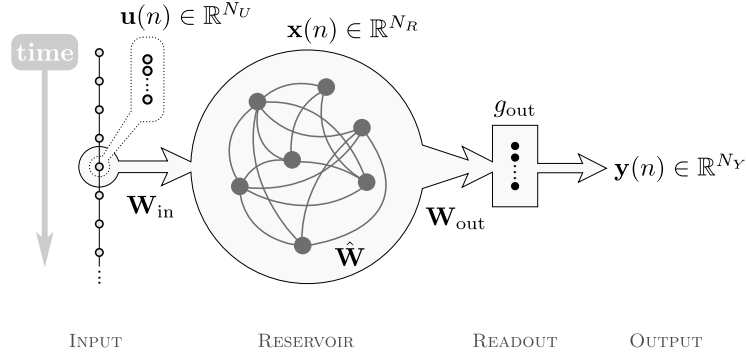


Figura 1.2: Schematizzazione grafica di una Echo State Network (ESN).

(BPDC) [22] o delle *Extreme Learning Machines* (ELM) [9]. In termini ancor più generali, l'approccio del Reservoir Computing è inoltre riconducibile all'utilizzo di un kernel, come sottolineato anche in [13].

1.2.1 Echo State Networks

Definiamo formalmente le ESN [10, 11, 12], 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:

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

dove f ed g_{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.

Varianti comuni prevedono inoltre l'esistenza di connessioni dirette dall'input al readout o all'indietro, dal readout al reservoir (si veda ad esempio [10, 11]). La presenza di queste ultime in particolare implica considerazioni specifiche, poiché ha impatto sulla stabilità complessiva del modello [14].

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 $\mathbf{y}_{\text{target}}(n)$. 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 $\hat{\mathbf{W}}$ rispettino determinate caratteristiche: informalmente possiamo dire che è necessario che lo stato interno della rete sia un'eco della sequenza in input.

In [10, 11] 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 σ_{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 λ_{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 [7] la presenza della echo state property viene messa in relazione con la *contrattività* della funzione di transizione di stato τ . In particolare si dimostra che se τ è contrattiva con parametro $C < 1$, ovvero

$$\exists C \in [0, 1) \text{ tale che } \forall \mathbf{u} \in \mathbb{R}^{N_U}, \forall \mathbf{x}, \mathbf{x}' \in \mathbb{R}^{N_R} : \quad \|\tau(\mathbf{u}, \mathbf{x}) - \tau(\mathbf{u}, \mathbf{x}')\| \leq C \|\mathbf{x} - \mathbf{x}'\| \quad (1.7)$$

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 *natural  arkoviana*: 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 [8, 23] 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 [18]). 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 [12], [19, pag. 8], o [15, 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 rete⁵. L’indipendenza fra i pesi delle singole unità di output ed il fatto che il reservoir non viene adattato permettono inoltre di estendere modelli sui quali sia già stato effettuato l’apprendimento, semplicemente aggiungendo nuove unità di output. 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 [17], 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 [17].

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

TODO (INTRODUZIONE AI MODELLI PER ALBERI E GRAFI (CRCC, GNN, NN4G). PROBLEMI LEGATI AL TRATTAMENTO DI GRAFI CICLICI.)

1.3.1 Graph Echo State Networks

Le *Graph Echo State Network* (GraphESN) [5] rappresentano un’estensione delle ESN (si veda il paragrafo 1.2.1) — e delle TreeESN [6] — 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 i dati, determinando l’applicabilità del modello ad una classe di strutture che comprende grafi non diretti e/o ciclici.

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

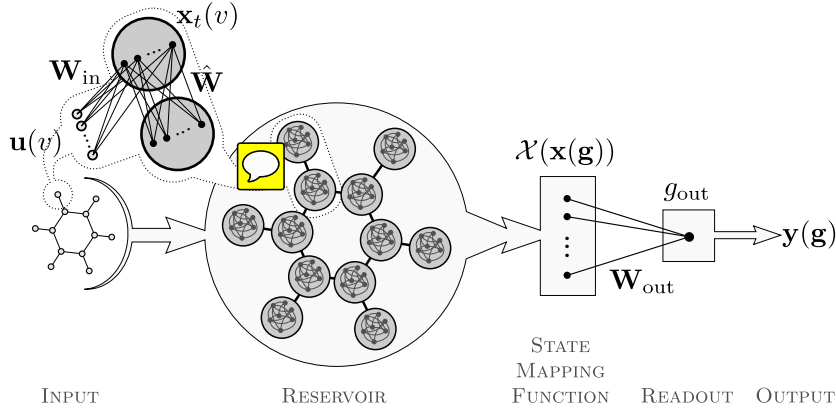


Figura 1.3: Schema di una GraphESN per l'apprendimento di trasduzioni structure-to-element.

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

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 $E(\mathbf{g}) = \{(u, v) : u, v \in V(\mathbf{g})\}$ 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 adiacenti⁶, $\mathcal{N}(v) = \{u \in V : (u, v) \in E\}$. Sia il grado di un vertice v il numero dei suoi vicini, $degree(v) = |\mathcal{N}(v)|$, 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 $(\mathbb{R}^{N_U})^\#$.

Una *trasduzione strutturale* \mathcal{T} è 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 in input.

Una trasduzione strutturale 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*.

⁶Nel 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.

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  $\mathbf{x}(\mathbf{g})$ 

```

Secondo l'approccio neurale — ricorsivo e ricorrente — la funzione di encoding \mathcal{T}_{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} ed indichiamo con $\mathbf{x}(\mathbf{g}) \in \mathbb{R}^{|V(\mathbf{g})|N_R}$ la concatenazione, secondo un ordine arbitrario, degli stati associati a tutti i vertici dell'input. Tale codifica viene realizzata in una GraphESN dal reservoir, che calcola iterativamente una *funzione di transizione locale di stato*: al passo t , la funzione associa ad ogni vertice v una codifica, o valore di stato, $\mathbf{x}_t(v)$ secondo la seguente equazione



$$\mathbf{x}_t(v) = \tau(\mathbf{u}(v), \mathbf{x}_{t-1}(\mathcal{N}(v))) = f \left(\mathbf{W}_{\text{in}} \mathbf{u}(v) + \sum_{w \in \mathcal{N}(v)} \hat{\mathbf{W}} \mathbf{x}_{t-1}(w) \right) \quad (1.10)$$

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 ricorrenti⁷ tra i vicini di un vertice ed f è la funzione di attivazione, tipicamente sigmoideale, delle unità del reservoir.

In questo caso, perché τ sia *contrattiva* rispetto allo stato, è necessario che valga la seguente condizione:

$$\exists C \in [0, 1) \text{ tale che } \forall \mathbf{u} \in \mathbb{R}^{N_U}, \forall \mathbf{x}_1, \dots, \mathbf{x}_k, \mathbf{x}'_1, \dots, \mathbf{x}'_k \in \mathbb{R}^{N_R} : \\ \|\tau(\mathbf{u}, \mathbf{x}_1, \dots, \mathbf{x}_k) - \tau(\mathbf{u}, \mathbf{x}'_1, \dots, \mathbf{x}'_k)\| \leq C \max_{i=1, \dots, k} \|\mathbf{x}_i - \mathbf{x}'_i\| \quad (1.11)$$

dove $\|\cdot\|$ è una qualsiasi norma su \mathbb{R}^{N_R} . Considerando per τ l'implementazione dell'equazione (1.10), 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

$$\sigma = \|\hat{\mathbf{W}}\|_2 k < 1 \quad (1.12)$$

⁷In questo caso il termine *ricorrente* è appropriato se si pensa alle analogie con i modelli neurali per il trattamento di domini strutturati e se si guarda alla rete nel suo complesso, può risultare per certi versi fuorviante: dall'unico reservoir ricorrente delle ESN si passa infatti in questo caso a più reservoir distinti, non singolarmente ricorrenti, **secondo una sorta di unfolding che segue la topologia del grafo** (si veda la Figura 1.3).

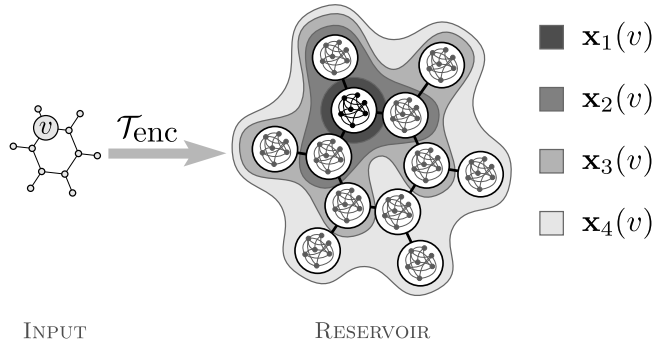


Figura 1.4: Vertici coinvolti nel calcolo di $\mathbf{x}_t(v)$ durante quattro passi del processo di encoding.

dove k è il grado massimo calcolato su tutti i grafi in \mathcal{G} e σ , che controlla il grado di contrattività delle dinamiche del reservoir, è chiamato *coefficiente di contrazione*. Una volta che la contrattività di τ sia assicurata, la convergenza del calcolo iterativo del reservoir è garantita dal *Principio di Contrazione di Banach* 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 ϵ . L'Algoritmo 1 a fronte 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 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 τ 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 [7] o alberi [6], al concetto di *vicinato comune* [5]. La Figura 1.4 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 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 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

$$\mathcal{T} = \mathcal{T}_{\text{out}} \circ \mathcal{X} \circ \mathcal{T}_{\text{enc}} \quad (1.13)$$

Benché la scelta della funzione \mathcal{X} sia arbitraria, si distinguono principalmente due alternative. Con *supersource state mapping* [6], lo stato dell'intero grafo $\mathbf{x}(\mathbf{g})$ viene mappato nello stato di un solo nodo *supersource*, qualora per la natura dei dati o del problema sia individuabile un vertice che dipenda da tutti gli altri⁸. In alternativa, con *mean state mapping*, $\mathcal{X}(\mathbf{x}(\mathbf{g}))$ viene calcolato come la media degli stati corrispondenti ai vertici di \mathbf{g} :

$$\mathcal{X}(\mathbf{x}(\mathbf{g})) = \frac{1}{|V(\mathbf{g})|} \sum_{v \in V(\mathbf{g})} \mathbf{x}(v) \quad (1.14)$$

Come in una ESN, la funzione di output \mathcal{T}_{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:

$$\mathbf{y}(v) = g(\mathbf{x}(v)) = g_{\text{out}}(\mathbf{W}_{\text{out}}\mathbf{x}(v)) \quad (1.15)$$

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_{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}_{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:

$$\mathbf{y}(\mathbf{g}) = g(\mathcal{X}(\mathbf{x}(\mathbf{g}))) = g_{\text{out}}(\mathbf{W}_{\text{out}}\mathcal{X}(\mathbf{x}(\mathbf{g}))) \quad (1.16)$$

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 \mathbf{W}_{out} , realizzabile attraverso regressione lineare⁹ 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 1.3 a pagina 16 riassume schematicamente la struttura ed il funzionamento complessivo di una GraphESN che modelli traduzioni structure-to-element.

⁸Questa caratteristica, basata sul concetto di *supersource*, nel caso di alberi e sequenze, è comunque realizzabile “artificialmente” per qualsiasi grafo aggiungendo un vertice che sia collegato a tutti gli altri.

⁹Purché g_{out} sia invertibile, è possibile ricondurre g ad una combinazione lineare semplicemente modificando i valori target: $\mathbf{y}'_{\text{target}} = g_{\text{out}}^{-1}(\mathbf{y}_{\text{target}})$.

Bibliografia

- [1] 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.
- [2] Kenji Doya. Bifurcations of recurrent neural networks in gradient descent learning. *IEEE Transactions on Neural Networks*, 1:75–80, 1993.
- [3] 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.
- [4] Chrisantha Fernando e Sampsa Sojakka. Pattern recognition in a bucket. In *Advances in Artificial Life, Lecture Notes In Computer Science*, pp. 588–597. Springer Berlin / Heidelberg, 2003.
- [5] 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.
- [6] Claudio Gallicchio e Alessio Micheli. TreeESN: a preliminary experimental analysis. In *Proceedings of the ESANN 2010*, pp. 333–338, d-side, 2010.
- [7] Claudio Gallicchio e Alessio Micheli. Architectural and markovian factors of Echo State Networks. *Neural Networks*, 24(5):440–456, 2011.
- [8] Barbara Hammer e Peter Tiño. Recurrent neural networks with small weights implement definite memory machines. *Neural Computation*, 15:1897–1929, 2003.
- [9] Guang-Bin Huang, Qin-Yu Zhu, e Chee-Kheong Siew. Extreme Learning Machine: theory and applications. *Neurocomputing*, 70(1-3):489–501, December 2006.
- [10] 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>.
- [11] 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>.

- [12] Herbert Jaeger e Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, April 2004.
- [13] 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.
- [14] 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.
- [15] Mantas Lukoševičius e Herbert Jaeger. Reservoir Computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, August 2009.
- [16] 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.
- [17] 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.
- [18] 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.
- [19] 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, 4 2007.
- [20] Frank J. Śmieja. Neural network constructive algorithms: trading generalization for learning efficiency? *Circuits Syst. Signal Process.*, 12:331–374, February 1993.
- [21] 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.
- [22] Jochen J. Steil. Backpropagation-Decorrelation: online recurrent learning with $O(N)$ complexity. In *Proc. IJCNN*, volume 1, pp. 843–848, Jul 2004.
- [23] Peter Tiño, Michal Čerňanský, e Ľubica Beňušková. Markovian architectural bias of recurrent neural networks. *IEEE Transactions on Neural Networks*, 15(1):6–15, 2004.
- [24] Paul J. Werbos. Backpropagation Through Time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, August 2002.
- [25] Ronald J. Williams e David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.*, 1(2):270–280, 1989.