Learning and Neural Encoding of Multiple Behaviors (in Italian)

Thesis · July 2013

DOI: 10.13140/RG.2.1.1348.2720

1 author:

Andrea de Giorgio
KTH Royal Institute of Technology
15 PUBLICATIONS **36** CITATIONS

SEE PROFILE

Università degli Studi di Napoli Federico II

Scuola Politecnica e delle Scienze di Base

Dipartimento di Ingegneria Elettrica e delle Tecnologie dell'Informazione

Corso di Laurea in Ingegneria Elettronica

Tesi di Laurea

APPRENDIMENTO E CODIFICA NEURALE DI COMPORTAMENTI MULTIPLI

Relatori
Prof. Guglielmo Tamburrini
Prof. Roberto Prevete

Candidato
Andrea de Giorgio
Matr. 528/1060

Correlatore
Prof. Carlo Sansone

Anno Accademico 2012/2013

Ad Antonio, Attilio, Bianca, Biancamaria, Brigida,
Carlo d.G., Carlo G., Claudio Fa., Claudio Fe., Clelia,
Davide, Dianna, Douglas, Elettra, Elio,
Elkhonon, Filippo, Gustavo, Ivana, James,
Leonardo, Luciano, Luigi, Manfredo, Marcello,
Mariano, Marina, Martina, Matteo, Nicola,
Ornella, Raffaele, Raimondo, Richard, Roberta,
Roberto, Salvatore, Stefania, Valentino e Violetta

in 9567 giorni di vita, con il vostro aiuto.

17 luglio 2013

Introduzione

Nell'ultima decade alcune evidenze sperimentali sembrano suggerire che singole aree del sistema nervoso siano capaci di sottendere a più di un singolo comportamento (funzione, compito) e di realizzare "transizioni" da un comportamento all'altro in maniera rapida e reversibile. Tale uso delle stesse aree per differenti funzioni è, in effetti, in contrasto con l'ipotesi maggiormente accettata in letteratura secondo cui il sistema nervoso centrale sia organizzato sulla base di aree dedicate a funzioni o compiti specifici. Le proprietà di rapidità e reversibilità di cambiamento di comportamento, inoltre, suggeriscono che tale fenomeno possa avvenire senza modificare la struttura sottostante (cioè l'efficacia sinaptica) e che, quindi, non sia un prodotto di qualche tipo di apprendimento. Alcuni recenti lavori presentati in letteratura si pongono, allora, il problema di come spiegare tale fenomeno. Un'interpretazione è legata alla possibilità che vi siano circuiti neurali, a struttura fissa, con due tipi differenti di input, ovvero, non solo quello "classico", ma anche un input ausiliare che contenga la codifica di una specifica struttura neurale, la quale è interpretata dal circuito neurale. Si pensa, cioè, che ogni singola rete neurale possa potenzialmente rappresentare l'insieme di più circuiti latenti, in grado di eseguire ora una funzione, ora un'altra, in base ad una codifica data in ingresso che ne riprogrammi il funzionamento. Una stessa rete neurale, ad esempio, potrebbe essere riutilizzata "in tempo reale" per gestire più di uno schema motorio, a seconda delle necessità, come risultato di input ausiliari differenti, ovviamente appresi in precedenza. Tale struttura programmabile, nel passaggio da un comportamento ad un altro, non richiedendo né ulteriori fasi di apprendimento per i comportamenti già acquisiti, né alcuna modifica strutturale, sembrerebbe proprio l'ideale per eseguire transizioni rapide e reversibili.

Obiettivo della mia tesi è studiare come la presenza di circuiti neurali programmabili possa rendere l'apprendimento di differenti comportamenti, da parte di una singola struttura, più efficace. A tal fine, ho messo a confronto due architetture, una programmabile e l'altra non programmabile, come parte di uno scenario robotico appositamente progettato per testare la possibilità di apprendere ed esibire comportamenti multipli. Più in particolare, ho supposto che un *robot* possa imparare, tramite la sua architettura di controllo,

programmabile o meno, a raggiungere, su richiesta, ciascuna delle otto terminazioni differenti di un labirinto costituito da una serie di tre biforcazioni e di dimensioni variabili. Fondamentale, in tale scenario, la necessità per l'architettura di controllo di apprendere, e poi esibire, comportamenti e non semplici traiettorie. In tal modo, ho potuto raccogliere dati sufficienti per realizzare un'opportuna analisi.

Architetture neurali artificiali, algoritmi di apprendimento e funzioni per la simulazione e l'analisi dei dati sono stati interamente sviluppati in codice Matlab. In particolare, il mio contributo si è concentrato sull'adattamento di un framework già presente progettando ed implementando nuovi algoritmi di simulazione e analisi per lo specifico problema da analizzare. Ho svolto, pertanto, un lavoro complessivo composto da fasi di analisi del problema, progettazione dell'esperimento, scrittura del codice, debugging, esecuzione finale ed elaborazione dei dati.

Nel capitolo 1 presento il problema biologico dei fenomeni di cambiamento di comportamento, in particolare, dopo una breve introduzione (§1.1), fornisco una descrizione delle soluzioni in termini biologici proposte in letteratura (§1.2) e illustro una possibile soluzione in termini di programmabilità (§1.3).

Nel capitolo 2 illustro la struttura del neurone biologico (§2.1) e presento il modello di rete neurale artificiale usato in questa tesi (§2.2). In seguito, introduco un'architettura di rete neurale, a partire da quella biologicamente plausibile scelta in precedenza, ma in più dotata della proprietà di programmabilità (§2.4) e ne illustro struttura e funzionamento tramite l'implementazione delle reti moltiplicative (§2.4.1) ed alcuni esempi funzionanti (§2.5).

Nel capitolo 3 descrivo l'esperimento robotico, con il suo scenario (§3.1), le strutture, i parametri caratteristici e le modalità di apprendimento delle architetture di tipo non programmabile (§3.2) e programmabile (§3.3) e i criteri di valutazione utilizzati nelle fasi di apprendimento e di analisi finale (§3.4).

Nel capitolo 4 riporto tutti i risultati ottenuti per l'apprendimento con uno specifico labirinto (§4.2), per diversi labirinti appresi contemporaneamente (§4.3) e per il test delle due architetture su labirinti di grandezza casuale (§4.4).

Le conclusioni finali, alcune possibili critiche e gli sviluppi futuri sono presentati nel capitolo 5.

Indice

Abbreviazioni

ANN *Artificial Neural Network* - Rete Neurale Artificiale

ANP Architettura Non Programmabile

AP Architettura Programmabile

CNS *Central Nervous System* - Sistema nervoso centrale

CTRNN *Continuous Time Recurrent Neural Network* - Rete Neurale Ricorrente a Tempo Continuo

ECF *Extracellular fluid* - Fluido extracellulare

ECM *Extracellular matrix* - Matrice extracellulare

FNN *Feedforward Neural Network* - Rete Neurale *Feedforward*

GJ *Gap Junctions* - Giunzioni di *gap*

LF *Left Follower* - Inseguitore di Sinistra

PNN *Programmable Neural Network* - Rete Neurale Programmabile

RF *Right Follower* - Inseguitore di Destra

STG Ganglio Stomatogastrico

TNT Tunneling Nanotubes - Tunnel a Nanotubi

VT *Volume Transmission* - Trasmissione di Volume

WT *Wiring Transmission* - Trasmissione Cablata

Capitolo 1

Fenomeni di cambiamento di comportamento

In questo capitolo illustriamo il problema biologico della necessità di esibire fenomeni di cambiamento di comportamento rapidi e reversibili, in riferimento al tipo di strutture e meccanismi neurali coinvolti.

1.1 Il problema

Il sistema nervoso centrale (CNS) è in definitiva responsabile di tutto ciò che percepiamo, facciamo e pensiamo. Esso svolge molte funzioni critiche che sfuggono alla nostra percezione. Per esempio, coordina l'attività dei nostri organi svolgendo una funzione fondamentale per il mantenimento dell'omeostasi. La funzione cerebrale è estremamente complessa e molto difficile da chiarire usando le correnti metodologie scientifiche. In letteratura, è comunemente accettata la teoria che il sistema nervoso centrale sia organizzato in aree neurali anatomofunzionali, cioè costituito da moduli "*special-purpose*" dedicati a singole funzioni. Nella storia della neurofisiologia questa teoria, infatti, è stata supportata dalle numerose osservazioni dirette sul comportamento di soggetti con specifiche aree lesionate, in quanto, un'area cerebrale non più integra avrà le proprie funzioni compromesse e, allora, tali funzioni potranno essere identificate e attribuite alla struttura in analisi (si vedano, ad esempio, i lavori di Goldberg, 2004).

Tuttavia, una nuova e ampiamente accettata proposta, in contrasto con la precedente, vuole che alcune aree cerebrali siano organizzate diversamente, cioè in modo da implementare più di una funzione nella medesima struttura. Sono infatti stati presentati alcuni modelli computazionali in cui, ad esempio, aree cerebrali vengono controllate da altre parti del cervello (Oztop e Arbib, 2002; Sauser e Billard, 2006; Hurley, 2008; Roy, 2008), oppure eseguono diverse operazioni sotto condizioni differenti (Fyhn *e altri*, 2007) o, an-

cora, che siano riciclate (Dehaene *e altri*, 2004), ovvero usate in differenti circuiti cerebrali per differenti categorie di funzioni e domini cognitivi (Anderson, 2010).

Come lo stesso Anderson (2010) ha però dichiarato, la comprensione di come il riuso sia effettivamente implementato nel cervello, lascia molte questioni ancora aperte, tra le quali, come una struttura possa esibire selettività tra più funzioni, in tempi comparabili con quelli di reazione agli stimoli e in luogo degli specifici collegamenti strutturali preesistenti; oppure come sia possibile per una struttura fissa implementare funzioni diverse quando questa sia eventualmente parte di differenti circuiti neurali.

Recenti osservazioni fisiche, sembrano inoltre supportare questo tipo di visione, delineando un tipo di rete neurale biologica in grado di esibire fenomeni di cambiamento di comportamento con transizioni rapide e reversibili, a partire da una struttura neurale unica che risulti necessariamente sovraconnessa e tale che quest'eccesso di connessioni non sia casuale o appreso, ma rappresenti in realtà dei circuiti latenti, attivabili di volta in volta per gestire differentemente il flusso d'informazione (Bargmann, 2012).

1.2 Sulle soluzioni proposte in letteratura

Tra le possibili soluzioni presentate in letteratura spiccano la proposta di Agnati *e altri* (2010) riguardo una possibile presenza di due tipi di trasmissioni che agiscano contemporaneamente formando reti più complesse e strettamente interconnesse tra loro: una di tipo *cablato* (WT) ed un'altra *di volume* (VT). Proposta in parte confermata da quella della Bargmann (2012), avvalorata da riscontri biologici, che affida ai soli neuromodulatori un ruolo nel cambiamento di comportamento dei circuiti neurali.

1.2.1 Trasmissione cablata e trasmissione di volume

Volendo considerare i neuroni dal punto di vista biologico, questi, in fin dei conti, sono cellule vere e proprie e la connettività tra cellula e cellula implica, in generale, dei meccanismi propriamente detti di **comunicazione intercellulare**. Agnati *e altri* (2010) definiscono nel loro lavoro due tipi principali di comunicazione intercellulare: trasmissione cablata (WT), dall'inglese *Wiring Transmission*, oppure trasmissione di volume (VT), ovvero *Volume Transmission*. Il criterio di classificazione scelto fa riferimento alle differenti caratteristiche del canale di comunicazione, con confini fisici, nel primo caso (WT), e non delimitato da alcun confine fisico, nel secondo (VT).

Del tipo WT sono la connessione sinaptica classica, le giunzioni di *gap* (GJ) e, di recente scoperta, i tunnel a nanotubi (TNT). La trasmissione classica, che riguarda le connessioni sinaptiche e la trasmissione dei potenziali d'azione, è indubbiamente il mec-

canismo primario su cui si basano i neuroni per trasmettere informazioni e controllare il comportamento.

La trasmissione di volume, al contrario, è caratterizzata dall'assenza di un canale di collegamento che metta in comunicazione la sorgente del segnale con il suo destinatario; questo collegamento è implicitamente non sicuro perché il segnale migra nel fluido extracellulare (ECF). Questo tipo di comunicazione utilizza alcuni canali spesso spazialmente tortuosi e divergenti, costituiti dalle fessure (circa 20 nm di diametro) tra le cellule che sono riempite da fluido extracellulare e dalla matrice extracellulare (ECM). I segnali migrano nello spazio extracellulare e potrebbero essere fermati se finiscono in un percorso senza uscita, un *cul-de-sac*, se vengono disattivati da enzimi, se vengono ripuliti attraverso i capillari oppure se vengono catturati da trasportatori cellulari. Tali segnali possono essere rilasciati da qualsiasi parte di una cellula cerebrale, ossia anche da dendriti, soma e dai terminali assonici mancanti interamente di specializzazioni delle membrane sinaptiche, oppure provvisti di tale specializzazioni e formanti cellule astrogliali, microgliali, oligodendrogliali ed ependimali.

Per la trasmissione di volume possiamo arruolare lo stesso set di segnali impegnati nella trasmissione cablata, ovvero trasmettitori e ioni, ma anche molti altri, tra cui, mediatori chimici intercellulari classici e non classici: neurotrasmettitori, neuromodulatori, fattori di crescita, ioni (ad esempio Ca^{2+}), gas (NO, CO_2, CO) oppure segnali fisici (ad esempio i potenziali d'azione).

Queste due reti, secondo Agnati *e altri* (2010), potrebbero rappresentare un meccanismo di gran lunga più complesso di quello attualmente modellato, in grado di interagire tra loro ed indurre cambiamenti funzionali l'una sull'altra.

1.2.2 Il fenomeno di neuromodulazione

Gli studi condotti sul *Caenorhabditis elegans* e su crostacei (Bargmann, 2012) suggeriscono come non sia possibile leggere il diagramma delle connessioni (connettoma) come se fosse un *set* di istruzioni. Le connessioni anatomiche costituirebbero, invece, un set di potenziali connessioni che prendono forma in riferimento al contesto e agli stati interni della rete, al fine di stabilire diversi percorsi per i flussi di informazione. Il contesto e gli stati interni sono spesso rappresentati molecolarmente da **neuromodulatori**, delle piccole molecole che attivano i recettori accoppiati alle proteine G per modificare le dinamiche neurali, l'eccitabilità e l'efficienza sinaptica. Questi neuromodulatori effettivamente cambiano la composizione di un circuito neurale, reclutando nuovi neuroni o escludendo i partecipanti precedenti (Harris-Warrick e Marder, 1991); riconoscere la loro importanza può essere fondamentale per retroingegnerizzare le funzioni dei circuiti neurali.

Ci interessa evidenziare come questi circuiti biologici cambino le proprie caratteristiche in maniera rapida e reversibile. In Bargmann (2012), possiamo osservare come un circuito motorio composto da 30 neuroni, ovvero il ganglio stomatogastrico (STG) di granchi e aragoste, isolabile e perfettamente funzionante in coltura cellulare, sia influenzato dai neuromodulatori e con questi cambi il suo comportamento per assolvere a diverse funzioni quali la gestione dei differenti ritmi della valvola pilorica e della macinazione gastrica di questi crostacei. Le funzioni assolte da questo *set* di neuroni sono identificabili in sottocircuiti dedicati, per i quali s'intende ovviamente sottoinsiemi di neuroni, le cui strutture non sono interamente separabili ma condividono alcuni neuroni. In particolare, questi stessi neuroni, grazie alla neuromodulazione, cambiano comportamento quando coinvolti da un particolare circuito piuttosto che da un altro. I tempi di variazione dei comportamenti, ossia dell'attivazione di un differente sottocircuito neurale, sono confrontabili con quelli della stimolazione da parte dei neuromodulatori che, nel funzionamento proprio del STG, è prodotta dall'interazione del crostaceo con l'ambiente. Da notare ancora, il ruolo degli stessi neuroni già attivi nel gestire le successive risposte alla neuromodulazione: un circuito in funzione può infatti gestire anche la sua risposta ai neuromodulatori in termini di auto-spegnimento e attivazione di altri circuiti neurali con scopi differenti. In definitiva, lo stato successivo della rete dipende sia dallo stato presente che dall'input neuromodulatorio.

La stessa Bargmann (2012) analizza le caratteristiche principali del connettoma di *C. Elegans*, un verme nematode fasmidario, fornitoci da lavori scientifici che risalgono al 1986, al fine di definire i limiti di un'indagine di tipo strutturale, in favore della strada, oggi ancora aperta, dello studio funzionale attraverso l'analisi dei meccanismi neuromodulatori.

Nel *C. Elegans* la prima caratteristica che salta all'occhio umano è come da una così semplice anatomia possa venir fuori una tale complessità. Alcune caratteristiche sono facilmente comprensibili osservando la sua anatomia, ad esempio, circa un terzo dei neuroni hanno specializzazioni indicanti una funzione sensoria, un altro terzo ha sinapsi all'interno di muscoli che li identifica come neuroni motori, e l'ultimo terzo ha sia input che output in abbondanza indicanti un ruolo nell'integrazione. Il diagramma delle connessioni, in generale, ha una forma gerarchica; i neuroni sensori sono più presinaptici che postsinaptici, per i neuroni motori è vero l'opposto. Alcuni gruppi di neuroni sono densamente connessi, suggerendo funzionalità in comune. Eppure, ad un livello più profondo, il connettoma era e resta difficile da leggere. I neuroni sono connessi tra loro, a volte iperconnessi, tanto che è possibile rintracciare il collegamento di ogni neurone a qualsiasi altro neurone in tre passaggi sinaptici. La modellazione ha pertanto fallito nel generare un'ipotesi che spieghi il comportamento generale del connettoma. La spiegazione teorica è molto semplice: non c'è, in effetti, un unico modo di leggere l'intero diagramma delle connessioni.

Un neurone, un comportamento Un'analisi dei neuroni coinvolti nel movimento ha riportato che tale funzionalità, ad esempio, è associata non tanto a singoli neuroni, in modo qualitativo, ma a funzioni neurali quantitative e distribuite. Ciò è stato possibile tramite lo studio di neuroni isolati e cresciuti in colture dedicate, agendo con opportune stimolazioni.

Un comportamento, più circuiti La profonda eccezione alla regola "un neurone, una funzione" è stata scoperta caratterizzando i comportamenti sotto differenti condizioni. Ad esempio, nel *C. Elegans*, lo scansare dell'odore repulsivo dell'ottanolo in particolari concentrazioni può essere generato da due differenti *set* di neuroni. Negli animali ben nutriti, questo comportamento è mediato dai neuroni nocicettivi ASH, ma dopo un'ora di digiuno, viene distribuito tra i neuroni nocicettivi ASH, AWB e ADL, rivelando un cambiamento nella composizione del circuito. Risultati iniziali hanno mostrato che lo stato di sazietà poteva essere simulato con della serotonina, un trasmettitore associato nel *C. Elegans* ai comportamenti relativi alla nutrizione. Ulteriori analisi hanno poi rivelato che dopamina, tiramina e octopamine, così come numerosi neuropeptidi fungano da input neuromodulatori. Queste ammine e peptidi sono prodotte da una varietà di neuroni e interagiscono in relazioni antagoniste mutue. Ad una prima approssimazione, i neuromodulatori sembrano scambiare i circuiti tra due stati di funzionamento alternativi: uno condotto dal ASH solamente e l'altro da ASH, AWB e ADL. Le azioni di questi modulatori sono in ogni caso molecolarmente distribuite nel sistema nervoso.

Un altro esempio lo vediamo per come il nutrirsi influenzi il circuito dedicato all'aerotassi, e lo stesso nutrimento è regolato da una modulazione di secondo ordine. L'aerotassi è più vigorosa negli animali affamati che in quelli ben nutriti, in quanto entrano in gioco più neuromodulatori. Un input neuropeptidico mediato dal recettore npr-1 accoppiato a proteine G è un importante segnale che impone una regolazione sul circuito di nutrizione; un'altra è il peptide relativo alla TGFbeta. Questi input di nutrizione si contrappongono ad un altro gruppo di neuroni che potenziano l'aerotassi, anche se solo una piccola parte di questi è connessa con la rilevazione diretta dell'ossigeno. Il circuito d'ossigenazione, d'altra parte, altera la nutrizione degli animali in ipossia (mancanza di ossigeno) che pertanto li rende insensibili alla presenza di cibo.

Un neuromodulatore, più comportamenti Il recettore neuropeptidico npr-1 che influenza l'aerotassi, regola anche un secondo comportamento, ovvero l'aggregazione di più animali in gruppi di caccia. L'aggregazione è innescata da un numero di neuroni sensoriali, che include l'ASH e i neuroni ossigeno-sensibili URX, i quali attuano una regolazione della sensibilità al feromone. Sorprendentemente, npr-1 può influire sulla scelta tra due diversi comportamenti avviati dallo stesso neurone sensorio quali l'aggregazione e lo schivare le

sostanze nocive. Quest'ultima è realizzata dal circuito ASH che agisce sui circuiti motori per indietreggiare. ASH gestisce quindi due comportamenti principali: una schivata, senza tener conto della neuromodulazione, e l'aggregazione, ma solo quando il livello di npr-1 è basso.

Implicazioni dirette di questi studi L'informazione che scorre nel connettoma di *C. Elegans* è mediata dai neuromodulatori, o meglio, dipende dai loro stati. Il diagramma delle connessioni racchiude il potenziale per manifestare comportamenti multipli, ma solo un *set* di quei comportamenti è accessibile in un dato momento. Questi risultati indicano che la neuromodulazione seleziona uno o più specifici *set* di sinapsi funzionali rispetto ad un numero più ampio anatomicamente definito. Inoltre il connettoma potrebbe non essere esattamente così come è stato delineato, proprio perché i neuromodulatori vengono rilasciati a livello extrasinaptico e possono costituire essi stessi dei collegamenti tra più circuiti anatomicamente separati. Il connettoma del *C. Elegans* consiste di 302 neuroni, ma il genoma codifica oltre 200 neuropeptidi, suggerendo che il potenziale per la neuromodulazione sia considerevole. Con questi numeri non si può non supporre che molte funzioni dei circuiti neurali siano *multiplexate* proprio da neuromodulatori.

In generale, come già accennato precedentemente, Bargmann (2012) si aspetta che qualsiasi rete neurale biologica risulti sovraconnessa e che l'eccesso di connessioni non sia casuale o appreso, ma rappresenti in realtà dei circuiti latenti, attivabili di volta in volta per gestire differentemente il flusso d'informazione.

1.3 Una possibile soluzione in termini di programmabilità

Sulla base di quanto detto finora, non è da escludere un'altra possibile soluzione per lo sviluppo di strutture neurali biologiche in grado di esibire funzioni multiple. L'idea che i tessuti neurali siano veramente dotati di quell'effettiva "programmabilità" a cui spesso, in letteratura, molti autori alludono solo metaforicamente o, comunque, senza una possibile soluzione interpretativa, potrebbe rivelarsi vincente e addirittura fondarsi proprio sui meccanismi di neuromodulazione o di doppia rete WT/VT descritti nel paragrafo precedente.

Donnarumma *e altri* (2010) hanno rivisitato l'idea di controllare il comportamento di una rete neurale attraverso i suoi input, quello "classico", più un input ausiliare a cui è affidata la codifica della struttura in termini di pesi sinaptici, che è parte di una lunga storia della quale possiamo isolare due tipi di approccio principali. Nel primo, gli input ausiliari sono in grado di modulare il comportamento della rete, sebbene non codifichino

la struttura della rete modulata (Jordan, 1986; Touretzky, 1990; Schmidhuber, 1992; Ito e Tani, 2004; Nishimoto *e altri*, 2008). Molti lavori con questo tipo di approccio hanno quasi completamente tralasciato ogni plausibilità biologica. Nel secondo approccio Schneider e Oliver (1991); Giles *e altri* (1992); Noelle e Cottrell (1994); Hochreiter *e altri* (2001); Prokhorov e Tyukin (2002), invece, gli input ausiliari hanno l'abilità effettiva di programmare la rete, specificando la sua struttura. In molti di questi lavori il concetto di fondo è stato di avere sistemi moltiplicativi che consentano di fornire i valori dei pesi di una rete attraverso degli input ausiliari. In tutti questi lavori però la plausibilità biologica delle reti neurali artificiali non è stata tenuta in conto, con l'eccezione del recente modello di *spiking* neurale di Eliasmith (Eliasmith, 2005) in grado di codificare dinamicamente trasformazioni nell'attività di certi livelli neurali, facendo sì che differenti circuiti implementino le trasformazioni specificate.

Nella letteratura sulle reti neurali, sia artificiali che biologiche, anche la nozione di controllo, strettamente connessa a quella di programmabilità, è divisa in due branche. Una comprende i lavori di quegli autori citati all'inizio del capitolo (Oztop e Arbib, 2002; Sauser e Billard, 2006; Hurley, 2008; Roy, 2008) tra i quali forse emerge il lavoro di Hurley (Hurley, 2008, p. 42), ovvero quello nel quale al controllo nel modello è data una descrizione funzionale che deliberatamente si astrae dai dettagli di implementazione neurale. L'altra, invece, è focalizzata sull'implementazione e comprende i lavori degli autori appartenenti al secondo approccio su menzionato. Queste due branche sembrerebbero essere disgiunte ma, grazie al lavoro di (Donnarumma *e altri*, 2012), possiamo notare che le seconde, in effetti, sembrano proprio completare il lavoro delle prime.

Donnarumma *e altri* (2012) hanno, quindi, proposto un'architettura neurale programmabile, biologicamente plausibile a livello neurale che ha alcune soluzioni in comune al modello di Eliasmith (2005), per il quale i criteri di robustezza, temporizzazione e approssimazione sono trattati esplicitamente così da soddisfare i requisiti (a), (b) e (c) per la programmabilità che saranno illustrati nel capitolo 2.

La similitudine con la programmazione è sorprendente. L'ipotesi di programmabilità non sarà proprio la soluzione adottata nel biologico, ma può fornire un buon esempio di come la natura realmente operi, a prescindere dal tipo sistema fisico sviluppatosi in natura (neuromodulazione, trasmissioni miste WT/VT, ecc.), e dimostrare come il cervello possa implementare effettivamente delle proprietà computazionali, idea che ha da sempre stuzzicato tutti i più grandi pensatori.

Quindi, una possibile chiave interpretativa al problema presentato nel paragrafo 1.1 verte sulla possibilità che le strutture neurali siano dotate di programmabilità. La proprietà di programmabilità di una rete, come spiegazione dei fenomeni di cambiamento di comportamento rapidi e reversibili, sembrerebbe non essere in disaccordo con le proposte

già presenti in letteratura che, in taluni casi, sono anche state verificate sperimentalmente (§ 1.2). Inoltre, risulterebbe una possibile spiegazione di più alto livello di come le strutture neurali siano effettivamente utilizzate dalla natura.

In questa tesi, nell'ambito di un "toy problem", ci proponiamo di comprendere se circuiti neurali dotati della proprietà di programmabilità abbiano un qualche vantaggio rispetto ai circuiti neurali privi di tale proprietà, nel compito di apprendere comportamenti multipli, ovvero più funzioni gestite dalla stessa architettura neurale.

Capitolo 2

Reti Neurali Programmabili

Nel capitolo precedente abbiamo esposto il problema nel biologico di fenomeni di cambiamento di comportamento rapidi e reversibili. Il nostro lavoro si concentra adesso sulla possibilità di fornire un modello artificiale, ma anche biologicamente plausibile, che ne rappresenti una possibile soluzione. In questo capitolo, allora, dopo una breve panoramica sul neurone biologico ed il suo ruolo nel sistema nervoso centrale, presentiamo la rete neurale artificiale che più si presta a modellarne il funzionamento, mantenendo una plausibilità biologica. Successivamente, proponiamo un metodo per dotare una rete neurale artificiale biologicamente plausibile della proprietà di programmabilità.

2.1 Sulla natura delle reti neurali biologiche

Le cellule neurali, le unità fondamentali di un cervello, sono relativamente semplici nella loro morfologia. Anche se un cervello umano contiene un numero straordinario di queste cellule, dell'ordine di 10^{11} neuroni, che possono essere classificate almeno in un migliaio di tipi differenti, tutte le cellule nervose condividono la stessa architettura. La complessità del comportamento umano dipende meno dalla specializzazione delle singole cellule nervose e più sul fatto che molte di queste cellule insieme formino precisi circuiti anatomici e che le proprietà della rete emergano dal modo in cui queste cellule sono interconnesse tra loro. Siccome da pochi semplici principi d'organizzazione può derivare una complessità considerevole, è possibile apprendere tantissimo su come il sistema nervoso produca i comportamenti a partire da alcune funzionalità di base:

- il meccanismo tramite il quale i neuroni producono un segnale;
- gli schemi di connessione tra cellule nervose;
- la relazione dei diversi schemi di interconnessione con i differenti tipi di comportamenti;

- i modi in cui i neuroni e le loro connessioni sono modificati dall'esperienza.

Un tipico neurone ha quattro regioni morfologicamente definite: il corpo cellulare, i dendriti, l'assone e i terminali presinaptici (fig. 2.1.1). Ciascuna di queste regioni ha ruoli distinti nella generazione dei segnali e nella comunicazione tra cellule nervose (Kandel *e altri*, 2000).

Il corpo cellulare, anche detto soma, è il centro metabolico della cellula; contiene il nucleo, che conserva i geni della cellula, così come il reticolo endoplastico nel quale vengono sintetizzate le proteine cellulari. Il corpo cellulare dà generalmente luogo a due tipi di processi: lo sviluppo di svariati dendriti corti e di un lungo assone tubolare.

I dendriti si diramano ad albero e costituiscono l'apparato principale per la ricezione dei segnali provenienti dalle altre cellule nervose. L'assone, invece, tende ad estendersi lontano dal corpo cellulare ed è il condotto principale per portare il segnale agli altri neuroni. Un assone può infatti convogliare segnali elettrici per distanze che vanno da 0.1 mm a 3 m. Questi segnali elettrici, chiamati potenziali d'azione, sono impulsi on-off rapidi e transitori, con un'ampiezza di 100 mV e una durata di circa 1 ms. I potenziali d'azione sono generati all'altezza di una regione d'avvio chiamata poggio assonico (il segmento iniziale dell'assone) e da là sono condotti lungo l'assone senza distorsione o errore ad una velocità di 1-100 m/s. L'ampiezza del potenziale d'azione in viaggio per l'assone rimane costante perché il potenziale d'azione è un impulso on-off che viene rigenerato a intervalli regolari lungo l'assone. Pertanto, l'informazione portata dal potenziale d'azione non è determinata dalla forma del segnale ma dal percorso lungo il quale il segnale viaggia. Tramite questi segnali il cervello riceve, analizza e trasferisce informazione. Per incrementare la velocità di trasporto del segnale, alcuni assoni sono avvolti in uno strato di mielina. La copertura mielinica è interrotta ad intervalli regolari dai nodi di Ranvier, gli *spot* dove il potenziale d'azione viene rigenerato. Alla fine dell'assone, il suo tubolare si divide in sottili ramificazioni che formano i siti di comunicazione con gli altri neuroni.

Il punto in cui due neuroni comunicano è conosciuto come sinapsi. La cellula nervosa che trasmette il segnale è chiamata cella presinaptica, la cellula ricevente è invece chiamata postsinaptica. La cellula presinaptica trasmette segnali dalle parti rigonfie finali dei rami assonici, chiamati terminali presinaptici. In ogni caso, una cella presinaptica non tocca realmente o comunica anatomicamente con la cellula postsinaptica in quanto le due cellule sono separate da uno spazio detto, fessura sinaptica. Molti terminali sinaptici finiscono sui dendriti dei neuroni postsinaptici, ma i terminali possono anche finire sul corpo cellulare o, più di rado, all'inizio o alla fine dell'assone della cellula ricevente (fig. 2.1.1).

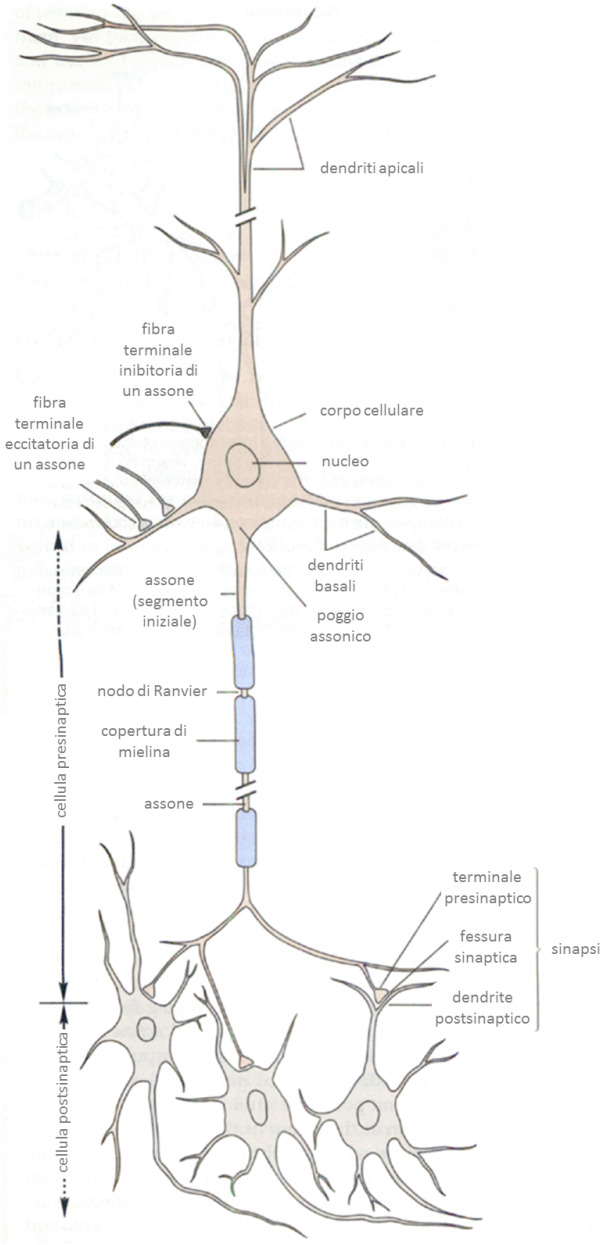


Figura 2.1.1: *Struttura di un neurone. Molti neuroni nel sistema nervoso dei vertebrati hanno parecchie caratteristiche principali in comune. Il corpo cellulare contiene il nucleo che conserva il materiale genetico e dal quale sono prodotti due processi cellulari: assoni e dendriti. Gli assoni, elementi trasmittenti del neurone, possono variare molto in lunghezza: alcuni possono estendersi per più di 3 m all'interno del corpo, altri sono invece molto sottili (tra 0,2 e 20 µm in diametro). Molti assoni sono isolati dalla mielina che è interrotta ad intervalli regolari dai nodi di Ranvier. Il potenziale d'azione, il segnale condotto dalle cellule, è generato all'altezza del poggio assonico, il segmento iniziale, oppure poco più avanti, all'altezza del primo nodo di Ranvier. Le ramificazioni assoniche di un neurone (il neurone presinaptico) trasmettono segnali ad un altro neurone (la cellula postsinaptica) in un sito chiamato sinapsi. Le ramificazioni di un singolo assone possono formare sinapsi anche con altri 1000 neuroni. Mentre l'assone è l'elemento di output del neurone, i dendriti (apicali e basali) sono gli elementi di input e, insieme con il corpo cellulare, ricevono i contatti cellulari da altri neuroni. (Immagine tratta da Kandel* e altri*, 2000)*

2.2 CTRNN come modelli di reti neurali biologiche

Le **Reti Neurali Ricorrenti a Tempo Continuo (CTRNN)** che intendiamo utilizzare sono reti di neuroni che si ispirano alla biologia, descritte dalla seguente equazione generale (Hopfield e Tank, 1986; Beer, 1995):

$$\tau_i \frac{dy_i}{dt} = -y_i + \sigma\left(\sum_{j=1}^{N} w_{ij} y_j + \theta_i + I_i^e(t)\right) \quad i \in \{1, \ldots, N\} \tag{2.2.1}$$

dove N è il numero di neuroni della rete, e per ogni neurone i:

- τ_i è la *costante di tempo della membrana*;
- y_i è la *frequenza di sparo media (nel tempo)*;
- θ_i è la *soglia* o *bias*;
- $\sigma(x)$è la *funzione di attivazione logistica standard*, che è $\sigma(x) = 1/(1 + e^{-x})$;
- $I_i^e = \sum_{j=N+1}^{N+L} w_{ij} x_j$ è una *corrente di ingresso esterna* in arrivo da L sorgenti esterne x_j;
- w_{ij} è l'*efficacia sinaptica* (*peso*) della connessione in arrivo dal neurone j, o dalla sorgente esterna x_j, al neurone i.

Sebbene questo modello sia più elementare di altri modelli più accurati presenti in letteratura, una CTRNN è un'astrazione molto invitante delle reti biologiche perché:

- è computazionalmente poco dispendiosa da implementare (Kier *e altri*, 2006);
- è matematicamente trattabile (Beer, 1995);
- è un approssimatore di dinamiche universale (Funahashi e Nakamura, 1993);
- i suoi neuroni possono avere un'interpretazione biologica diretta (Vedi ad esempio Dunn *e altri*, 2004; Kier *e altri*, 2006).

L'ultimo punto è particolarmente importante per i nostri scopi in quanto afferma che è possibile sviluppare architetture CTRNN come modelli plausibili di circuiti neurali biologici. Inoltre, la proprietà di universalità delle CTRNN afferma che assegnata una soluzione traiettoria di un generico sistema dinamico continuo, esiste sempre una CTRNN con una data struttura tale da fornire una soluzione che la approssimi con qualsiasi grado di precisione (Funahashi e Nakamura, 1993).

Un ulteriore chiarimento, per quanto ci riguarda, va dato sul tipo di segnali di input che utilizzeremo. Se consideriamo gli input di un gruppo di celle neurali come provenienti da segnali esterni sensoriali o da altri gruppi di neuroni, ci sembra più plausibile biologicamente restringere questi alle sole correnti di ingresso esterne $I_i^e(t)$.

Per quel che riguarda gli output, invece, le scelte più comuni sono le traiettorie, soluzioni dell'equazione (2.2.1), oppure gli stati stabili della rete - a volte definite come *attrattori computazionali* (Hopfield e W., 1985; Siegelman *e altri*, 2001). Nel caso particolare in cui l'equazione (2.2.1) sia globalmente stabile, questa avrà un unico punto fisso stabile, indipendentemente dalle condizioni iniziali, considerando come output lo stato stabile della rete, la CTRNN implementa la seguente funzione:

$$\boldsymbol{f} \,:\, \boldsymbol{x} \in \mathbb{R}^L \rightarrow \boldsymbol{f}(\boldsymbol{x}) \equiv \bar{\boldsymbol{y}}(\boldsymbol{x}) \in \mathbb{R}^N$$

dove $\bar{\boldsymbol{y}}(\boldsymbol{x})$ è il punto fisso relativo all'input x. In generale, la scelta degli attrattori computazionali di cui sopra implica che gli ordini di grandezza dei tempi di variazione degli input siano grandi rispetto all'ordine di grandezza del tempo - determinato dalla costante di tempo τ_i - della CTRNN.

Per concludere, osserviamo che, per quanto questo modello non includa alcuna caratteristica della trasmissione di volume, limitandosi cioè alla sola trasmissione cablata, la scelta non ci impedisce di sviluppare un'ipotesi completa su un meccanismo di apprendimento comportamentale, quale quello che presenteremo più avanti.

2.3 Requisiti di programmabilità

Grazie al lavoro di (Garzillo e Trautteur, 2009) siamo in grado di definire dei requisiti di programmabilità per una classe di sistemi di processo simbolico, non necessariamente discreta, sia artificiale che biologica:

(a) Esiste un'effettiva codifica della struttura dei singoli sistemi in pattern di input, output e variabili interne.

(b) I codici forniti da tale codifica possono essere applicati a specifici sistemi della classe, designati come interpreti (o universali), così che l'interprete realizzi il comportamento del sistema codificato.

(c) I codici possono essere processati dai sistemi della classe al pari con gli input, gli output e le variabili interne.

Allora, circuiti cerebrali locali o reti di circuiti cerebrali locali, dotati di programmabilità, senza cambiare la connettività o l'efficacia associata con le connessioni sinaptiche, possono

esibire cambiamenti qualitativi di comportamento al volo, causati e controllati da input ausiliari (programmazione) - condizioni (b) e (c) - i quali codificano differenti circuiti cerebrali - condizione (a).

Per lo scopo prefissatoci possiamo utilizzare delle Reti Neurali Artificiali (ANN) biologicamente plausibili, ovvero partire dalle CTRNN descritte nel paragrafo 2.2. Questo tipo di ANN biologica deve ricevere degli input ausiliari che possano controllare il comportamento della rete stessa.

Donnarumma *e altri* (2012) hanno proposto, sulla base di tale premessa, una architettura neurale programmabile, biologicamente plausibile a livello neurale, che ha alcune soluzioni in comune al modello di Eliasmith (Eliasmith, 2005), per il quale i criteri di robustezza, temporizzazione e approssimazione sono trattati esplicitamente così da soddisfare i requisiti (a), (b) e (c) per la programmabilità.

2.4 Reti neurali a pesi fissi programmabili

Per ottenere un'architettura ANN a pesi fissi programmabile, in accordo con i punti della sezione precedente, ricordiamo che l'input ai neuroni biologici è modellato come somme di prodotti tra i segnali di output che giungono da altri neuroni attraverso connessioni pesate e i pesi associati a queste connessioni. Pertanto l'evoluzione di una rete è confinata tra le somme dei prodotti tra pesi e segnali d'uscita. Il sistema utilizzato da (Donnarumma *e altri*, 2012) consiste nell'*estrarre* l'operazione di moltiplicazione usando delle sottoreti che producano i risultati di tali operazioni di moltiplicazione tra output e pesi. In tal modo, i pesi diventano un input ausiliare alla rete alla quale siano state aggiunte le sottoreti di moltiplicazione. Un'architettura neurale del genere viene allora creata con due tipi di linee di input: le linee di **input ausiliare (o di programmazione)** e le linee di input standard. Notiamo che alle linee di input ausiliare viene fornito un codice, o programma, che descriva la rete originale; tale azione ricorda il codice di una macchina virtuale fornito ad un'architettura computazionale standard oppure i numeri di Gödel forniti ad una macchina di Turing universale.

Per chiarire meglio il nostro approccio, supponiamo che **mul** sia una ANN ideale che effettui una moltiplicazione tra i due valori di input a e b. La rete mul è composta da M neuroni, con i pesi appropriati, e uno di questi M neuroni è il neurone di output k, con il valore di uscita y_k tale che $y_k = a \cdot b$.

Data una generica ANN G, composta di N neuroni, consideriamo due qualsiasi neuroni i e j legati da una connessione con peso w_{ij}. E' allora possibile costruire una **Rete Neurale Programmabile (PNN)** G_{mul} che duplichi esattamente il comportamento di G attraverso la rete di moltiplicazione mul, secondo i seguenti passi:

1. Reindirizzamento dell'output del neurone j come input a di mul.
2. Settaggio dell'input di b di mul a $p = w_{ij}$.
3. Reindirizzamento dell'output del neurone k di mul come input del neurone i, con peso pari ad 1.

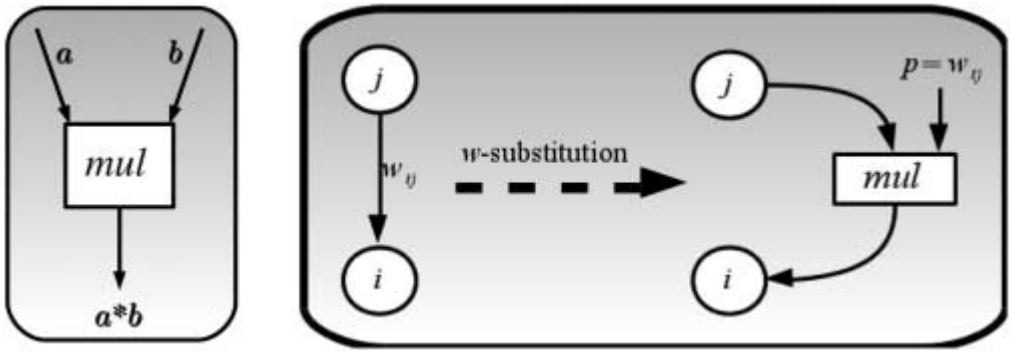


Figura 2.4.1: *L'*estrazione *della moltiplicazione. A sinistra è mostrata la rete mul che prende due input a e b e li moltiplica. A destra la* w-sostituzione *della connessione*w_{ij} *tra i neuroni i e j attraverso la rete mul. (Immagine tratta da Donnarumma* e altri*, 2012)*

Supponiamo per il momento che la ANN mul produca esattamente il suo output atteso e senza alcun ritardo. Allora, il comportamento della PNN G_{mul}, ristretto ai neuroni i e j, è identico al comportamento della rete originale G perché per qualsiasi valore di output del neurone presinaptico i, il segnale postsinaptico ottenuto attraverso la sottorete mul è identico a quello ottenuto nella rete originale G.

Notiamo adesso che se si fa l'ipotesi classica che i neuroni della ANN abbiano valori di output contenuti nell'intervallo $(0, 1)$ e che i pesi w_{ij} abbiano valori compresi in un intervallo (min, max), è possibile riformulare la procedura precedente in questo modo: sostituiamo la connessione w_{ij} con una nuova connessione con peso pari a min, in parallelo ad una rete di moltiplicazione mul che riceve come input standard $y_j \in (0, 1)$ e come input di programmazione $p = (w_{ij} - min)/(max - min) \in (0, 1)$, collegata al neurone i con una connessione di peso pari a $(max - min)$, come mostrato nella . Adesso che gli input della rete mul apparterranno sempre all'intervallo $(0, 1)$, abbiamo bisogno solamente di una rete mul che esegua un prodotto di variabili appartenenti allo stesso intervallo $(0, 1)$. D'ora in avanti ci riferiremo a tale operazione come ad una ***w-sostituzione***, ovvero la sostituzione di una connessione pesata w_{ij}.

Adesso, data una generica ANN G composta da N neuroni, con input $\boldsymbol{x} = [x_{N+1}, \ldots, x_{N+L}]$ e pesi $w_{ij} \in (min, max)$, con $i \in \{1, \ldots, N\}$ e $j \in \{1, \ldots, N + L\}$, possiamo costruire una PNN G_{mul} applicando uniformemente una *w-sostituzione*.

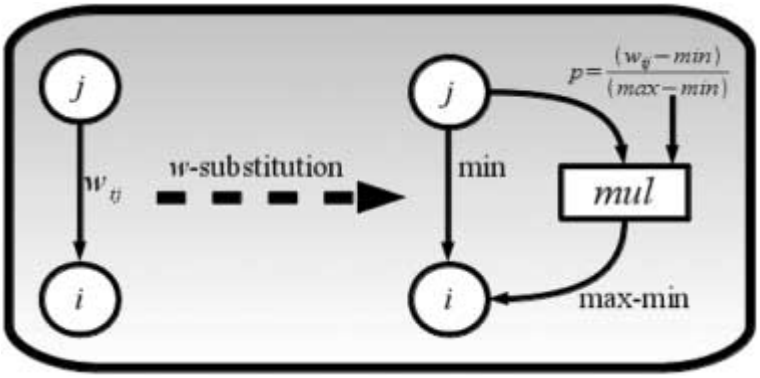


Figura 2.4.2: *Procedura di w-sostituzione. La procedura di w-sostituzione con i pesi* $w_{ij} \in (min, max)$. *Per ogni peso è usata una rete mul alla quale viene fornito il corrispondente programma* $p = (w_{ij} - min)/(max - min)$. *(Immagine tratta da Donnarumma* e altri*, 2012)*

In questo modo si ottiene una PNN G_{mul} composta da $N + N \cdot M \cdot (N + L)$ neuroni, con dati di input $\boldsymbol{x} = [x_{N+1}, \ldots, x_{N+L}]$ e input ausiliari $\boldsymbol{p} = [p_1, \ldots, p_{N(N+L)}]$, che esibisce un comportamento pari a quello della rete G.

Notiamo inoltre che questa PNN G_{mul}, creata a partire da una ANN di N neuroni, non riproduce solamente l'esatto comportamento della rete G di partenza, ma qualsiasi comportamento di una ANN di N neuroni, dati i valori $\boldsymbol{p}$ corrispondenti. Allora la PNN G_{mul} possiede programmabilità perché risponde esattamente alle condizioni (a), (b) e (c) di programmabilità formulate nella sezione precedente. Infatti i valori $\boldsymbol{p}$ codificano in modo effettivo una ANN di N neuroni, con L input i cui pesi sono decodificati tramite:

$$w_{ij} = (max - min) \cdot p_h + min \tag{2.4.1}$$

Inoltre il codice $\boldsymbol{p}$ ha valori nello stesso intervallo delle variabili dei neuroni ordinari che possono essere processati insieme agli input, output e variabili interne ordinari. In questo senso gli input ausiliari $\boldsymbol{p}$ giocano il ruolo dei programmi che sono interpretati dalle PNN G_{mul}.

2.4.1 Implementazione della rete mul

Le CTRNN presentate precedentemente possono essere utilizzate per creare una rete mul^* che approssimi la rete *mul* ideale appena presentata. Per ottenere la mul^* utilizziamo un algoritmo di apprendimento di *Differential Evolution* Price (1999). Questo algoritmo era pensato per essere eseguito su popolazioni di 30 piccole CTRNN con $\tau_i = \tau^* = 1$.

Senza dubbio, bisogna tenere in conto gli aspetti temporali di questa realizzazione perché la rete mul^* con attrattori computazionali ha un tempo di attesa finito per rag-

giungere i valori asintotici, a paragone con l'attesa nulla delle sottoreti *mul* ideali richieste dalla w-sostituzione definita precedentemente. Per approssimare questo risultato abbiamo scelto *mul**con costanti di tempo $\tau^* \ll \tau_i$, dove τ_i sono le costanti di tempo della rete originale a cui la w-sostituzione fa riferimento.

2.5 Implementazione di reti neurali programmabili

Riportiamo qui il comportamento di alcune reti neurali programmabili perché sia d'esempio su quanto teorizzato nel precedente paragrafo. Ricordiamo che le PNN a peso fisso G_{mul^*} usano le sottoreti moltiplicative mul^* che approssimano una sottorete moltiplicativa ideale.

Forniremo alcuni esempi di reti artificiali nelle quali, per ogni neurone i, la soglia corrispondente θ_i può essere interpretata come la presenza di una connessione di peso θ_i e valore in input fisso pari ad 1. Di conseguenza, le soglie saranno trattate come pesi addizionali. Allora, data una generica ANN G, composta di N neuroni che hanno come input $\boldsymbol{x} = [x_{N+1}, \ldots, x_{N+L}]$, la relativa PNN G_{mul^*} è stata ottenuta per applicazione della *w-sostituzione* di k_w connessioni e k_θ soglie di G, per $0 < k_w < N(L+1)$ e $0 \leq k_\theta < N$. Pertanto, abbiamo una PNN G_{mul} di input programmabili k_w e k_θ.

Alcuni esperimenti sono stati eseguiti (Donnarumma *e altri*, 2012) su piccole CTRNN che sono state integrate con il metodo di Eulero. Il passo di integrazione usato dell'algoritmo di Eulero è stato scelto pari a $\triangle T = 0.1$ e le costanti di tempo dei neuroni sono state settate ai valori di $\tau_i \geq 1$.

Notiamo che negli esempi che saranno mostrati non è mai stato fatto uso di algoritmi di apprendimento: l'unica fase di apprendimento è stata quella relativa alla costruzione delle mul^*, così come descritta nel paragrafo 2.4.1.

Tutti gli esperimenti in questa tesi sono stati svolti per costruzione diretta delle reti interpreti con il nostro sistema di *w-sostituzione*. La grandezza delle reti interpreti e l'intervallo di *w-sostituzione* (min, max) sono stati scelti nei diversi esperimenti in relazione alla grandezza dell'intervallo di peso della classe delle reti da simulare.

2.5.1 Reti a singolo neurone

Come un esempio di PNN, cominciamo il nostro studio considerando una rete a singolo neurone con un'autoconnessione. Questa rete può essere descritta dalla seguente equazione differenziale:

$$\tau \frac{dy}{dt} = -y + \sigma\left(wy + \theta + I\right). \tag{2.5.1}$$

Possiamo studiare i comportamenti qualitativi di tale piccola rete (Beer, 1995) in maniera analitica: la variazione dei due parametri w e I modifica il diagramma di fase della rete. Esistono particolari valori dei parametri per il quale il sistema subisce biforcazioni. In particolar modo, avviene un cambiamento qualitativo nei comportamenti quando w passa per il valore 4. L'equazione 2.5.1 mette in evidenza un punto di equilibrio di stabilità globale quando $w < 4$ (figura 2.5.2, riquadri (a) e (c)) e tre punti d'equilibrio, per un intervallo di valori di I, quando $w > 4$ (fig. 2.5.2, riquadri (e) e (g)). In quest'ultimo caso, per valori di I fuori dell'intervallo, l'equazione 2.5.1 rivela un punto di stabilità globale, mentre per I appartenente all'intervallo i due punti di equilibrio più esterni sono stabili e il più interno instabile.

Allora, chiamiamo $NetOne^1$, $NetOne^2$, $NetOne^7$ e $NetOne^8$, quattro differenti reti a singolo neurone con $\theta = 0$, e pesi delle autoconnessioni rispettivamente pari a $w_1 = 1$, $w_2 = 2$, $w_3 = 7$ e $w_4 = 8$ (il numero in apice dei nomi delle reti indicano il valore del peso delle autoconnessioni). Queste quattro reti differiscono per i diversi pesi delle autoconnessioni. Dopodiché, una singola PNN chiamata $NetOne_{mul^*}$ è stata generata scegliendo una delle reti scelte e applicando la *w-sostituzione* usando mul^*. A $NetOne_{mul^*}$ è stato dato un solo input di programma $p_w = (w - min)/(max - min) \in (0, 1)$ con $min = 0.5$ e $max = 8.5$. In figura 2.5.1 si può vedere una schematizzazione di questa *w-sostituzione*.

Notiamo che la peculiarità della rete a pesi fissi $NetOne_{mul^*}$ è che questa può essere programmata in modo da ottenere due tipi di comportamenti qualitativamente differenti scegliendo la relativa p_w.

In figura 2.5.2, i riquadri (a), (c), (e) e (g) mostrano gli equilibri di $NetOne^1$, $NetOne^2$, $NetOne^7$ e $NetOne^8$ in funzione di I, comparate con gli equilibri calcolati numericamente della $NetOne_{mul^*}$ a pesi fissi (ristretti all'output del neurone originale) alimentato, rispettivamente, con gli input ausiliari $p_{w_1} = 0.062$, $p_{w_2} = 0.19$, $p_{w_3} = 0.81$ e $p_{w_4} = 0.94$ (fig. 2.5.2, riquadri (b), (d), (f) e (h)). Quando l'input di programma p_w è uguale a 0.062 o 0.19, la rete $NetOne_{mul^*}$ esibisce un punto di equilibrio di stabilità globale come in $NetOne^1$ e $NetOne^2$. Quando l'input di programma p_w è uguale a 0.81 o 0.94, la rete $NetOne_{mul^*}$ esibisce due punti di equilibrio di stabilità globale nell'intervallo $[I_1, I_2]$ di valori di I, mentre per valori di I al di fuori di questo intervallo, si ha un punto di equilibrio di stabilità globale come in $NetOne^7$ e $NetOne^8$.

In tal senso, $NetOne_{mul^*}$, alimentata con gli input di programma p_{w_1}, p_{w_2}, p_{w_3} e p_{w_4}, si comporta rispettivamente come le quattro reti $NetOne^i$ con pesi w_1, w_2, w_3 e w_4. La singola PNN $NetOne_{mul^*}$ è, pertanto, in grado di eseguire un'implementazione virtuale delle quattro distinte reti $NetOne^i$.

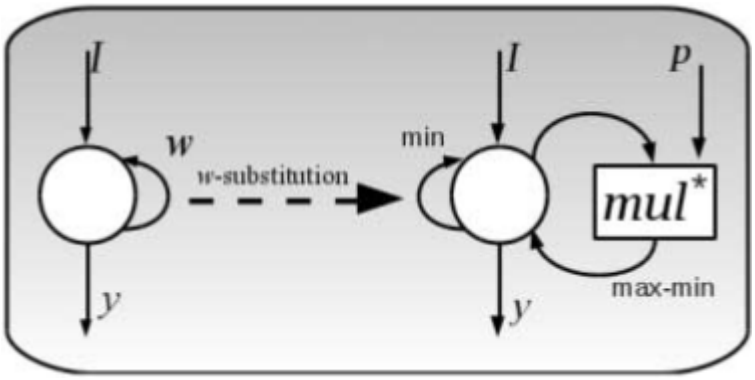


Figura 2.5.1: *La* w-sostituzione *applicata alla rete NetOne (a sinistra) risultante nella produzione di* $NetOne_{mul^*}$ *(a destra). (Immagine tratta da Donnarumma* e altri*, 2012)*

2.5.2 Reti a due neuroni

Nell'esempio appena descritto, abbiamo mostrato come una semplice PNN possa esibire un numero di semplici ma qualitativamente differenti comportamenti senza cambiare la sua struttura quando le linee di input ricevano i programmi appropriati. Vogliamo proporre un secondo esempio nel quale è stato selezionato un *set* di CTRNN a due neuroni che esibissero comportamenti differenti. In particolare, otto tipi di CTRNN che chiamiamo G^1, G^{1lc}, G^3, G^{3lc}, G^5, G^{5lc}, G^7 e G^9. Il numero in apice indica il numero di punti fissi nei corrispondenti diagrammi di fase. Il suffisso "lc" indica la presenza di un ciclo limite. I parametri e il comportamento di queste reti a due neuroni sono schematizzati in tabella 2.1. I pesi w_{12} e w_{21}, così come le soglie θ_1 e θ_2 rimangono uguali in tutte le reti, mentre i pesi w_{11}, w_{22}, w^e_{11} e w^e_{21} cambiano da una rete all'altra.

La PNN G_{mul^*} è stata alimentata con otto gruppi di input di programma $p_{w_{11}}$, $p_{w_{22}}$, $p_{w^e_{11}}$ e $p_{w^e_{21}}$ come riportato in tabella 2.2. Ogni gruppo di input di programma codifica i pesi w_{11}, w_{22}, w^e_{11} e w^e_{21} di una delle reti selezionate G^s, con $s \in \{1, 1lc, 3, 3lc, 5, 5lc, 7, 9\}$.

I risultati (figg. 2.5.3 e 2.5.4) mostrano che le reti G^s e le G_{mul^*} esibiscono un comportamento molto simile quando G_{mul^*} riceve gli input di programma appropriati. In particolare, il numero di punti fissi di stabilità è sempre preservato, mentre i valori di questi punti fissi di stabilità sono molto vicini agli originali. Nel caso della rete G^{1lc}, anche il ciclo limite è preservato (fig. 2.5.3, seconda riga), mentre il ciclo limite sembra sparire per G^{3lc} (fig. 2.5.3, quarta riga) e G^{5lc} (fig. 2.5.4, seconda riga). Inoltre, molti dei punti fissi di instabilità sembrano essere preservati. Notiamo allora che i diagrammi di fase delle reti G^s, comparati con quelli delle reti G_{mul^*} alimentate con gli input di programmazione corrispondenti sono abbastanza sovrapponibili.

Quindi tutti questi esempi mostrano che una singola PNN può effettivamente esibire molti comportamenti qualitativamente distinti, coerentemente con gli input di programma. Di conseguenza, questo suggerisce come i circuiti neurali biologici possano sostenere

la programmabilità e consentire un riuso di aree cerebrali in differenti circuiti o il controllo di esse da parte di altre aree cerebrali durante l'esecuzione di *task* complessi.

Tabella 2.1: *Otto CTRNN a due neuroni che esibiscono comportamenti qualitativamente differenti. La soglia di tutti i neuroni è pari a* -7*. I pesi* w^e_{11} *e* w^e_{12} *sono quelli associati con due connessioni esterne in arrivo, rispettivamente, nei neuroni 1 e 2. Queste due connessioni esterne ricevono lo stesso input* $x_1 = 1$*. Per tutte le reti, i pesi* w_{12} *e* w_{21} *sono uguali, rispettivamente, a 1 e -1.*

	G^1	G^{1lc}	G^3	G^{3lc}	G^5	G^{5lc}	G^7	G^9
w_{11}	3	4.5	5.25	5.5	6.5	6	6.3	6.5
w_{22}	3	4.5	5.25	5.5	5.7	5.5	6.3	6.5
w^e_{11}	5.56	4.72	4	4.19	3.61	3.76	3.61	3.61
w^e_{21}	6.67	5.83	5	5.83	5.17	5.87	4.83	4.72
# **p.f. stabili**	1	0	2	1	2	2	3	4
# **p.f. instabili**	0	0	0	0	1	0	1	1
# **spirali instabili**	0	1	0	1	0	1	0	0
# **p.f. sella**	0	0	1	1	2	2	3	4
# **cicli limite**	0	1	0	1	0	1	0	0

Tabella 2.2: *Input di programma della PNN* G_{mul^*} *In ciascuna colonna sono riportati gli input della* G_{mul^*} *per simulare la corrispondente CTRNN. Questi valori di input sono stati ottenuti dall'equazione 2.4.1 considerando* $min = 2.9$ *e* $max = 6.7$*.*

	G^1	G^{1lc}	G^3	G^{3lc}	G^5	G^{5lc}	G^7	G^9
$p_{w_{11}}$	0.026	0.42	0.62	0.68	0.95	0.82	0.89	0.95
$p_{w_{22}}$	0.026	0.42	0.62	0.68	0.74	0.68	0.89	0.95
$p_{w^e_{11}}$	0.70	0.48	0.29	0.34	0.19	0.23	0.19	0.19
$p_{w^e_{21}}$	0.99	0.77	0.55	0.77	0.60	0.78	0.51	0.48

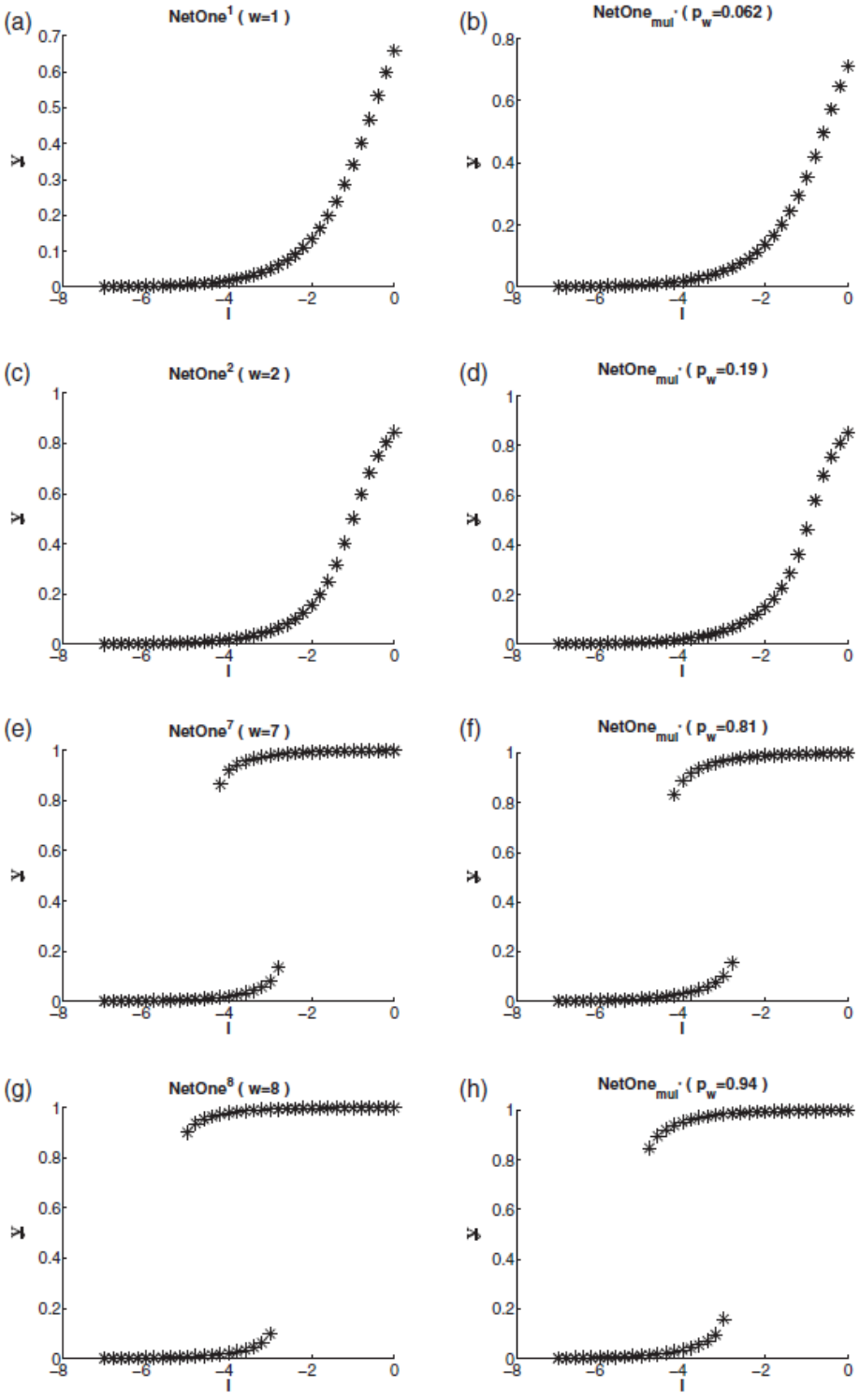


Figura 2.5.2: *Le superfici di stabilità di $NetOne^1$, $NetOne^2$, $NetOne^7$ e $NetOne^8$, rispettivamente nei riquadri (a), (c), (e) e (g), confrontate con le superfici di stabilità di $NetOne_{mul^*}$ (ristretta al neurone originale) alimentata con gli input ausiliari $p_{w_1} = 0.062$, $p_{w_2} = 0.19$, $p_{w_3} = 0.81$ e $p_{w_4} = 0.94$, rispettivamente nei riquadri (b), (d), (f) e (h). Le costanti di tempo τ^* dei neuroni di mul* sono state impostate ad un ordine di grandezza in meno rispetto a quelle di $NetOne^i$. (Immagine tratta da Donnarumma* e altri, *2012)*

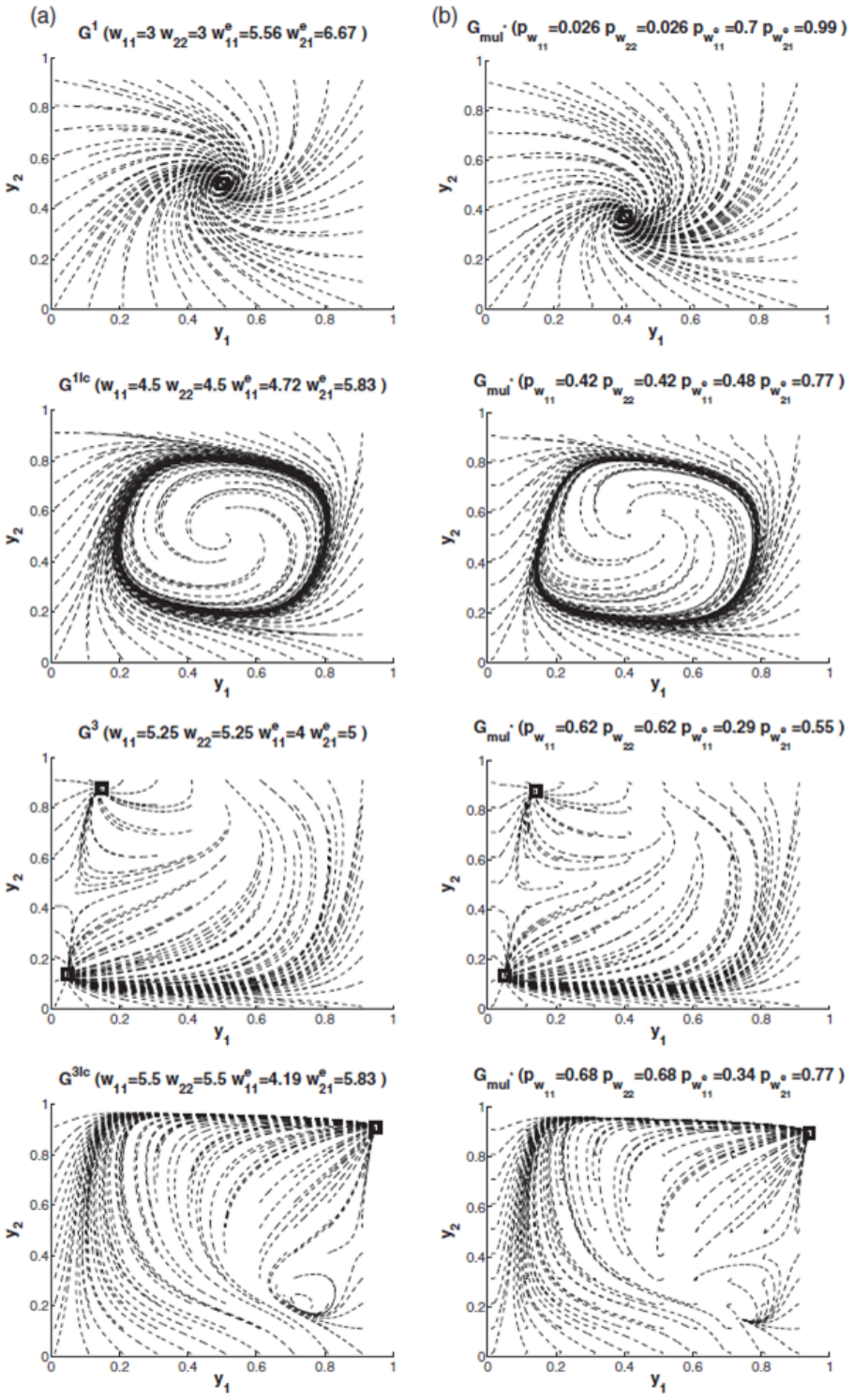


Figura 2.5.3: *Comportamenti della PNN G_{mul^*}. Il riquadro (a) mostra i diagrammi di fase delle quattro distinte reti G^1, G^{1lc}, G^3 e G^{3lc}. Il riquadro (b) mostra i diagrammi di fase della PNN G_{mul^*} quando è alimentata con uno degli input di programma che codifica le reti G^1, G^{1lc}, G^3 o G^{3lc}. I punti fissi di stabilità sono indicati da marcatori quadrati. (Immagine tratta da Donnarumma* e altri, *2012)*

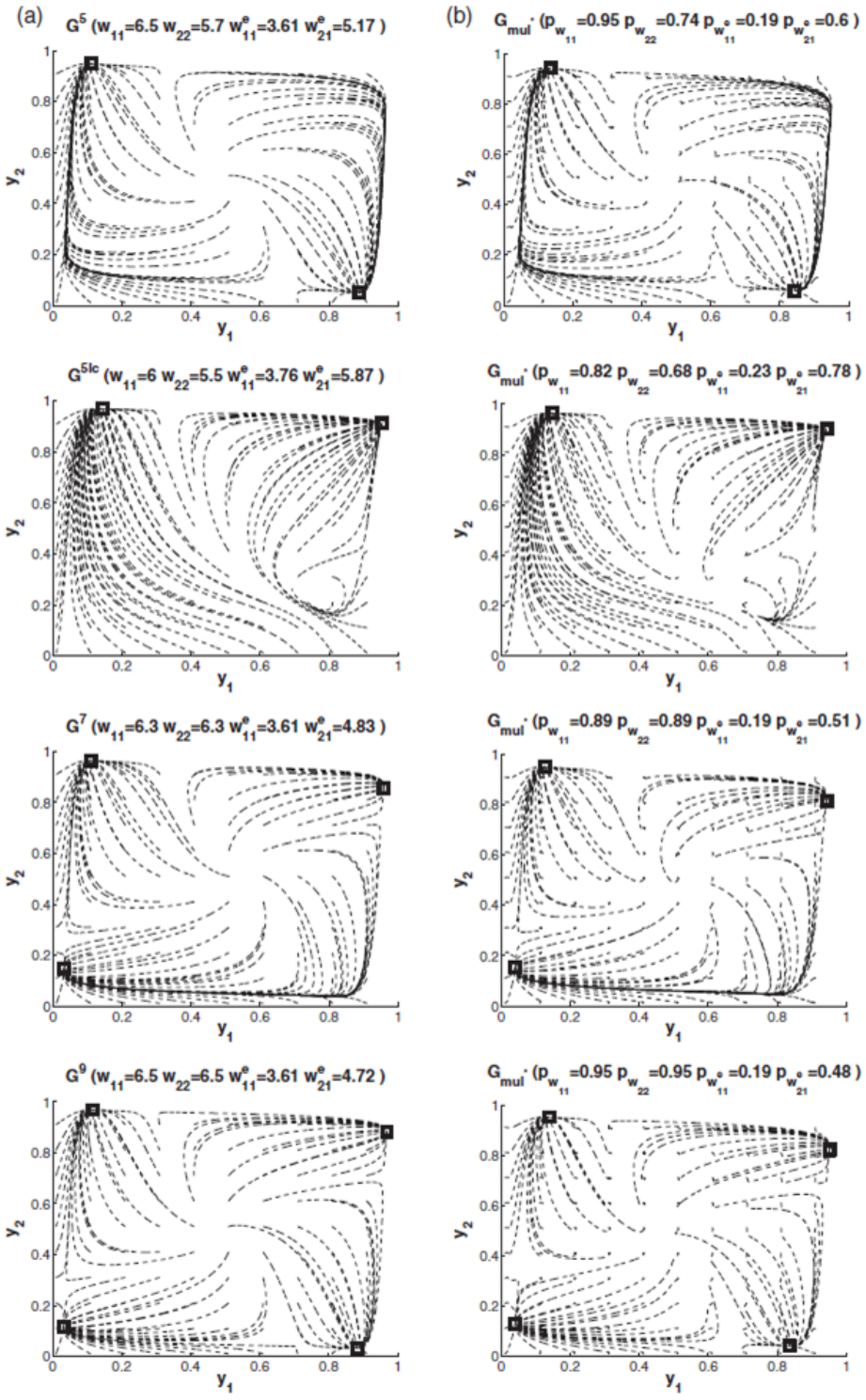


Figura 2.5.4: *Comportamenti della PNN G_{mul^*}. Il riquadro (a) mostra i diagrammi di fase delle quattro distinte reti G^5, G^{5lc}, G^7 e G^9. Il riquadro (b) mostra i diagrammi di fase della PNN G_{mul^*} quando è alimentata con uno degli input di programma che codifica le reti G^5, G^{5lc}, G^7 o G^9. I punti fissi di stabilità sono indicati da marcatori quadrati. (Immagine tratta da Donnarumma* e altri*, 2012)*

2.6 Robustezza delle PNN e problemi di scala temporale

Abbiamo illustrato come sia possibile l'ipotesi di programmabilità delle CTRNN assumendo un tempo di ritardo trascurabile in riferimento all'attività delle sottoreti moltiplicative rispetto alla scala temporale delle CTRNN originali, che è $\tau^* \ll \tau_i$, dove τ_i sono le costanti di tempo della rete originale.

Notiamo che applicando la *w-sostituzione* su una rete G otteniamo una PNN G_{mul^*} composta da due parti: una è il *set* di reti moltiplicative e l'altra consiste dei neuroni originali della rete G. Le scale temporali delle due componenti del sistema dovrebbero differire significativamente perché la programmabilità abbia luogo.

In accordo con quanto detto, è stato testato come i cambiamenti di scala temporale delle due componenti del sistema influenzino l'operatività dell'architettura programmabile (Donnarumma *e altri*, 2012). In particolare, è stata testata la robustezza della PNN per variazioni del rapporto tra le costanti di tempo $r = \tau/\tau^*$, dove τ è la costante di tempo dei neuroni appartenenti alla rete G di partenza e τ^* è la costante di tempo dei neuroni appartenenti alle sottoreti mul^*. L'ipotesi di avere un tempo trascurabile nella stabilizzazione delle sottoreti mul^*rispetto alla scala temporale della rete originale corrisponde a trattare il comportamento di G_{mul^*} come un'approssimazione adiabatica del comportamento di G . Nel caso in cui le scale temporali delle due componenti del sistema non differiscano significativamente, cioè per r piccolo, le dinamiche di G_{mul^*} diventano molto più complesse: il comportamento delle reti mul^* influenza il comportamento complessivo dell'intero sistema in modi inattesi e, in definitiva, l'uso proposto delle sottoreti per fornire le suddette proprietà di programmabilità risulta altresì compromesso.

Riportiamo i risultati di due gruppi di esperimenti per questo studio. Nel primo gruppo è stata usata una CTRNN composta da due nodi, $Net - 2$. Nel secondo, invece, è stata usata una rete composta da cinque nodi, $Net - 5$. In entrambi i casi le reti sono state integrate con il metodo di Eulero e gli esperimenti sono stati condotti nel seguente modo:

1. Sono stati scelti S *set* di pesi e condizioni iniziali per la CTRNN. Ogni peso w_{ij} e condizione iniziale $y_i(0)$ sono state scelte casualmente, rispettivamente, negli intervalli $(min = -10,\ max = 10)$ e $(y_{min} = 0,\ y_{max} = 1)$, così da ottenere S reti G^i, con $i \in \{1, \ldots, S\}$. Le costanti di tempo dei neuroni appartenenti a G^i sono state settate a τ.

2. Sono state scelte tre sottoreti mul^* che differissero solo nelle scale temporali: mul_1^*, mul_2^* e mul_3^*, con costanti di tempo dei neuroni rispettivamente pari a $\tau_1^* = \tau/r_1$, $\tau_2^* = \tau/r_2$ e $\tau_3^* = \tau/r_3$.

3. Per ogni rete G^i, sono state costruite tre PNN $G^i_{mul^*_1}$, $G^i_{mul^*_2}$e $G^i_{mul^*_3}$ applicando la *w-sostituzione* uniformemente su ogni peso e soglia della rete usando, rispettivamente, mul^*_1, mul^*_2 e mul^*_3.

4. Per ogni rapporto $r_k = \tau/\tau^*_k$, con $k \in \{1, 2, 3\}$, sono state messe a confronto le evoluzioni degli output delle reti $G^i_{mul^*_k}$ rispetto alle reti G^i, con $i \in \{1, \ldots, S\}$. Se chiamiamo $\boldsymbol{y}^i(n \cdot \tau)$ e $\tilde{\boldsymbol{y}}^{i,k}(n \cdot \tau)$ i punti delle traiettorie di G^i e $G^i_{mul^*_k}$, ristrette ai neuroni in G^i, al tempo $n \cdot \tau$, allora, ad ogni istante n, possono essere valutate le distanze euclidee:
$$d^{i,k}(n) = \left\| \boldsymbol{y}^i(n \cdot \tau) - \tilde{\boldsymbol{y}}^{i,k}(n \cdot \tau) \right\|.$$

5. In tal modo, la media e la deviazione standard di $d^{i,k}(n)$ su S reti dà una misura di quanto i comportamenti di G e G_{mul^*} differiscano quando G_{mul^*} è ottenuta applicando uniformemente la *w-sostituzione* su G usando sottoreti mul^* agenti su una scala temporale che sia r_k volte più veloce di quella della rete G.

Nei riquadri (a), (b) e (c) della figura 2.6.1 sono mostrate delle traiettorie campionate dei due neuroni di $Net - 2$ e dei due originali neuroni di $Net - 2_{mul^*}$. Il rapporto delle costanti di tempo r assume, rispettivamente, i valori r_1, r_2 e r_3 nei riquadri (a), (b) e (c). Come si può vedere, le traiettorie sono ben preservate quando $r = 10$ oppure $r = 100$. Notiamo che la concordanza tra $Net - 2$ e $Net - 2_{mul^*}$ aumenta con il tempo, perfino quando $r = 1$, forse per via della scelta degli attrattori computazionali fatta che, in questo caso, tende ad un punto fisso. L'analisi più quantitativa ed estensiva fatta nel riquadro (d) mostra, per ogni costante di tempo, il valor medio delle distanze $d^{i,k}$ tra traiettorie di un vasto campione di coppie di reti $Net - 2$ e $Net - 2_{mul^*}$. Come ci si può aspettare, i grafici partono da una distanza nulla all'origine e, inizialmente, crescono. Poi decrescono e si stabilizzano ad un valore sufficientemente piccolo, in particolare per $r = 100$, di nuovo per gli attrattori computazionali, che tende a punti fissi (si veda il riquadro (d)). Analogamente, in figura 2.6.2, sono mostrate le traiettorie campionate dei cinque neuroni di $Net-5$ e dei cinque originali neuroni di $Net-5_{mul^*}$ così come i valori medi delle distanze $d^{i,k}$ tra le traiettorie di coppie di reti $Net - 5$ e $Net - 5_{mul^*}$. In questo caso, possiamo notare che, sebbene per rapporti r maggiori la similitudine delle traiettorie sia ancora buona, è presente una piccola riduzione di efficienza. Le cause sono probabilmente legate al ridimensionamento del numero di neuroni che può richiedere, non solo maggiori valori del rapporto r, ma anche una migliore approssimazione del prodotto reale, possibilmente da ottenersi tramite un tipo di sottorete mul^* più complessa.

Tabella 2.3: *Parametri dell'esperimento sulla robustezza e sul problema di scala temporale.*

Parametri	**Gruppo sperimentale n. 1**	**Gruppo sperimentale n. 2**
N (numero di neuroni di G originale)	2	5
Numero di neuroni di G_{mul^*}	20	95
Intervallo di pesi e soglie	$(-10, 10)$	$(-10, 10)$
Intervallo della condizione iniziale (y_{min}, y_{max})	$(0, 1)$	$(0, 1)$
$r_1 = \tau/\tau_1^*$	1	1
$r_2 = \tau/\tau_2^*$	10	10
$r_3 = \tau/\tau_3^*$	100	100
S (numero di reti)	10^2	10^2

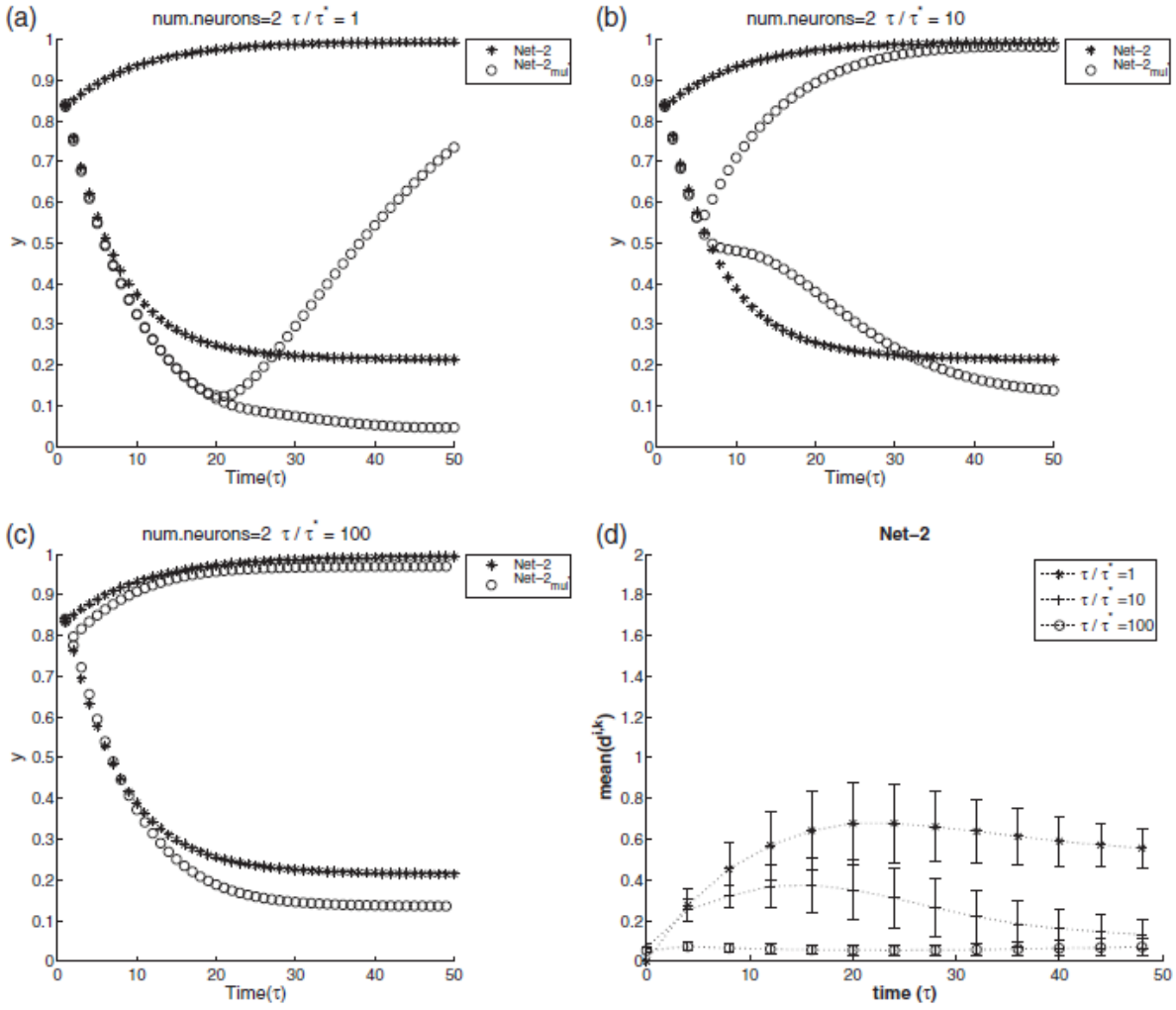


Figura 2.6.1: *Esperimento per la scala temporale di* $Net-2_{mul^*}$. *I riquadri (a), (b) e (c) mostrano le traiettorie campionate di*$Net-2$ *e* $Net-2_{mul^*}$ *per i tre differenti valori di del rapporto* $r = \tau/\tau^*$. *Il riquadro (d) riporta la media e la deviazione standard delle distanze* $d^{i,k}$ *tra traiettorie di coppie di reti* $Net-2$ *e* $Net-2_{mul^*}$. *Per una miglior visualizzazione la deviazione standard è stata dimezzata. (Immagine tratta da Donnarumma* e altri, *2012)*

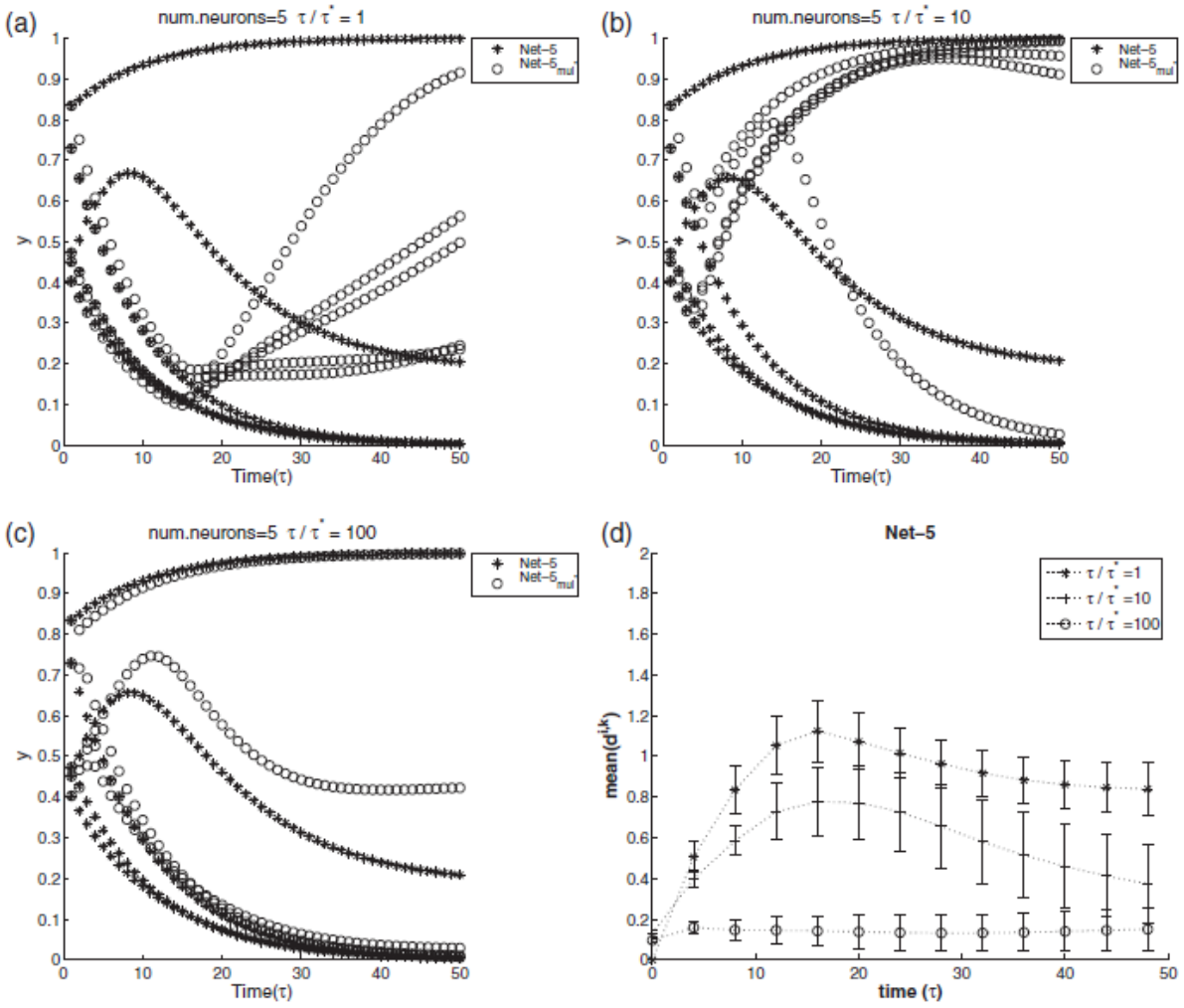


Figura 2.6.2: *Esperimento per la scala temporale di* $Net-5_{mul^*}$. *I riquadri (a), (b) e (c) mostrano le traiettorie campionate di*$Net-5$ *e* $Net-5_{mul^*}$ *per i tre differenti valori di del rapporto* $r = \tau/\tau^*$. *Il riquadro (d) riporta la media e la deviazione standard delle distanze* $d^{i,k}$ *tra traiettorie di coppie di reti* $Net-5$ *e* $Net-5_{mul^*}$. *Per una miglior visualizzazione la deviazione standard è stata dimezzata. (Immagine tratta da Donnarumma* e altri, *2012)*

Capitolo 3

Architetture a confronto: Un esperimento robotico

Nell'esperimento robotico che stiamo per definire abbiamo messo a confronto due architetture di controllo diverse, comparabili tra loro in quanto aventi stessi parametri d'ingresso e d'uscita, entrambe biologicamente plausibili e teoricamente idonee ad apprendere comportamenti multipli. La prima - a cui ci riferiremo con Architettura Programmabile (AP) - si compone di due reti, ovvero una Rete Neurale *Feedforward* (FNN) che codifica i programmi ed una Rete Neurale Programmabile (PNN) che funge da interprete. La seconda - a cui ci riferiremo con Architettura Non Programmabile (ANP) - è costituita da un'unica Rete Neurale Ricorrente a Tempo Continuo (CTRNN) in modalità *full connected*.

3.1 Descrizione dell'esperimento robotico

3.1.1 Lo scenario robotico

Lo scenario idealizzato nell'esperimento è un labirinto percorso da un *robot* (fig. 3.1.1). I percorsi rettilinei del labirinto dall'ingresso fino alla prima biforcazione, e da ogni biforcazione alla successiva, hanno tutti uguale lunghezza δ. Qualsiasi sia il percorso scelto dal *robot*, si presenteranno sempre e solo tre biforcazioni separate l'un l'altra da uno dei suddetti percorsi rettilinei.

3.1.2 Le primitive eseguite dal *robot*

Le primitive che possono essere eseguite dal *robot* sono due, ovvero *Left Follower* (LF, o semplicemente L) cioè "segui il muro a sinistra" oppure *Right Follower* (RF, o semplicemente R) cioè "segui il muro a destra". Il *robot* si muoverà all'interno del labirinto con l'aiuto di opportuni sensori che daranno un feedback costante sulla presenza di un muro

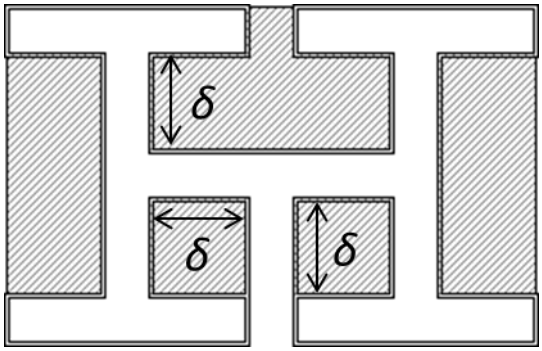


Figura 3.1.1: *Lo scenario robotico. Un immagine del labirinto utilizzato negli esperimenti. In basso nella figura si nota il punto di partenza del* robot*, ovvero l'entrata del labirinto, in corrispondenza della parete aperta. Ogni tratto rettilineo, tra una svolta (o l'ingresso) e la successiva, ha lunghezza* δ*.*

a destra o a sinistra, in mancanza del muro (biforcazione) il *robot* eseguirà una svolta di 90° per recuperarlo. L'assenza del muro, ovvero lo stato di svolta, segnalato contemporaneamente dai sensori di sinistra e di destra, vedremo che corrisponde allo stato alto del **segnale di *trigger***.

3.1.3 I parametri dell'architettura di controllo

L'architettura di controllo generica (fig. 3.1.2) si configura come una *black box* che preleva segnali in ingresso e restituisce segnali di uscita. In seguito saranno definite due possibili architetture che si differenziano per la loro programmabilità o non programmabilità. Il ruolo di questa architettura di controllo è quello di apprendere e far esibire al *robot* comportamenti multipli, per raggiungere la terminazione indicata dal programma, all'interno di un labirinto di qualsiasi dimensione.

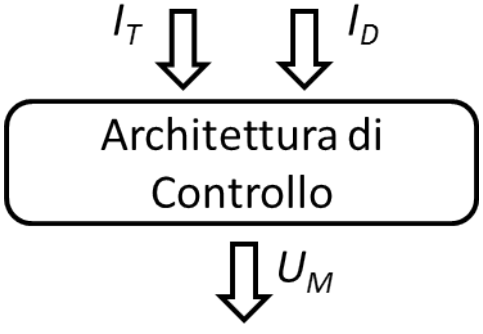


Figura 3.1.2: *Architettura di controllo generica. In ingresso l'input di task* I_T *ed il segnale di* trigger I_D*, in uscita l'output motorio* U_M*.*

3.1.3.1 I segnali di ingresso: *task* e *trigger*

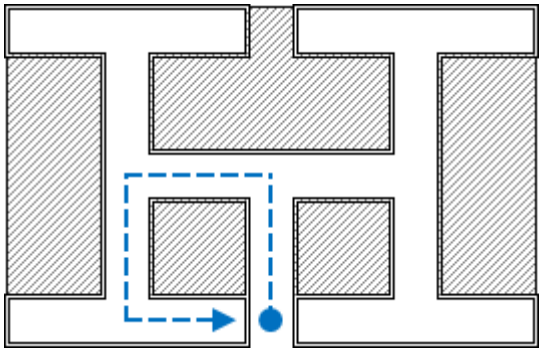
I segnali in ingresso, comuni ad entrambe le architetture sono due, di seguito descritti.

Il programma e l'input di *task*

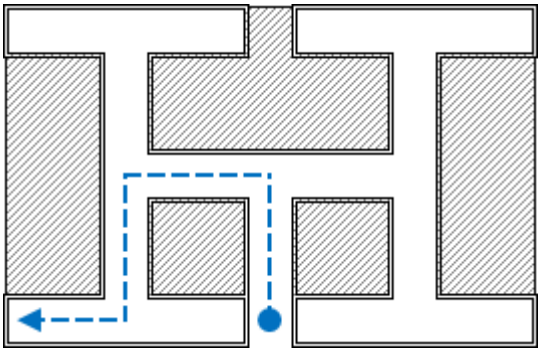
Dato il tipo di scenario presentato (fig. 3.1.1), è facile definire ogni possibile combinazione di tre svolte in successione che consenta al *robot* di raggiungere una differente terminazione del labirinto (fig. 3.1.3), ovvero un **programma** P, a cui è associato il relativo input di *task* I_T. Si ottengono otto vettori di tre elementi costanti, che rappresentano ciascuna combinazione di svolte a sinistra **L** o a destra **R**:

$$
\begin{aligned}
P_1 &= \begin{bmatrix} L & L & L \end{bmatrix} \rightarrow I_{T_1} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} \\
P_2 &= \begin{bmatrix} L & L & R \end{bmatrix} \rightarrow I_{T_1} = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \\
P_3 &= \begin{bmatrix} L & R & L \end{bmatrix} \rightarrow I_{T_1} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \\
P_4 &= \begin{bmatrix} L & R & R \end{bmatrix} \rightarrow I_{T_1} = \begin{bmatrix} 0 & 1 & 1 \end{bmatrix} \\
P_5 &= \begin{bmatrix} R & L & L \end{bmatrix} \rightarrow I_{T_1} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \\
P_6 &= \begin{bmatrix} R & L & R \end{bmatrix} \rightarrow I_{T_1} = \begin{bmatrix} 1 & 0 & 1 \end{bmatrix} \\
P_7 &= \begin{bmatrix} R & R & L \end{bmatrix} \rightarrow I_{T_1} = \begin{bmatrix} 1 & 1 & 0 \end{bmatrix} \\
P_8 &= \begin{bmatrix} R & R & R \end{bmatrix} \rightarrow I_{T_1} = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix}
\end{aligned}
$$

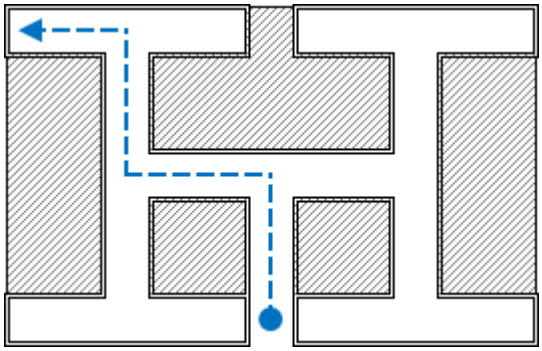
Tali programmi possono essere interpretati come veri e propri comportamenti che si chiederà al *robot* di manifestare all'interno del labirinto per raggiungere le differenti terminazioni. Infatti i programmi non definiscono univocamente le traiettorie che, a loro volta, potranno quindi variare a seconda degli output motori che verranno appresi e successivamente prodotti dalla rete neurale che governa il *robot*.



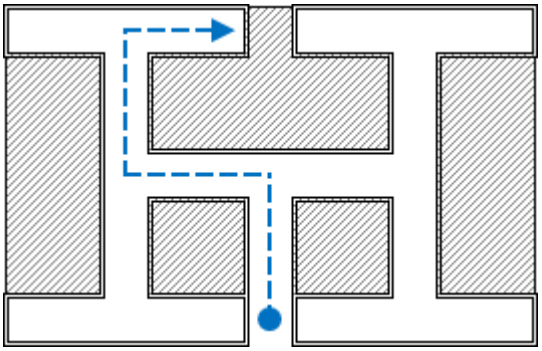
Programma 1: $P_1 = \left[\begin{array}{ccc} L & L & L \end{array} \right]$



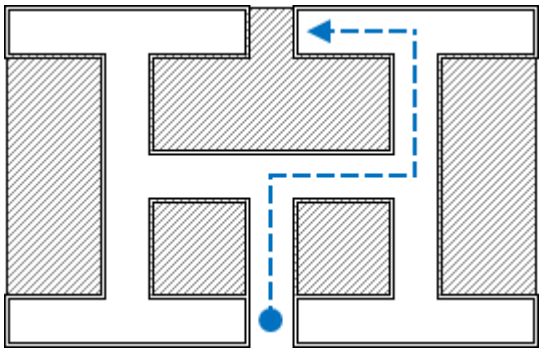
Programma 2: $P_2 = \left[\begin{array}{ccc} L & L & R \end{array} \right]$



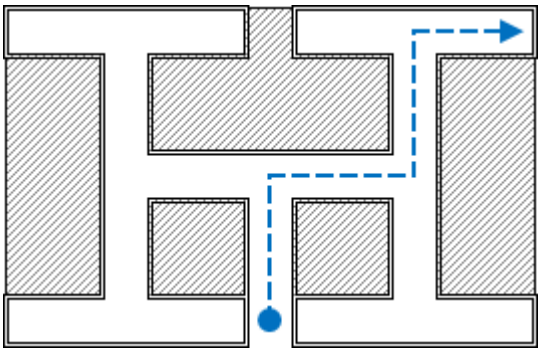
Programma 3: $P_3 = \left[\begin{array}{ccc} L & R & L \end{array} \right]$



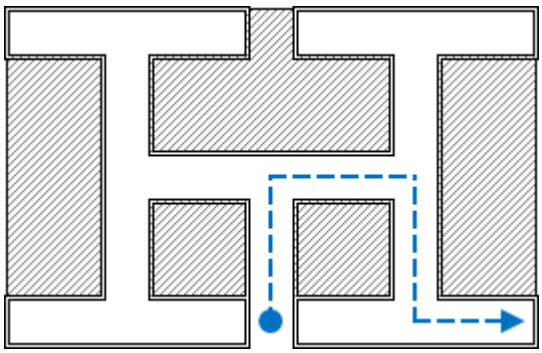
Programma 4: $P_4 = \left[\begin{array}{ccc} L & R & R \end{array} \right]$



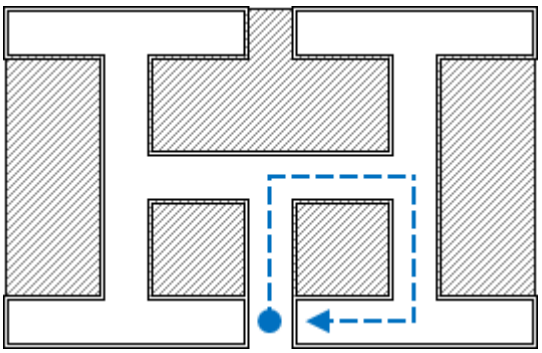
Programma 5: $P_5 = \left[\begin{array}{ccc} R & L & L \end{array} \right]$



Programma 6: $P_6 = \left[\begin{array}{ccc} R & L & R \end{array} \right]$



Programma 7: $P_7 = \left[\begin{array}{ccc} R & R & L \end{array} \right]$



Programma 8: $P_8 = \left[\begin{array}{ccc} R & R & R \end{array} \right]$

Figura 3.1.3: *Gli otto programmi P definiti per lo scenario robotico, costituiti da tutte le possibili combinazioni di svolte all'interno del labirinto scelto.*

Il *trigger*

I sensori del *robot* forniscono come segnale in ingresso un ***trigger*** $I_D(\tau) \in [0,1]$ che quando è alto indica la presenza di una biforcazione, per ogni intervallo di tempo τ. Il *trigger* fondamentalmente consente al *robot* di svoltare e avviare il programma successivo. Il motivo di questa scelta risiede nel fatto che non si voglia vincolare il *robot* ad un percorso preciso, bensì valutare di volta in volta se la posizione raggiunta è quella corretta, tenendo in conto l'errore spaziale che si accumula lungo il percorso, rispetto ad una traiettoria ideale, ovvero quella di cui si potrebbero calcolare i tempi di percorrenza in base alla dimensione dei tratti del labirinto.

Supponendo che il *robot* si sposti a velocità costante, se si presentano N svolte nel labirinto, possiamo notare come la durata del segnale di *trigger* nello stato basso $\triangle\tau_{I_D LO}$: $I_D(\tau) = 0 \; \forall \tau \in \triangle\tau_{I_D LO}$, limitatamente ai tratti che precedono le prime $N-1$ svolte a partire dall'ingresso, sia proporzionale alla lunghezza δ dei percorsi rettilinei del labirinto:

$$\triangle\tau_{I_D LO} \propto \delta. \tag{3.1.1}$$

Infatti, durante questa fase, il *robot* si suppone stia seguendo il muro in un tratto rettilineo; il *trigger* passa allo stato alto non appena termina il muro, momento nel quale il *robot* inizia ad effettuare la svolta che termina in corrispondenza della ricomparsa del muro su entrambi i lati, la durata è allora $\triangle\tau_{I_D HI}$: $I_D(\tau) = 1 \; \forall \tau \in \triangle\tau_{I_D HI}$. In questo senso, aumentare il rapporto tra stato basso e stato alto del segnale corrisponde ad un'espansione in scala delle dimensioni del labirinto, mentre diminuire tale rapporto rappresenta una contrazione in scala delle dimensioni del labirinto.

Possiamo, inoltre, fissare il tempo di svolta - durata del *trigger* nello stato alto - ritenendo plausibile che questo non cambi con le dimensioni del labirinto ma dipenda dalle sole dimensioni fisiche del *robot*. Approssimazione che ci consente di utilizzare il parametro adimensionale:

$$\lambda \hat{=} \triangle\tau_{I_D LO} / \triangle\tau_{I_D HI} \tag{3.1.2}$$

e in tal senso definire:

$$\delta \hat{=} \lambda \cdot k \tag{3.1.3}$$

con k $[m]$ costante di proporzionalità, in luogo della variazione della lunghezza δ dei tratti rettilinei nel labirinto.

Infine, indichiamo la durata di un ciclo del segnale di *trigger* con:

$$T = \triangle\tau_{I_D LO} + \triangle\tau_{I_D HI}. \tag{3.1.4}$$

In caso sia presente del rumore, ovvero delle variazioni sui tempi di percorrenza del

labirinto, sia di motivo topologico, quali ad esempio eventuali asimmetrie del labirinto, sia relative ad errore negli spostamenti del *robot*, le proporzioni introdotte continuano a valere solo nel caso in cui il rumore sia almeno di un ordine di grandezza inferiore rispetto ai tempi di percorrenza considerati.

3.1.3.2 I segnali di uscita

In uscita si ha un solo segnale $U_M(\tau) \in [0,1]$ che viene inviato ai circuiti attuatori del *robot* per controllarne il movimento. Lo stato basso corrisponde ad un comportamento *Left Follower* mentre lo stato alto corrisponde ad un comportamento *Right Follower*. In sostanza, il segnale d'uscita è la codifica delle primitive.

Si può considerare alto il segnale in uscita se superiore ad una soglia logica posta esattamente a metà dell'ampiezza del segnale, basso in caso contrario. Tale interpretazione può essere affidata a semplici circuiti per il ripristino del livello logico del segnale come, ad esempio, un numero N pari di invertitori in cascata.

Notiamo che lo stato del segnale di uscita, pur essendo un segnale continuo, verrà letto dal *robot* solamente in corrispondenza della necessità di avviare il programma successivo, ovvero all'ingresso del labirinto e dopo l'arrivo del segnale di trigger che conferma l'avvenuta svolta del *robot* all'interno del labirinto, nel nostro caso, per un totale di tre volte. Nonostante l'esperimento mantenga alcune caratteristiche di idealità nella sua definizione teorica, vogliamo supporre che la lettura dello stato d'uscita non sia immediata, ma avvenga, al contrario, con un certo *offset* che abbiamo stimato, per eccesso, in 5τ, dove τ è l'unità di tempo caratteristica della rete. Tale *offset* verrà utilizzato per leggere le uscite delle reti e valutare il successo dei singoli programmi dai risultati che presenteremo più avanti.

3.1.4 Generazione dei *dataset* per l'apprendimento

Il *dataset* si compone di una serie di dati di prova generati *ad hoc* per la fase di apprendimento dell'architettura di controllo. In particolare, contiene un numero finito di *set* composti da:

- un input di *task* I_{T_i} con $i \in [1,8]$;
- un input di *trigger* $I_D(\tau)$;
- il relativo output ideale $U_M^I(\tau)$ atteso dall'architettura di controllo.

3.1.4.1 *Dataset* per i test e labirinto base

Il *dataset* per i test (eq. 3.1.4), corrispondente a quello che chiameremo il **labirinto base** ($\lambda = 2$), contiene otto *set* di dati, corrispondenti al numero di programmi (combinazioni di svolte) possibili, contenenti ciascuno:

- un input di *task* I_{T_i} con $i \in [1, 8]$;
- una copia dell'unico segnale di trigger valido per tutti i *set* con $\triangle\tau_{I_D HI} = 5$ (durata di svolta) e $\triangle\tau_{I_D LO} = 2 \cdot \triangle\tau_{I_D HI} = 10$ (tempo di percorrenza del tratto rettilineo di lunghezza δ), ovvero con $\lambda = 2$ (eq. 3.1.2);
- la relativa uscita ideale $U^I_{M_i}(\tau)$, calcolata tramite i due ingressi menzionati.

Tale *dataset* viene maggiormente utilizzato per valutare le prestazioni delle architetture di controllo prima di passare alla fase di apprendimento vera e propria. Ricordiamo che la corrispondenza con il labirinto base è giustificata dalla proporzionalità tra la durata del segnale di trigger e la lunghezza dei percorsi del labirinto (eq. 3.1.3).

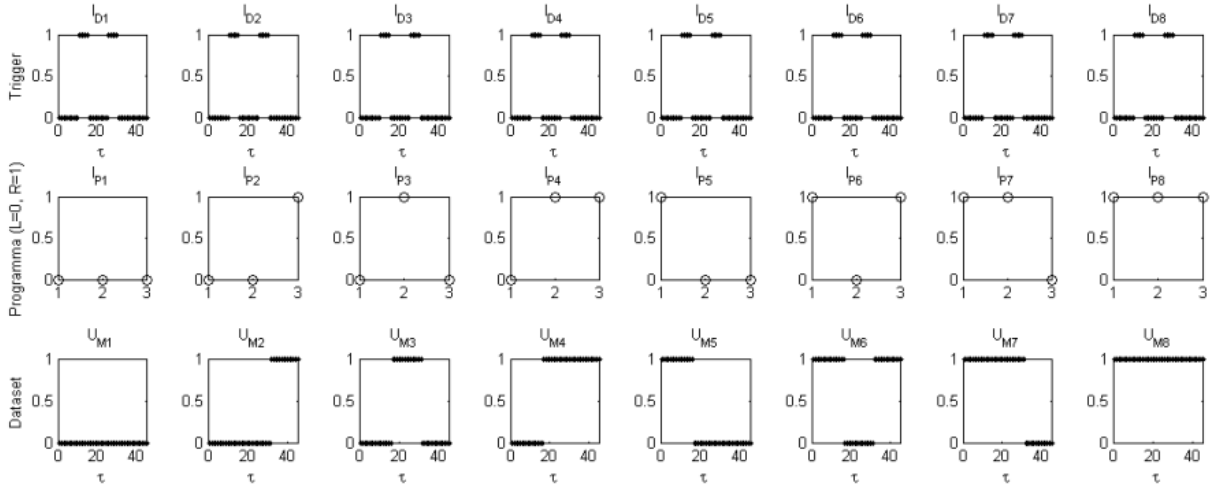


Figura 3.1.4: Dataset *del labirinto base. Usato per i test, corrispondente al labirinto base ($\lambda = 2$), contiene otto* set *di dati, uno per ogni programma possibile. Ogni* set *contiene un input di* task I_{T_i} *con* $i \in [1, 8]$*; una copia del segnale di* trigger *con* $\triangle\tau_{I_D HI} = 5$ *(durata di svolta) e* $\triangle\tau_{I_D LO} = 2 \cdot \triangle\tau_{I_D HI} = 10$ *(tempo di percorrenza del tratto rettilineo di lunghezza* δ*); la relativa uscita ideale* $U^I_{M_i}(\tau)$*, calcolata tramite i due ingressi menzionati.*

3.1.4.2 *Dataset* per l'apprendimento di comportamenti

Il *dataset* completo (eq. 3.1.5), corrisponde a ventiquattro *set* di dati, relativi a tre labirinti con parametro λ rispettivamente pari a due, tre e quattro. Per quanto possa essere banale ottenere una rappresentazione completa a partire dai parametri sintetici appena citati,

cerchiamo di esplicitare i dati rilevanti. I primi otto *set* contengono di fatto il **labirinto base**. Il secondo ottetto di *set* corrisponde a:

- otto input di *task* I_{T_i} con $i = 1, \ldots, 8$;
- una copia dell'unico segnale di trigger valido per tutti i *set* con $\triangle\tau_{I_D HI} = 5$ (durata di svolta) e $\triangle\tau_{I_D LO} = \lambda \cdot \triangle\tau_{I_D HI} = 15$ (tempo di percorrenza del tratto rettilineo di lunghezza δ) (eq. 3.1.2);
- le relative uscita ideali $U^I_{M_i}(\tau)$, calcolate tramite i due rispettivi ingressi menzionati.

Il terzo ed ultimo ottetto di *set* corrisponde invece a:

- otto input di *task* I_{T_i} con $i = 1, \ldots, 8$;
- una copia dell'unico segnale di trigger valido per tutti i *set* con $\triangle\tau_{I_D HI} = 5$ (durata di svolta) e $\triangle\tau_{I_D LO} = \lambda \cdot \triangle\tau_{I_D HI} = 20$ (tempo di percorrenza del tratto rettilineo di lunghezza δ) (eq. 3.1.2);
- le relative uscita ideali $U^I_{M_i}(\tau)$, calcolate tramite i due rispettivi ingressi menzionati.

Tale *dataset* viene utilizzato nella fase di apprendimento dei comportamenti. La presenza di più di un labirinto permette alle architetture di controllo a reti neurali di fare una generalizzazione sui dati appresi e garantire prestazioni migliori per segnali di ingresso che differiscano significativamente dai valori forniti in fase di apprendimento, ovvero labirinti di grandezza qualsiasi. Ricordiamo che quello che vogliamo apprendere sono comportamenti, definiti dal programma in ingresso, e non traiettorie.

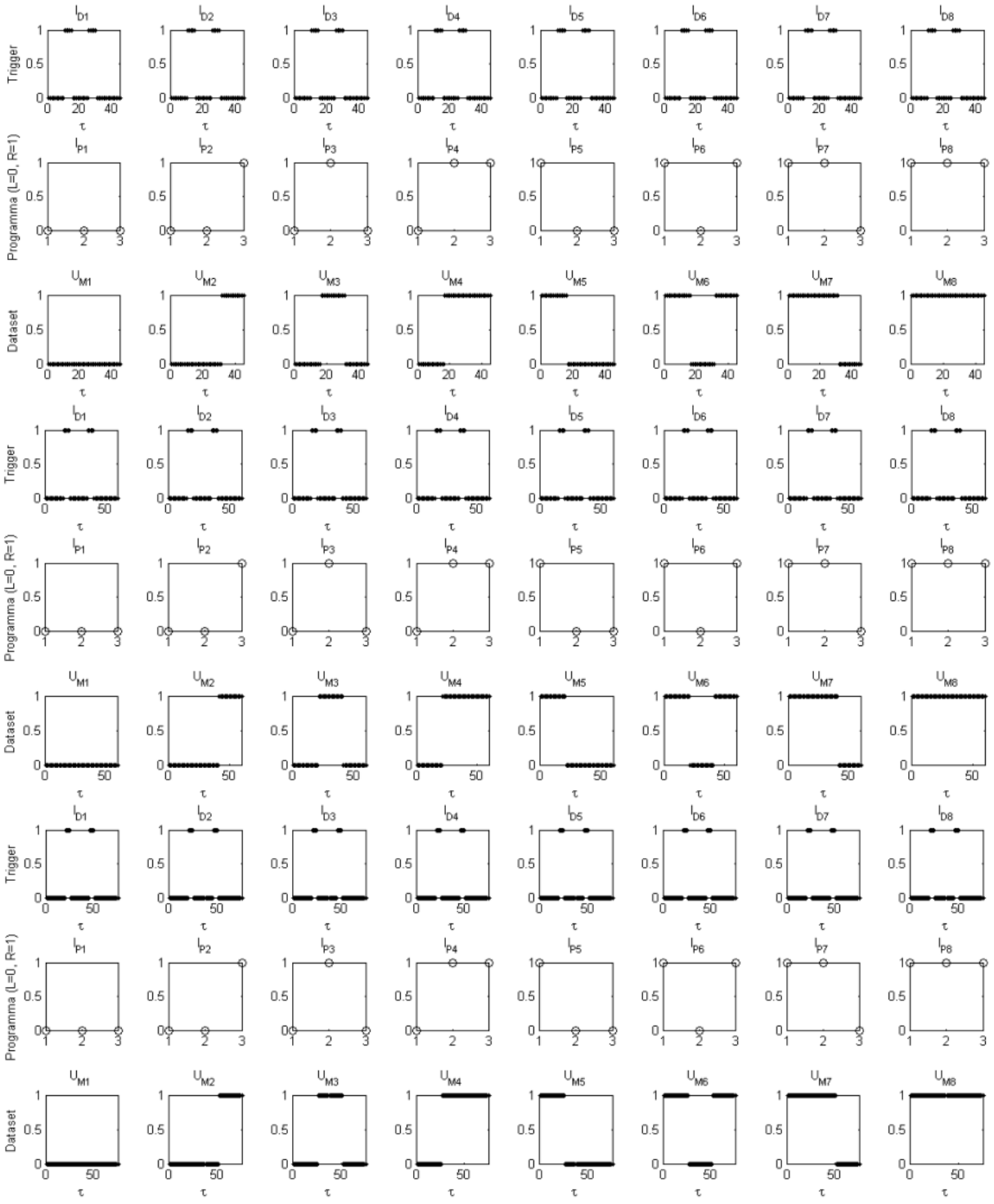


Figura 3.1.5: *Dataset per tre labirinti. Gli stessi otto programmi si ripetono per 3 labirinti di diverse dimensioni:* $\lambda = 2$ *(righe 1-3),* $\lambda = 3$ *(righe 4-6),* $\lambda = 4$ *(righe 7-9). Ad ogni coppia di input* task-trigger*, con la durata del* trigger *basso che è proporzionale alla grandezza del labirinto, è associato l'output atteso* $U^I_{M_i}(\tau)$.

3.2 Architettura Non Programmabile

Esaminiamo le caratteristiche principali dell'Architettura Non Programmabile (ANP) e fissiamo i parametri e le modalità entro i quali effettuare l'apprendimento dei comportamenti.

3.2.1 Struttura

Questa architettura è costituita da una sola Rete Neurale Ricorrente a Tempo Continuo (CTRNN) in modalità *full connected* (fig. 3.2.1), ovvero ogni singolo neurone è collegato ad ognuno degli altri. Ricordiamo che tale ANP rappresenta il più semplice tipo di architettura neurale biologicamente plausibile ed è quella più usata in letteratura per affrontare compiti come quello che ci proponiamo in questa sede: apprendere otto comportamenti differenti, uno per ognuno degli otto diversi programmi forniti alla rete.

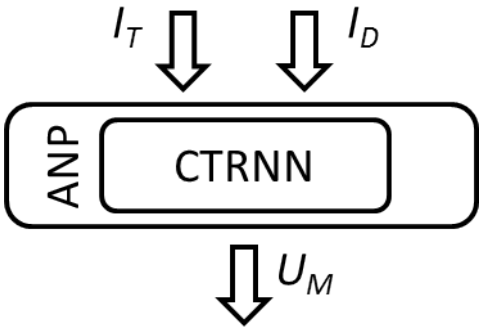


Figura 3.2.1: *Architettura Non Programmabile. In ingresso ad una rete CTRNN di tipo* full connected *troviamo l'input di* task I_T *ed il segnale di* trigger I_D*, in uscita l'output motorio* U_M*.*

3.2.2 Apprendimento della CTRNN

L'algoritmo utilizzato per l'apprendimento della CTRNN è la *Differential Evolution.* I parametri su cui possiamo agire sono:

- il **numero di neuroni** N_N che compongono la rete neurale;
- la **popolazione** N_{pop}, cioè il numero di varianti dei pesi sinaptici w_{ij} (della connessione dal neurone i al neurone j) che vengono generate e testate ad ogni passo d'apprendimento;
- l'**intervallo minimo/massimo dei pesi sinaptici** $[w_{min}, w_{max}]$, entro il quale variano i valori della popolazione;

- la **costante di tempo** τ della rete;
- l'**intervallo di tempo** $\triangle\tau$ utilizzato nell'integrazione tramite l'algoritmo di Eulero;
- il **numero di passi** S d'apprendimento.

Questa architettura ci consente di eseguire due diverse modalità d'apprendimento che differiscono sostanzialmente per il modo in cui i *dataset* vengono utilizzati.

3.2.2.1 Modalità d'apprendimento di tipo *online*

Il primo metodo d'apprendimento, denominato "*online*", consiste nel suddividere il *dataset* nei *set* riguardanti ciascuno degli N_P programmi e utilizzare ciascun *set* per un numero fissato S_P di passi d'apprendimento per singolo programma. La funzione d'errore, anche detta funzione di costo o *fitness*, viene resettata al passaggio da un programma al successivo. Completato l'apprendimento per tutti gli N_P programmi, si ricomincia dal primo programma, passando così al **ciclo** d'apprendimento successivo. Il tutto viene eseguito per un numero fissato di cicli C.

Il **numero dei passi** d'apprendimento in questo caso sarà:

$$S = S_P \cdot N_P \cdot C$$

cioè dipenderà sostanzialmente da due variabili: S_P e C.

3.2.2.2 Modalità d'apprendimento di tipo *batch*

Il secondo metodo d'apprendimento, denominato "batch", consiste nell'utilizzare l'intero *dataset* per un numero fissato S di passi d'apprendimento. In tal modo la funzione d'errore non distingue più tra i singoli programmi, ma valuta la risposta data dalla rete ad un unico ingresso equivalente, composto dalla successione di tutti i possibili ingressi. Questa modalità, in sintesi, è una mera applicazione del principio di sovrapposizione degli effetti.

3.2.2.3 Confronto tra modalità di tipo online e batch

Sebbene la modalità *online* sembri di primo acchito più accurata, la modalità *batch* porta, in taluni casi, un enorme vantaggio dal punto di vista della qualità d'apprendimento rispetto alla prima. Se, infatti, la rete apprende tutti i programmi contemporaneamente, ciascun passo d'apprendimento allora non può portare un beneficio maggiore ad un singolo programma a discapito degli altri. La funzione d'errore, che nella modalità *online* viene resettata ad ogni passaggio da un programma all'altro ed è valutata indipendentemente per

ciascun programma, è infatti incrementata indirettamente durante l'apprendimento degli altri programmi dalla modifica dei pesi sinaptici della rete in favore della riduzione della funzione di costo dell'unico programma valutato al dato passo d'apprendimento. Tale effetto diventa notevole quando aumentano i passi d'apprendimento per singolo programma S_P e il numero di programmi N_P.

Si ritiene tuttavia utile, ai fini del confronto tra le diverse architetture, fornire dati per l'ANP derivanti da entrambi i tipi di apprendimento qui descritti.

3.3 Architettura Programmabile

Esaminiamo le caratteristiche principali dell'Architettura Programmabile (AP) e fissiamo i parametri e le modalità entro i quali effettuare l'apprendimento dei comportamenti.

3.3.1 Struttura

Questa architettura è costituita da due reti, ovvero una Rete Neurale *Feedforward* (FNN) che codifica i programmi ed una Rete Neurale Programmabile (PNN, § 2.4) che funge da interprete (fig. 3.3.1). L'AP rappresenta un tipo di architettura neurale biologicamente plausibile, sicuramente più complessa rispetto all'ANP dal punto di vista strutturale, ma consente di apprendere comportamenti differenti in modo totalmente innovativo rispetto all'approccio classico.

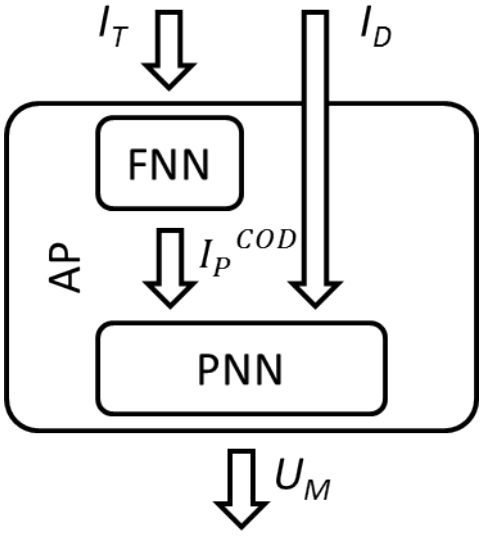


Figura 3.3.1: *Architettura Programmabile. In ingresso ad una FNN troviamo l'input di* task I_T*, la cui uscita di programma codificato* I_P^{COD} *va, insieme al segnale di* trigger I_D*, in ingresso ad una PNN. In uscita, dalla PNN, l'output motorio* U_M*. Il ruolo della FNN è di memorizzare la codifica di programma utilizzata per descrivere la struttura della PNN in termini di pesi sinaptici.*

3.3.2 Apprendimento della FNN

L'algoritmo utilizzato per l'apprendimento della FNN è la *Back Propagation*. Questa rete è un'evidente semplificazione rispetto alle CTRNN utilizzate, giustificata però dal fatto che giochi un ruolo decisamente marginale rispetto alla PNN. La FNN, infatti, si limita semplicemente ad associare ad un task I_{T_i} in ingresso, la sua codifica P_i^{COD} che andrà in ingresso alla rete PNN. La sua presenza è necessaria perché la memorizzazione della codifica dei programmi, per restare biologicamente plausibile, deve essere pur sempre affidata ad una rete neurale.

La fase d'apprendimento è molto più rapida, tant'è che viene eseguita ogni volta che viene modificata una codifica di programma P_i^{COD} della rete PNN, per un totale di mille passi d'apprendimento, più che sufficienti a portare l'errore complessivo della rete a valori, per alcuni centesimi, prossimi allo zero.

3.3.3 Apprendimento della PNN

Sebbene parlare di PNN, e non di "CTRNN programmabile", possa portarci a ritenere che si tratti di una rete neurale del tutto nuova nel suo genere, vale la pena di ricordare che dal punto di vista dinamico rimane una CTRNN e gode delle stesse identiche caratteristiche fisiche biologicamente plausibili. L'algoritmo utilizzato per l'apprendimento della PNN è sempre e comunque la *Differential Evolution*. Vedremo però che, in effetti, una CTRNN dotata di programmabilità, che ha notevoli differenze strutturali rispetto ad una semplice CTRNN, risulta notevolmente migliore già a partire dalle nuove possibili modalità d'apprendimento e, come ci proponiamo di dimostrare in questo lavoro, dovrebbe risultare in generale molto più efficace nell'apprendere i diversi comportamenti. I parametri su cui possiamo agire sono:

- il **numero di neuroni** N_N che compongono la rete neurale;
- la **popolazione** N_{pop}, cioè il numero di varianti dei pesi sinaptici w_{ij} (della connessione dal neurone i al neurone j) che vengono generate e testate ad ogni passo d'apprendimento;
- l'**intervallo minimo/massimo dei pesi sinaptici** $[w_{min}, w_{max}]$, entro il quale variano i valori della popolazione;
- la **costante di tempo** τ della rete;
- l'**intervallo di tempo** $\triangle\tau$ utilizzato nell'integrazione tramite l'algoritmo di Eulero;
- il **numero di passi** S d'apprendimento.

Questa architettura ci consente di eseguire una modalità d'apprendimento *standard* che, come vedremo, non ha più senso separare in *online* e *batch*, oppure una modalità che ho sviluppato durante questo lavoro, indicata come modalità d'apprendimento a soglia.

3.3.3.1 Modalità d'apprendimento *standard*

Il metodo d'apprendimento *standard* consiste nell'eseguire, una per volta, l'apprendimento di N_P reti differenti, quanti sono i programmi, per un numero S_P di passi d'apprendimento per ciascuna rete. A tale scopo vengono utilizzati i soli *set* del *dataset* riguardanti il programma P_i. La funzione d'errore viene chiaramente resettata all'inizio dell'apprendimento di una rete diversa. Invece, alla fine dell'apprendimento di una rete, corrispondente al programma P_i, in cui viene sostanzialmente modificato P_i^{COD}, viene avviato anche l'apprendimento della codifica dei programmi P_i^{COD} $\forall i$ della rete FNN. Completato l'apprendimento di N_P reti, si passa al ciclo d'apprendimento successivo. Il tutto viene eseguito per un numero fissato di cicli C.

Il **numero dei passi** d'apprendimento in questo caso sarà:

$$S = S_P \cdot N_P \cdot C$$

e, proprio come per la modalità *online*, dipenderà dalle due variabili S e C.

La novità introdotta dalla programmabilità consiste nel poter considerare la rete neurale che deve apprendere diversi comportamenti come l'insieme di più reti distinte che apprendono un singolo comportamento, ma tutte codificate nella stessa struttura. Tale astrazione ci permette di eseguire l'apprendimento di reti equivalenti a delle CTRNN a comportamento singolo che, in termini pratici, cercheremo di dimostrare, rende queste reti incredibilmente precise nel discriminare e apprendere correttamente i singoli comportamento.

Vediamo perché non ha più senso adesso parlare di modalità *online* o *batch*. Una rete monocomportamentale, richiede l'apprendimento con un solo *dataset* intero, come per la modalità *batch*, con un unico insieme di input che produce sempre e solo un unico genere di output, ovvero il comportamento desiderato. Eppure, come per la modalità *online*, l'apprendimento avviene con una funzione d'errore dedicata al singolo programma. Si può intuire, ma lo dimostreremo, come allora tale modalità d'apprendimento porti con se i vantaggi di entrambe le modalità d'apprendimento utilizzate con la CTRNN non programmabile.

3.3.3.2 Modalità d'apprendimento con soglia

Questa modalità d'apprendimento, per quanto abbiamo ritenuto di doverla identificare con un nome nuovo, si basa esattamente sul principio d'apprendimento della modalità *standard*. La novità introdotta consiste nello stabilire a priori una **soglia** entro la quale "accontentarsi" dell'errore sul singolo programma appreso. Se può sembrare una banalità, è proprio l'eleganza di alcune soluzioni che le rende indispensabili dopo averne fatto uso. Cerchiamo di comprendere quale uso facciamo di questa soglia e tutto sarà chiaro.

L'apprendimento di una rete neurale multicomportamentale, in generale, consiste nell'apprendimento in parallelo dei singoli comportamenti, per i quali l'errore minimizzato dalla funzione d'errore è ovviamente diverso da comportamento a comportamento o, nel nostro caso, da programma a programma. Quando stabiliamo quanti passi d'apprendimento S dobbiamo eseguire, stiamo implicitamente affermando che cercheremo di abbassare contemporaneamente questi errori, per ogni programma, almeno per un numero di volte pari ad S/N_P. Tanto avviene in un apprendimento di tipo *online* per una CTRNN multicomportamentale. Se mettessimo una soglia sull'errore, l'apprendimento potrebbe fermarsi qualora tutti i programmi scendano al di sotto di questa soglia. Il vantaggio sarebbe di non controllare l'apprendimento dal numero di passi d'apprendimento totale, per accontentarci di una soluzione vaga o godere dell'affinamento ottenuto, a seconda dei casi, ma fermarlo automaticamente per soddisfare giusto il tipo di esigenze che abbiamo, senza sprechi di tempo. Purtroppo questo ragionamento è elegante quanto inutile in una rete multicomportamentale CTRNN, di complessità quale quella richiesta nel nostro esperimento, perché spesso non si raggiungerà una soglia tale da rendere l'errore trascurabile per tutti i programmi.

Dove possiamo invece sfruttare questa tecnica? Chiaramente nelle reti monocomportamentali, proprio quelle che in generale non avrebbero senso d'essere studiate, ma che nel nostro caso corrispondono alle singole reti CTRNN codificate in una PNN.

Ricapitolando, se impostiamo una soglia sulla funzione d'errore nell'apprendimento della rete per il singolo programma, premesso che la *Differential Evolution* ci assicura di esplorare tutto lo spazio delle soluzioni con un numero di passi d'apprendimento che tende ad infinito, potrà capitare di trovare una soluzione con errore inferiore alla soglia, per il dato programma, già dopo un esiguo numero di passi d'apprendimento. Non resterà allora che fermare l'apprendimento di quella data rete, la cui soluzione è già definitivamente codificata in una combinazione di pesi ottimale.

I passi d'apprendimento rimanenti potranno essere così stimati:

$$S^{Rimanenti} = S_P \cdot \left(N_P - N_P^{Completati} \right) \cdot C^{Rimanenti}.$$

Si nota immediatamente un notevole risparmio, in termini computazionali, sul numero di passi d'apprendimento necessari.

3.3.3.3 Nota finale sulle modalità d'apprendimento

A differenza dell'ANP, per la quale intendiamo fornire dati derivanti da entrambi i tipi di apprendimento *online* e *batch*, l'AP non richiede apprendimenti sia in modalità *standard* che a soglia in quanto, come già detto, quest'ultima è solo una variante della modalità *standard* in grado di portare vantaggi in termini computazionali (eliminando cicli inutili) piuttosto che qualitativi. Laddove specificato o meno, intenderemo sempre che si sia usato l'approccio con soglia.

3.4 Valutazione dell'errore: la funzione di costo

La funzione di costo è calcolata in base ad una soglia, posta a 0.5, e somma un punto intero se il valore confrontato, cioè l'uscita della rete neurale, in riferimento al valore atteso fornito dal *dataset*, si trova dalla parte sbagliata del grafico rispetto alla soglia. Inoltre, se il valore confrontato cade nella porzione del grafico attesa, l'errore è incrementato di un valore pari allo scarto quadratico tra uscita della rete e il valore atteso.

Tale funzione di costo ci assicura di ottenere un vantaggio notevole se la rete corregge il suo output, dapprima in riferimento alla soglia, e poi rispetto all'uscita attesa fornita dal *dataset*, perché, nell'eventualità di successo, nel primo caso, basterà filtrare l'output finale - dopo l'apprendimento - tramite il valore di soglia per ottenere un perfetto segnale on-off d'uscita ed eliminare la parte d'errore quadratico che si somma nel secondo caso.

Capitolo 4

Risultati e commenti

Qui di seguito i risultati dell'esperimento robotico illustrato nel capitolo precedente, nel quale abbiamo messo a confronto le due architetture proposte, ovvero una dotata di programmabilità e l'altra di tipo classico, non programmabile.

4.1 Esperimenti preliminari

Per poter raggiungere dei buoni risultati per il raffronto delle due reti abbiamo dovuto svolgere degli esperimenti preliminari nelle varie modalità d'apprendimento. Quanto segue sono i valori ottimali sui quali abbiamo basato tutte le prove successive, per entrambe le architetture, assicurando così la confrontabilità dei risultati ottenuti.

4.1.1 Sui parametri in comune a tutte le reti CTRNN

Popolazione Abbiamo scelto $N_{pop} = 100$ perché, per un numero S di passi d'apprendimento fissato, all'aumentare di tale valore non corrispondeva più un significativo aumento dei programmi appresi. Ricordiamo che N_{pop} indica il numero di varianti dei pesi sinaptici w_{ij} (della connessione dal neurone i al neurone j) che vengono generate e testate ad ogni passo d'apprendimento.

Passi d'apprendimento Abbiamo limitato gli apprendimenti di tipo *online* a $S^{online} = 20000$ passi totali, che corrispondono a 2500 passi per ogni programma; gli apprendimenti di tipo batch a $S^{batch} = S^{online}/N_P = 2500$ passi totali, che corrispondono anche a 2500 passi per ogni programma (i programmi vengono appresi tutti insieme contemporaneamente); gli apprendimenti di tipo a soglia ad un numero massimo di $S^{PNN} = 2 \cdot S^{online} = 2 \cdot N_P \cdot S^{batch} = 40000$ passi totali, che corrispondono a 5000 passi per ogni programma. In questo modo utilizziamo la *Differential Evolution* offrendo pari opportunità di apprendimento alle architetture testate, sia per quanto riguarda

l'ANP, composta da reti CTRNN *full connected,* sia per l'AP per la quale, come sarà chiaro dalla lettura dei risultati sperimentali, la scelta di raddoppiare i passi d'apprendimento non influenza in alcun modo il successo dell'apprendimento dei programmi e non fornisce vantaggi rispetto all'apprendimento dell'ANP.

Numero di neuroni Abbiamo scelto $N_N = 4$, ovvero dotato tutte le reti CTRNN, programmabili e non, di 4 neuroni, che è un numero adeguato per apprendere otto comportamenti e non troppo dispendioso in termini di risorse computazionali per simulare i diversi tipi di struttura.

Intervallo minimo/massimo dei pesi sinaptici La scelta di $[w_{min}, w_{max}] = [-5, 5]$, intervallo entro il quale variano i valori della popolazione, è stata effettuata dopo aver sperimentato che le reti programmabili non apprendevano tutti i programmi in un numero di passi d'apprendimento non eccessivo per un intervallo di $[w_{min}, w_{max}] = [-1, 1]$, ma non mostravano d'altra parte miglioramenti significativi all'allargarsi dell'intervallo rispetto al valore qui scelto. Le reti non programmabili, invece, non hanno mai fornito nella fase preliminare risultati tali da condizionare una scelta diversa su tale intervallo.

Costante di tempo La costante di tempo $\tau = 2$ delle CTRNN è stata fissata su tale valore in primo luogo per semplificare l'apprendimento dal punto di vista computazionale. In ogni caso non influenza i risultati in quanto è sia inferiore ai tempi di accensione del segnale di *trigger*, che di pari valore per tutte le reti.

Intervallo di tempo Il $\triangle\tau = 0.2$, utilizzato nell'integrazione tramite l'algoritmo di Eulero, è pari ad un decimo della costante di tempo scelta e assicura una buona resa della dinamica delle CTRNN.

4.1.2 Sulla soglia dell'apprendimento per la PNN

Nell'apprendimento a soglia abbiamo scelto i valori di soglia basandoci sulle caratteristiche della funzione d'errore utilizzata. La soglia d'errore per il *dataset* con tre labirinti è stata scelta allo 0.99 perché un errore pari ad uno costituisce almeno un errore rispetto alla soglia logica per un istante di tempo τ. I valori decimali che seguono lo zero invece rappresentano solo una possibile somma di tutti gli scarti quadratici sommati dalla funzione d'errore. Un output che corrisponda ad una funzione d'errore inferiore o uguale allo 0.99, con un semplice ripristino del valore logico diventa un output ideale.

La soglia per il *dataset* con il labirinto base è stata scelta invece allo 0.1 per sfruttare un numero maggiore di passi d'apprendimento, entro il limite massimo offerto alla PNN,

e affinare il risultato direttamente in fase d'apprendimento. Chiaramente un valore di soglia dello 0.99 sarebbe stato più che adeguato anche in questo caso.

4.2 Apprendimento del dataset con il labirinto base

Il labirinto base è quello corrispondente al *dataset* descritto al paragrafo 3.1.4.1. Abbiamo addestrato le architetture programmabili e non programmabili a riconoscere gli otto programmi di ingresso e ad esibire otto relativi comportamenti, secondo le modalità descritte nel capitolo3. Proponiamo qui di seguito i risultati ottenuti ed un primo raffronto.

4.2.1 Architettura Non Programmabile in modalità *online*

L'Architettura Non Programmabile (fig. 3.2.1) è stata addestrata in modalità *online* (nel paragrafo successivo troviamo i risultati per l'altro tipo di addestramento, in modalità *batch*), ovvero alternando l'apprendimento dei singoli programmi, in ordine crescente di nome, per un numero fissato di passi e reiterando l'intera sequenza più volte. Sebbene questo tipo di apprendimento porti a buoni risultati nell'ambito della ricerca di una soluzione ottimale per il singolo programma, l'alternarsi degli obiettivi finisce per confondere la funzione d'errore che viene ottimizzata ora in un senso, ora verso un altro tipo di soluzione; nel complesso non si raggiunge una soluzione in grado di portare l'errore a zero per tutti i programmi. In particolare, quanto appena detto, può essere subito osservato dalle tabelle 4.1 dei risultati e 4.2 con l'analisi statistica dei risultati dove si evince che il programma P_8 ha prestazioni di gran lunga migliori degli altri sette, per l'esattezza, è anche l'unico (tab. 4.3) che svolga il suo compito a dovere, portando il robot alla corretta terminazione del labirinto, in tutte e 20 le prove effettuate.

Un altro fenomeno da osservare è quello che potremmo definire uno stallo di questo algoritmo d'apprendimento: per quanti cicli d'apprendimento si possa pensare di eseguire, solo gli ultimi S_P passi d'apprendimento riguardanti ciascuno degli otto programmi sono quelli che hanno impatto sull'errore finale, questo perché ogni volta che la *fitness* ricomincia decrescere minimizzando l'errore sul primo programma, ovvero inizia un nuovo ciclo esterno, tende subito a "dimenticare" i risultati ottenuti al ciclo precedente. In definitiva, se il numero di passi d'apprendimento scelto è stato:

$$S = S_P \cdot N_P \cdot C = 20000$$

con cicli $C = 50$, passi d'apprendimento $S_P = 50$, numero di programmi $N_P = 8$, allora, avremmo anche potuto scegliere:

$$S = S_P \cdot N_P \cdot C = 400$$

con $C = 1$, $S_P = 50$, $N_P = 8$, ed ottenere risultati identici. Ecco perché non ci deve meravigliare un risultato identico per tutte le prove: un addestramento per singolo programma così rapido ($S_P = 50$), a partire dalla stessa inizializzazione della rete neurale, porta la fitness a raggiungere sempre gli stessi valori (tab. 4.1) con conseguente deviazione standard nulla (tab. 4.2).

Nella figura 4.2.1 vediamo i risultati di una qualsiasi rete, da considerarsi *best e worst case* contemporaneamente (tutte le reti si comportano allo stesso modo, pertanto i due risultati coincidono). Si nota subito come l'output U_{M8} della rete CTRNN si sposi perfettamente con quello atteso del *dataset*. La rete CTRNN ha infatti appreso perfettamente solo l'ultimo comportamento che consente al *robot* di raggiungere l'ottava terminazione del labirinto, e si comporta sempre allo stesso modo, qualsiasi sia il programma P_i presentato in ingresso.

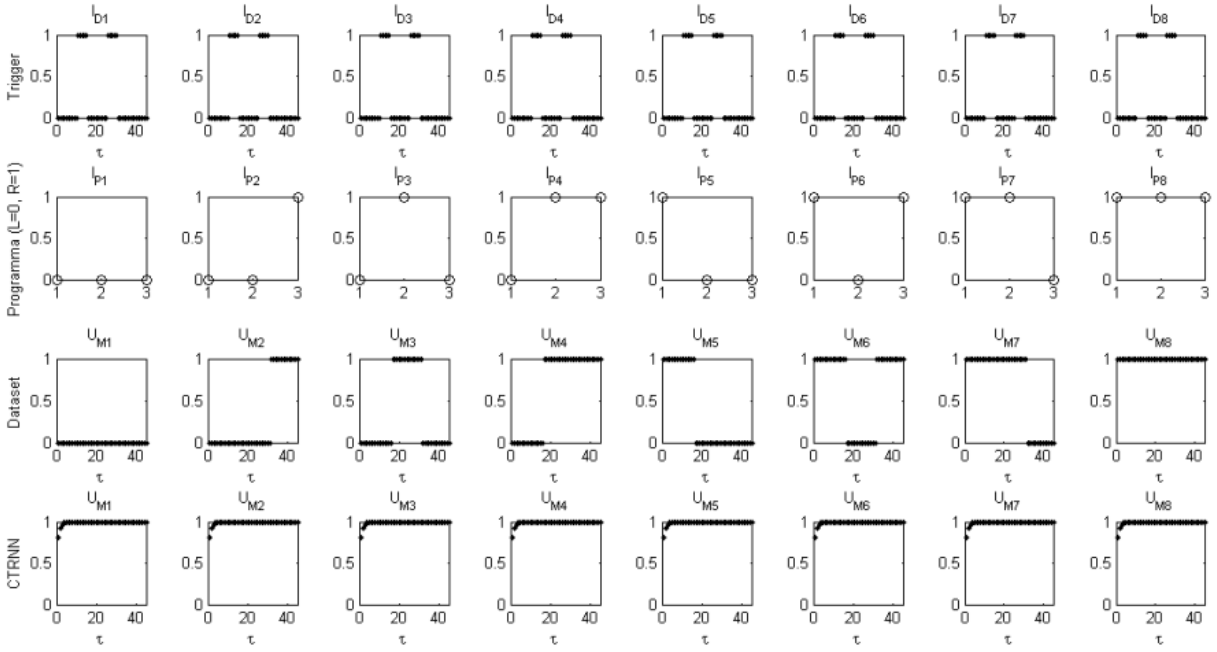


Figura 4.2.1: *Risposta della rete CTRNN numero 1 (*best/worst case*) addestrata in modalità* online *e con il* dataset *per il labirinto base, ovvero con*$\lambda = 2$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). L'unico programma corretto è il* P_8*, per il quale le uscite* U_M *del dataset e della CTRNN - al di sotto e al di sopra della soglia a 0,5 - coincidono. La lettura di* U_M *come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5* τ *dall'istante iniziale e dopo ogni discesa del segnale di* trigger.

Tabella 4.1: *Errori della CTRNN addestrata in modalità* online *con il labirinto base. L'apprendimento mostra chiaramente un'incapacità di raggiungere soluzioni ottimali per la rete con la conseguente tendenza della fitness a favorire l'ultimo programma appreso. Inoltre, anche in luogo dell'alto numero di iterazioni effettuate ($S = S_P \cdot N_P \cdot C = 20000$), non consente alcuna differenziazione dei risultati tra una prova e l'altra a causa di un'inefficienza dell'algoritmo utilizzato per l'apprendimento di comportamenti multipli.*

CTRNN (online)	**Errore del programma (Numero di svolte corrette)**							
	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
1	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
2	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
3	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
4	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
5	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
6	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
7	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
8	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
9	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
10	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
11	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
12	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
13	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
14	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
15	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
16	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
17	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
18	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
19	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
20	45.447 (0/3)	31.307 (1/3)	30.297 (1/3)	16.157 (2/3)	29.293 (1/3)	15.153 (2/3)	14.143 (2/3)	0.003 (3/3)
Successo (3/3)	0%	0%	0%	0%	0%	0%	0%	100%

Tabella 4.2: *Statistica dei risultati per la CTRNN addestrata in modalità* online *con il labirinto base. L'apprendimento mostra chiaramente un'incapacità di raggiungere soluzioni ottimali per la rete con la conseguente tendenza della fitness a favorire l'ultimo programma appreso, l'unico con errore medio quasi nullo. Inoltre, l'alto numero di iterazioni effettuate ($S = S_P \cdot N_P \cdot C = 20000$) non consente alcuna differenziazione dei risultati tra una prova e l'altra, infatti, la deviazione standard risulta nulla in tutti i casi e i valori massimi e minimi d'errore coincidono.*

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
Errore medio	45.447	31.307	30.297	16.157	29.293	15.153	14.143	0.003
Deviazione standard	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Errore massimo	45.447	31.307	30.297	16.157	29.293	15.153	14.143	0.003
Errore minimo	45.447	31.307	30.297	16.157	29.293	15.153	14.143	0.003
Frequenza di successo	0%	0%	0%	0%	0%	0%	0%	100%

Tabella 4.3: *Frequenza di successo per la CTRNN addestrata in modalità* online *con il labirinto base. Con questo tipo di addestramento l'ANP riesce ad apprendere uno ed un solo comportamento.*

Programmi appresi	1/8	2/8	3/8	4/8	5/8	6/8	7/8	8/8
Frequenza di successo	100%	0%	0%	0%	0%	0%	0%	0%

4.2.2 Architettura Non Programmabile in modalità *batch*

L'Architettura Non Programmabile (fig. 3.2.1) addestrata in modalità *batch* presenta, rispetto alla modalità *online*, risultati di gran lunga migliori (tab. 4.4 e tab. 4.5). Questa volta, infatti, i programmi P_1, P_5 e P_8 hanno una percentuale di successo del 100%. La CTRNN apprende almeno 4 su 8 programmi nel 100% dei casi e anche un quinto programma nell'85% dei casi (tab. 4.6). Come possiamo vedere dalla figura 4.2.2 la CTRNN, nella ottava esecuzione che è il *best case*, presenta, per i programmi P_1, P_3, P_4, P_5 e P_8, output analoghi a quelli del *dataset* per il labirinto base.

Il *worst case*, corrispondente alla diciassettesima rete appresa, del resto, presenta giusto un programma in meno rispetto al *best case* (fig. 4.2.3). Questo risultato è dovuto al tipo di algoritmo di apprendimento scelto: la modalità *batch*, infatti, consente di minimizzare gli errori dei singoli programmi, tenendo però sempre in memoria l'intera sequenza di uscite corrispondente a tutti i possibili ingressi. Un lavoro che porta a risultati non ottimali, ma significativamente buoni per quanto riguarda la differenziazione dei possibili comportamenti appresi.

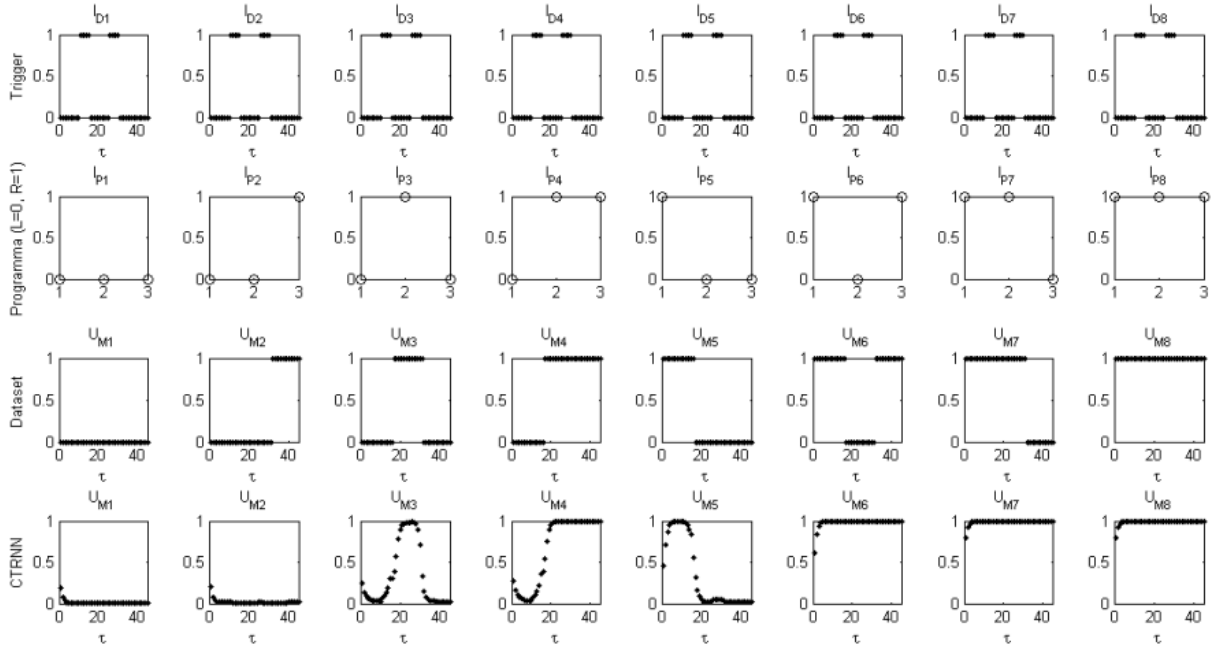


Figura 4.2.2: *Risposta della rete CTRNN numero 8 (*best case*) addestrata in modalità* batch *e con il* dataset *per il labirinto base, ovvero con*$\lambda = 2$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). I programmi corretti sono P_1, P_3, P_4, P_5 e P_8, per i quali le uscite U_M del dataset e della CTRNN hanno andamenti - al di sotto e al di sopra della soglia a 0,5 - coincidenti. La lettura di U_M come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5 τ dall'istante iniziale e dopo ogni discesa del segnale di* trigger*.*

Tabella 4.4: *Errori della CTRNN addestrata in modalità* batch *con il labirinto base. Più programmi, ma non tutti, vengono appresi correttamente. Tra le 20 esecuzioni si possono isolare un* best case *(fig. 4.2.2), con 5 su 8 programmi appresi, ed un* worst case*, con 4 su 8 programmi appresi (fig. 4.2.3).*

CTRNN (*batch*)	**Errore del programma (Numero di svolte corrette)**							
	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
1	0.004 (3/3)	1.028 (3/3)	15.155 (2/3)	4.046 (3/3)	1.031 (3/3)	15.154 (2/3)	14.144 (2/3)	0.004 (3/3)
2	0.003 (3/3)	14.149 (2/3)	15.153 (2/3)	5.068 (3/3)	0.024 (3/3)	15.154 (2/3)	2.063 (3/3)	0.004 (3/3)
3	0.003 (3/3)	5.099 (3/3)	15.168 (2/3)	4.051 (3/3)	1.021 (3/3)	15.158 (2/3)	14.143 (2/3)	0.004 (3/3)
4	0.004 (3/3)	2.064 (3/3)	15.154 (2/3)	1.034 (3/3)	3.034 (3/3)	15.154 (2/3)	14.147 (2/3)	0.006 (3/3)
5	0.003 (3/3)	14.144 (2/3)	15.154 (2/3)	3.050 (3/3)	2.035 (3/3)	15.155 (2/3)	2.052 (3/3)	0.004 (3/3)
6	0.005 (3/3)	1.057 (3/3)	15.158 (2/3)	3.047 (3/3)	2.034 (3/3)	15.154 (2/3)	14.144 (2/3)	0.006 (3/3)
7	0.004 (3/3)	10.139 (3/3)	15.154 (2/3)	0.034 (3/3)	2.029 (3/3)	15.157 (2/3)	14.144 (2/3)	0.007 (3/3)
8 (*best case*)	0.003 (3/3)	14.144 (2/3)	2.050 (3/3)	0.032 (3/3)	1.032 (3/3)	15.157 (2/3)	14.143 (2/3)	0.003 (3/3)
9	0.004 (3/3)	2.063 (3/3)	15.158 (2/3)	3.050 (3/3)	2.037 (3/3)	15.156 (2/3)	14.146 (2/3)	0.016 (3/3)
10	0.004 (3/3)	14.146 (2/3)	15.153 (2/3)	5.079 (3/3)	1.035 (3/3)	15.154 (2/3)	3.062 (3/3)	0.006 (3/3)
11	0.003 (3/3)	1.041 (3/3)	15.154 (2/3)	4.054 (3/3)	0.028 (3/3)	15.158 (2/3)	14.144 (2/3)	0.009 (3/3)
12	0.003 (3/3)	14.144 (2/3)	15.154 (2/3)	0.042 (3/3)	0.022 (3/3)	15.154 (2/3)	14.143 (2/3)	0.003 (3/3)
13	0.003 (3/3)	4.086 (3/3)	15.153 (2/3)	3.047 (3/3)	1.023 (3/3)	15.154 (2/3)	14.145 (2/3)	0.004 (3/3)
14	0.003 (3/3)	7.125 (3/3)	15.158 (2/3)	3.051 (3/3)	0.026 (3/3)	15.154 (2/3)	14.144 (2/3)	0.004 (3/3)
15	0.003 (3/3)	14.145 (2/3)	15.154 (2/3)	0.028 (3/3)	2.032 (3/3)	15.154 (2/3)	1.110 (3/3)	0.004 (3/3)
16	0.003 (3/3)	1.104 (3/3)	15.153 (2/3)	4.124 (2/3)	3.043 (3/3)	15.154 (2/3)	14.148 (2/3)	0.006 (3/3)
17 (*worst case*)	0.003 (3/3)	14.144 (2/3)	15.153 (2/3)	0.045 (3/3)	0.024 (3/3)	15.153 (2/3)	14.144 (2/3)	0.003 (3/3)
18	0.003 (3/3)	14.144 (2/3)	15.153 (2/3)	10.119 (3/3)	1.031 (3/3)	15.154 (2/3)	2.044 (3/3)	0.004 (3/3)
19	0.003 (3/3)	14.144 (2/3)	4.057 (3/3)	2.034 (3/3)	2.034 (3/3)	16.160 (2/3)	14.143 (2/3)	0.003 (3/3)
20	0.009 (3/3)	2.128 (3/3)	15.157 (2/3)	4.092 (3/3)	1.030 (3/3)	15.154 (2/3)	14.144 (2/3)	0.004 (3/3)
Successo (3/3)	100%	55%	10%	95%	100%	0%	25%	100%

Tabella 4.5: *Statistica dei risultati per la CTRNN addestrata in modalità* batch *con il labirinto base. L'apprendimento mostra una buona capacità dell'ANP ad apprendere comportamenti multipli ma non sufficiente ad apprenderne 8 su 8 proposti. I risultati mostrano per tutti i programmi una deviazione standard più bassa rispetto alla media e ciò suggerisce che il presentarsi di un risultato migliore, anche a fronte di un numero superiore di prove, non sia auspicabile.*

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
Errore medio	0.004	8.212	13.945	2.956	1.280	15.205	11.125	0.005
Deviazione standard	0.001	5.919	3.739	2.447	0.971	0.225	5.375	0.003
Errore massimo	0.009	14.149	15.168	10.119	3.043	16.160	14.148	0.016
Errore minimo	0.003	1.028	2.050	0.028	0.022	15.153	1.110	0.003
Frequenza di successo	100%	55%	10%	95%	100%	0%	25%	100%

Tabella 4.6: *Frequenza di successo per la CTRNN addestrata in modalità* batch *con il labirinto base. Con questo tipo di addestramento l'ANP riesce ad apprendere 5 su 8 comportamenti nel* best case *(85% dei casi) e almeno 4 su 8 nel* worst case *(il restante 15%).*

Programmi appresi	1/8	2/8	3/8	4/8	5/8	6/8	7/8	8/8
Frequenza di successo	100%	100%	100%	100%	85%	0%	0%	0%

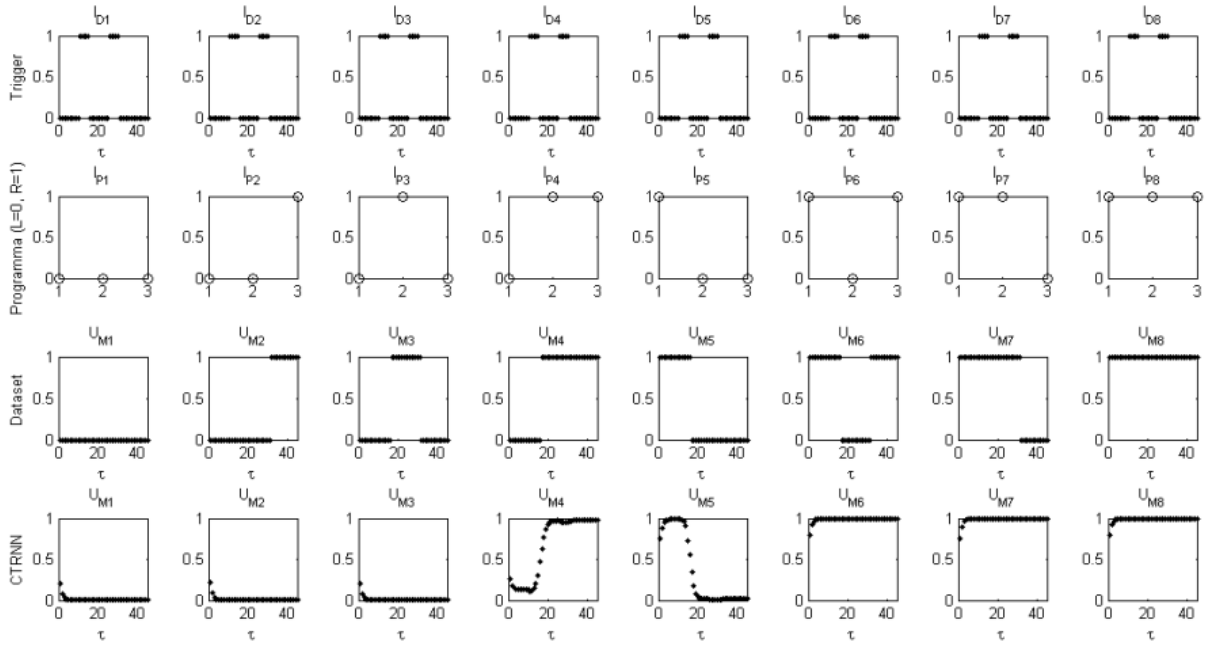


Figura 4.2.3: *Risposta della rete CTRNN numero 17 (*worst case*) addestrata in modalità* batch *e con il* dataset *per il labirinto base, ovvero con*$\lambda = 2$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). I programmi corretti sono* P_1, P_4, P_5 *e* P_8, *per i quali le uscite* U_M *del dataset e della CTRNN hanno andamenti - al di sotto e al di sopra della soglia a 0,5 - coincidenti. La lettura di* U_M *come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5* τ *dall'istante iniziale e dopo ogni discesa del segnale di* trigger.

4.2.3 Architettura Programmabile

L'Architettura Programmabile (fig. 3.3.1) è stata addestrata, per un campione totale di venti PNN, e relative FNN, sul *dataset* con il labirinto base. Prima di osservare i risultati, facciamo alcune brevi considerazioni sulle FNN. Per come è strutturata l'AP, questa rete deve occuparsi solo dell'apprendimento della codifica di programma e non dell'apprendimento dei comportamenti; è però vero che una codifica errata potrebbe influenzare negativamente il lavoro della PNN. Nel nostro caso la FNN apprende le codifiche di tutti i programmi nel 100% dei casi e, pertanto, non risulta utile, ai fini della valutazione delle prestazioni, presentarne dei risultati sperimentali.

La PNN, d'altro canto, essendo una CTRNN dotata di programmabilità, così come spiegato nel capitolo 2, ha prestazioni confrontabili con una rete CTRNN full connected che costituisce l'ANP scelta per il raffronto. Come si può vedere dalle tabelle 4.7 e 4.8, la rete PNN ha prestazioni di gran lunga superiori rispetto all'ANP e, infatti, riesce ad apprendere con successo otto programmi su otto, nel 40% dei casi e, sempre e comunque, almeno sette programmi, appresi solamente nel 60% dei casi (tab. 4.9). Il programma che presenta più difficoltà ad essere appreso risulta essere P_3, la cui frequenza di successo è del 45%.

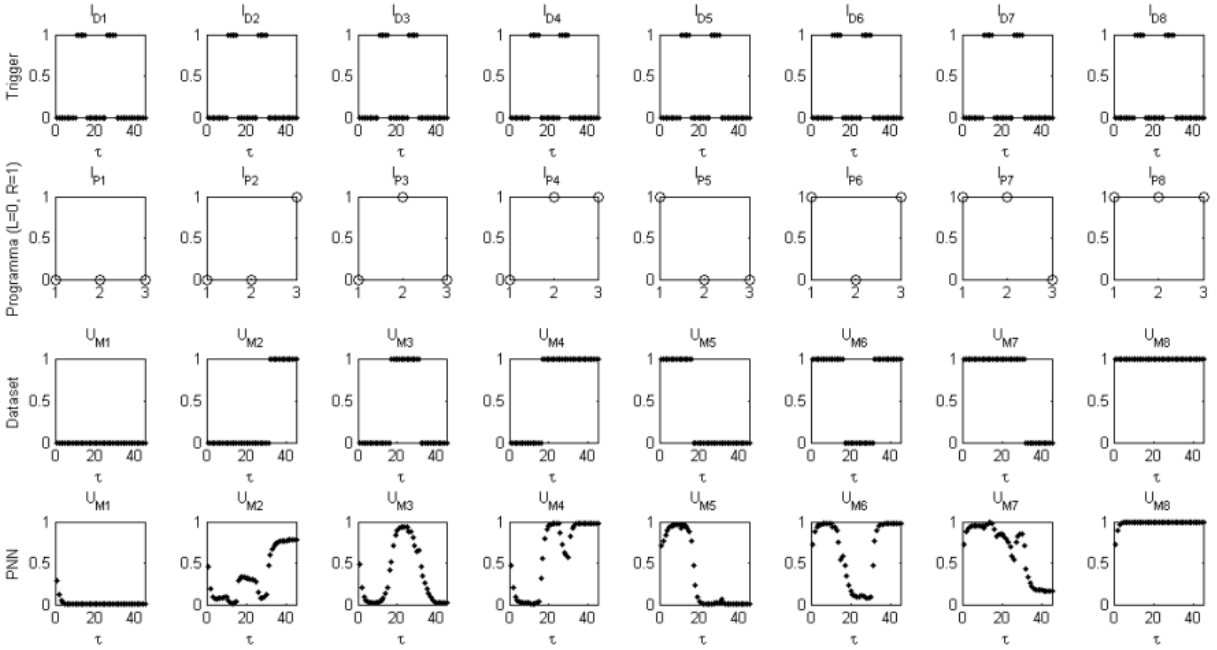


Figura 4.2.4: *Risposta della rete PNN numero 14 (*best case*) addestrata con il* dataset *per il labirinto base, ovvero con*$\lambda = 2$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). I programmi sono tutti e otto corretti, infatti, tutte le uscite* U_M *del dataset e della PNN hanno andamenti - al di sotto e al di sopra della soglia a 0,5 - coincidenti. La lettura di* U_M *come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5* τ *dall'istante iniziale e dopo ogni discesa del segnale di* trigger.

Tabella 4.7: *Errori della PNN addestrata con il labirinto base. Grazie all'uso di un'AP, tutti i programmi possono essere appresi correttamente. Tra le 20 esecuzioni si possono isolare un* best case *(fig. 4.2.4), con 8 su 8 programmi appresi, ed un* worst case*, con 7 su 8 programmi appresi (fig. 4.2.5).*

PNN	**Errore del programma (Numero di svolte corrette)**							
	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
1	0.005 (3/3)	0.079 (3/3)	15.155 (2/3)	0.046 (3/3)	0.022 (3/3)	0.073 (3/3)	0.040 (3/3)	0.005 (3/3)
2	0.005 (3/3)	0.068 (3/3)	15.155 (2/3)	0.050 (3/3)	0.031 (3/3)	0.082 (3/3)	0.093 (3/3)	0.005 (3/3)
3	0.005 (3/3)	0.072 (3/3)	15.155 (2/3)	0.041 (3/3)	0.028 (3/3)	0.040 (3/3)	0.025 (3/3)	0.005 (3/3)
4	0.005 (3/3)	0.097 (3/3)	0.092 (3/3)	0.038 (3/3)	0.021 (3/3)	0.042 (3/3)	0.094 (3/3)	0.005 (3/3)
5	0.005 (3/3)	0.095 (3/3)	1.033 (3/3)	0.048 (3/3)	0.054 (3/3)	0.069 (3/3)	0.087 (3/3)	0.005 (3/3)
6	0.005 (3/3)	0.076 (3/3)	15.155 (2/3)	0.060 (3/3)	0.048 (3/3)	0.055 (3/3)	0.046 (3/3)	0.005 (3/3)
7	0.005 (3/3)	0.077 (3/3)	0.094 (3/3)	0.040 (3/3)	0.043 (3/3)	0.055 (3/3)	14.145 (2/3)	0.005 (3/3)
8	0.005 (3/3)	0.100 (3/3)	15.155 (2/3)	0.037 (3/3)	0.030 (3/3)	0.052 (3/3)	0.047 (3/3)	0.005 (3/3)
9	0.005 (3/3)	0.051 (3/3)	15.155 (2/3)	0.061 (3/3)	0.034 (3/3)	0.088 (3/3)	0.048 (3/3)	0.005 (3/3)
10	0.005 (3/3)	0.052 (3/3)	15.155 (2/3)	0.079 (3/3)	0.027 (3/3)	0.046 (3/3)	0.092 (3/3)	0.005 (3/3)
11	0.005 (3/3)	0.092 (3/3)	0.059 (3/3)	0.048 (3/3)	0.026 (3/3)	0.063 (3/3)	0.080 (3/3)	0.005 (3/3)
12	0.005 (3/3)	0.074 (3/3)	0.075 (3/3)	0.046 (3/3)	0.018 (3/3)	0.052 (3/3)	0.094 (3/3)	0.005 (3/3)
13	0.005 (3/3)	0.078 (3/3)	1.033 (3/3)	1.020 (3/3)	0.069 (3/3)	0.057 (3/3)	0.049 (3/3)	0.005 (3/3)
14 (*best case*)	0.005 (3/3)	0.091 (3/3)	0.063 (3/3)	0.044 (3/3)	0.027 (3/3)	0.053 (3/3)	0.075 (3/3)	0.005 (3/3)
15 (*worst case*)	0.005 (3/3)	0.079 (3/3)	15.155 (2/3)	0.079 (3/3)	0.092 (3/3)	0.053 (3/3)	0.087 (3/3)	0.005 (3/3)
16	0.005 (3/3)	0.067 (3/3)	1.033 (3/3)	0.046 (3/3)	0.022 (3/3)	0.042 (3/3)	0.062 (3/3)	0.005 (3/3)
17	0.005 (3/3)	0.085 (3/3)	15.155 (2/3)	0.059 (3/3)	0.029 (3/3)	0.058 (3/3)	0.066 (3/3)	0.005 (3/3)
18	0.005 (3/3)	0.076 (3/3)	0.072 (3/3)	0.064 (3/3)	0.070 (3/3)	0.058 (3/3)	0.053 (3/3)	0.005 (3/3)
19	0.005 (3/3)	0.093 (3/3)	9.097 (2/3)	0.029 (3/3)	0.034 (3/3)	0.057 (3/3)	0.088 (3/3)	0.005 (3/3)
20	0.005 (3/3)	0.092 (3/3)	15.155 (2/3)	0.056 (3/3)	0.044 (3/3)	0.064 (3/3)	0.093 (3/3)	0.005 (3/3)
Successo (3/3)	100%	100%	45%	100%	100%	100%	95%	100%

Tabella 4.8: *Statistica dei risultati per la PNN addestrata con il labirinto base. L'apprendimento mostra una buona capacità dell'AP ad apprendere comportamenti multipli e più che sufficiente ad apprenderne anche 8 su 8 proposti. La deviazione standard è leggermente alta per i programmi che mostrano una certa difficoltà ad essere appresi (P_3 e P_7), lo si può notare dal raffronto diretto con la loro frequenza di successo. In ogni caso, P_3 è l'unico programma che ha anche una media d'errore leggermente alta, ovvero tale da sbagliare al massimo solo una svolta delle tre previste per raggiungere la terminazione corretta del labirinto, ma comunque rendendo errato l'intero programma, concordemente con la sua frequenza di successo (3/3 svolte corrette) al di sotto del 50%.*

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
Errore medio	0.005	0.080	8.210	0.100	0.038	0.058	0.773	0.005
Deviazione standard	0.000	0.014	7.379	0.217	0.019	0.013	3.147	0.000
Errore massimo	0.005	0.100	15.155	1.020	0.092	0.088	14.145	0.005
Errore minimo	0.005	0.051	0.059	0.029	0.018	0.040	0.025	0.005
Frequenza di successo	100%	100%	45%	100%	100%	100%	95%	100%

Tabella 4.9: *Frequenza di successo per la PNN addestrata con il labirinto base. Con questo tipo di addestramento l'AP riesce ad apprendere 8 comportamenti su 8 nel* best case *(40% dei casi) e 7 su 8 nel* worst case *(il restante 60%).*

Programmi appresi	1/8	2/8	3/8	4/8	5/8	6/8	7/8	8/8
Frequenza di successo	100%	100%	100%	100%	100%	100%	100%	40%

Tabella 4.10: *Passi d'apprendimento della PNN con il labirinto base. Il limite massimo di passi totali per l'apprendimento della PNN è impostato a* $S^{PNN} = 2 \cdot S^{online} = 2 \cdot N_P \cdot S^{batch} = 2 \cdot 8 \cdot 2500 = 40000$ *passi totali, che corrispondono a 5000 passi massimi per ciascun programma. In realtà, come si vede dalla tabella, il massimo di passi raggiunto è di 12500 che è di gran lunga inferiore rispetto ai 20000 passi massimi previsti per l'apprendimento dell'ANP in modalità* online*. L'apprendimento a soglia ha ridotto da un minimo di* $40000 - 12500 = 27500$ *passi, ad un massimo di* $40000 - 2450 = 37550$ *passi, l'apprendimento* standard *per una PNN. Anche se non si evince dai dati finali, un programma che non converga ad un minimo globale entro "pochi" passi d'apprendimento, convergerà comunque dopo "pochi" passi d'apprendimento ad un minimo locale entro il quale si manterrà fino a fine apprendimento, ovvero, nel nostro caso, al 5000° passo, a meno che non si protragga l'apprendimento all'infinito. In questo senso, allora, una soglia massima di 2500 passi d'apprendimento per programma (pari a quella impostata per l'ANP) avrebbe fornito risultati, a meno di affinamenti degli errori di ordine inferiore, quasi del tutto identici. L'algoritmo di apprendimento a soglia è dunque più efficiente sia di quello* online *che di quello* batch *perché ne acquista, di entrambi, i vantaggi: meno passi totali (modalità* batch*) e la possibilità, trattandosi di PNN, di apprendere i programmi singolarmente (modalità* online*), con maggior efficacia (minimizzazione dell'errore) sui singoli programmi.*

PNN	**Passi d'apprendimento per programma**								**Passi totali**
	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	
1	50	100	5000	100	100	1050	350	50	6800
2	50	200	5000	150	100	900	700	50	7150
3	50	200	5000	100	100	550	450	50	6500
4	50	150	1150	250	100	1100	550	50	3400
5	50	100	5000	200	50	700	250	50	6400
6	50	150	5000	100	50	1000	850	50	7250
7	50	250	800	200	50	750	5000	50	7150
8	50	150	5000	150	100	700	900	50	7100
9	50	150	5000	100	50	600	400	50	6400
10	50	150	5000	250	100	550	350	50	6500
11	50	200	1000	100	100	450	500	50	2450
12	50	300	800	100	100	850	450	50	2700
13	50	150	5000	5000	50	650	1550	50	12500
14	50	400	600	150	50	900	900	50	3100
15	50	150	5000	50	50	750	1000	50	7100
16	50	150	5000	100	100	950	400	50	6800
17	50	150	5000	100	50	1050	300	50	6750
18	50	350	850	100	50	600	1350	50	3400
19	50	200	5000	350	50	550	650	50	6900
20	50	200	5000	150	50	950	300	50	6750
Media	50	192	3760	390	72	780	860	50	6155
Dev. standard	0	78	1946	1087	25	199	1037	0	2269
Massimo	50	400	5000	5000	100	1100	5000	50	12500
Minimo	50	100	600	50	50	450	250	50	2450

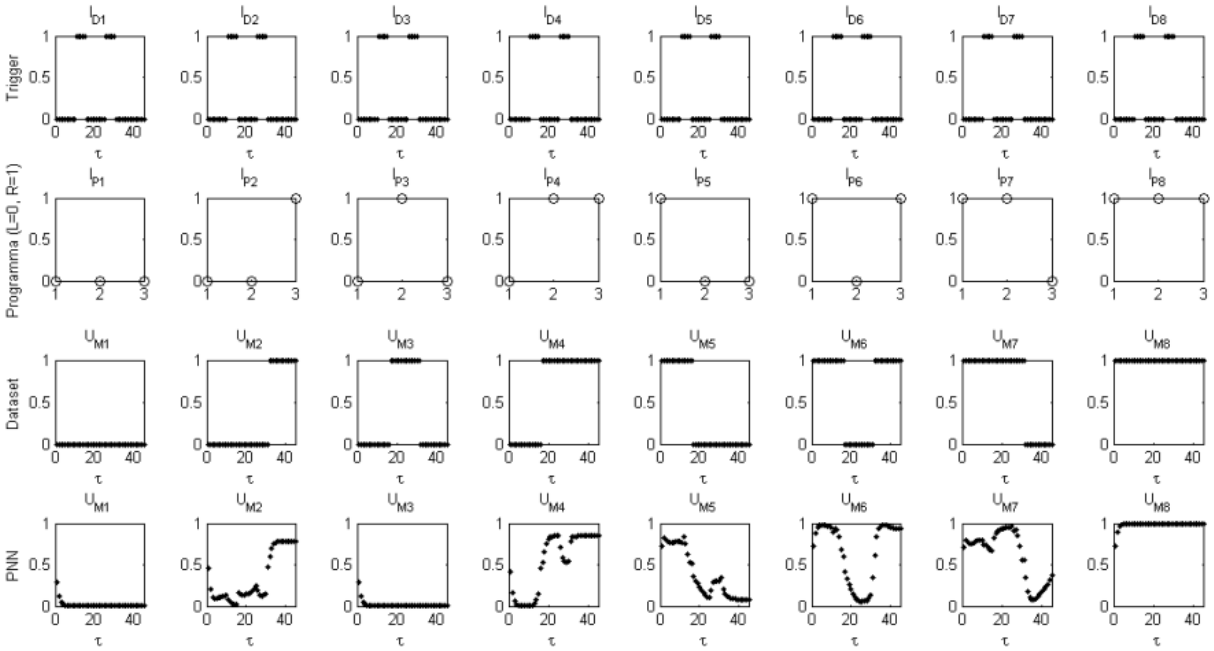


Figura 4.2.5: *Risposta della rete PNN numero 15 (*worst case*) addestrata con il* dataset *per il labirinto base, ovvero con*$\lambda = 2$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). I programmi corretti sono* P_1, P_2, P_4, P_5, P_6, P_7 *e* P_8, *per i quali le uscite* U_M *del dataset e della PNN hanno andamenti - al di sotto e al di sopra della soglia a 0,5 - coincidenti. La lettura di* U_M *come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5* τ *dall'istante iniziale e dopo ogni discesa del segnale di* trigger.

4.2.4 Confronto dei risultati

Dai risultati dell'apprendimento con il *dataset* per il labirinto base, risulta che i programmi della PNN vengono appresi tutti e otto nel 40% dei casi (tab. 4.9) e, mediamente, in soli 769 passi d'apprendimento (tab. 4.10). Nell'altro 60% dei casi, vengono appresi comunque 7 programmi su 8. Un risultato notevole rispetto a quello della CTRNN addestrata in modalità *batch* che ne apprende massimo 5 nell'85% dei casi e almeno 4 nel restante 15% (tab. 4.6) con 2500 passi fissi d'apprendimento, oppure rispetto a quello della CTRNN addestrata in modalità *online* che riesce ad apprendere uno ed un solo programma nel 100% dei casi (tab. 4.3), con 20000 passi fissi d'apprendimento.

In sintesi, lo scenario di base, cioè con un apprendimento limitato ad un solo labirinto (§ 3.1.4.1), favorisce di gran lunga l'Architettura Programmabile composta da PNN e FNN, piuttosto che l'Architettura Non Programmabile composta da una CTRNN *full connected*, sia in termini di successo dell'apprendimento, con almeno 3 programmi in più appresi, che in termini di risparmio di risorse computazionali, visto che sono necessari meno passi d'apprendimento per raggiungere un risultato addirittura migliore.

L'Architettura Programmabile si dimostra l'unica in grado di apprendere tutti e otto i comportamenti proposti.

4.3 Apprendimento del dataset con tre labirinti

I tre labirinti scelti per questa sessione d'apprendimento corrispondono al *dataset* descritto al paragrafo 3.1.4.2. La scelta di un *dataset* più complesso è motivata dalla necessità di insegnare al *robot* a generalizzare i comportamenti multipli appresi per più di uno scenario dalle caratteristiche simili. L'architettura di controllo deve cioè interpretare degli input con una maggiore variabilità e consentire al *robot* di continuare a manifestare gli stessi comportamenti correttamente. Imponiamo e verifichiamo, in tal modo, la condizione necessaria che richiede di apprendere comportamenti e non semplici traiettorie.

I labirinti scelti corrispondono ai parametri $\lambda = 2$, $\lambda = 3$ e $\lambda = 4$ che ne esprimono sinteticamente la grandezza in termini adimensionali, sfruttando la proporzionalità tra i tempi di percorrenza e le lunghezze dei tratti rettilinei percorsi dal *robot* a velocità costante (eq. 3.1.3).

4.3.1 Architettura Non Programmabile in modalità *online*

L'Architettura Non Programmabile (fig. 3.2.1) è stata addestrata anche con tre labirinti di grandezza diversa, ovvero per $\lambda = 2$,$\lambda = 3$ e $\lambda = 4$ (parametro adimensionale proporzionale alle dimensioni del labirinto), in modalità *online* (nel paragrafo successivo

troviamo i risultati per l'altro tipo di addestramento, in modalità *batch*), ovvero alternando l'apprendimento dei singoli programmi, in ordine crescente di nome, per un numero fissato di passi e reiterando l'intera sequenza più volte. Sebbene questo tipo di apprendimento porti a buoni risultati nell'ambito della ricerca di una soluzione ottimale per il singolo programma, l'alternarsi degli obiettivi finisce per confondere la funzione d'errore che viene ottimizzata ora in un senso, ora verso un altro tipo di soluzione; nel complesso non si raggiunge una soluzione in grado di portare l'errore a zero per tutti i programmi. In particolare, quanto appena detto, può essere subito osservato dalle tabelle 4.11 e 4.12 dei risultati e 4.13 con l'analisi statistica dei risultati dove si evince che il programma P_8 ha prestazioni di gran lunga migliori degli altri sette, per l'esattezza, è anche l'unico (tab. 4.14) che svolga il suo compito a dovere, portando il robot alla corretta terminazione del labirinto, in tutte e 20 le prove effettuate.

Un altro fenomeno da osservare è quello che potremmo definire uno stallo di questo algoritmo d'apprendimento: per quanti cicli d'apprendimento si possa pensare di eseguire, solo gli ultimi S_P passi d'apprendimento riguardanti ciascuno degli otto programmi sono quelli che hanno impatto sull'errore finale, questo perché ogni volta che la *fitness* ricomincia decrescere minimizzando l'errore sul primo programma, ovvero inizia un nuovo ciclo esterno, tende subito a "dimenticare" i risultati ottenuti al ciclo precedente. In definitiva, se il numero di passi d'apprendimento scelto è stato:

$$S = S_P \cdot N_P \cdot C = 20000$$

con cicli $C = 50$, passi d'apprendimento $S_P = 50$, numero di programmi $N_P = 8$, allora, avremmo anche potuto scegliere:

$$S = S_P \cdot N_P \cdot C = 400$$

con $C = 1$, $S_P = 50$, $N_P = 8$, ed ottenere risultati identici. Ecco perché non ci deve meravigliare un risultato identico per tutte le prove: un addestramento per singolo programma così rapido ($S_P = 50$), a partire dalla stessa inizializzazione della rete neurale, porta la fitness a raggiungere sempre gli stessi valori (tabb. 4.11 e 4.12) con conseguente deviazione standard nulla (tab. 4.13).

Nelle figure 4.3.1, 4.3.2 e 4.3.3 vediamo i risultati di una qualsiasi rete, da considerarsi *best e worst case* contemporaneamente (tutte le reti si comportano allo stesso modo, pertanto i due risultati coincidono). Si nota subito come l'output U_{M8} della rete CTRNN si sposi perfettamente con quello atteso del *dataset*, in tutti e tre i labirinti. La rete CTRNN ha infatti appreso perfettamente solo l'ultimo comportamento che consente al *robot* di raggiungere l'ottava terminazione del labirinto, e si comporta sempre allo stesso

modo, qualsiasi sia il programma P_i presentato in ingresso.

Tabella 4.13: *Statistica dei risultati per la CTRNN addestrata in modalità* online *con tre labirinti di grandezza diversa, ovvero per* $\lambda = 2$,$\lambda = 3$ *e* $\lambda = 4$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). L'apprendimento mostra chiaramente un'incapacità di raggiungere soluzioni ottimali per la rete con la conseguente tendenza della fitness a favorire l'ultimo programma appreso, l'unico con errore medio quasi nullo. Inoltre, l'alto numero di iterazioni effettuate (*$S = S_P \cdot N_P \cdot C = 20000$*) non consente alcuna differenziazione dei risultati tra una prova e l'altra, infatti, la deviazione standard risulta nulla in tutti i casi e i valori massimi e minimi d'errore coincidono.*

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
Errore medio	181.791	124.221	121.191	63.621	118.179	60.609	57.579	0.009
Deviazione standard	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.000
Errore massimo	181.791	124.221	121.191	63.621	118.179	60.609	57.579	0.009
Errore minimo	181.790	124.221	121.191	63.621	118.179	60.609	57.579	0.009
Frequenza di successo	0%	0%	0%	0%	0%	0%	0%	100%

Tabella 4.14: *Frequenza di successo per la CTRNN addestrata in modalità* online *con tre labirinti di grandezza diversa, ovvero per* $\lambda = 2$,$\lambda = 3$ *e* $\lambda = 4$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). Con questo tipo di addestramento l'ANP riesce ad apprendere uno ed un solo comportamento.*

Programmi appresi	1/8	2/8	3/8	4/8	5/8	6/8	7/8	8/8
Frequenza di successo	100%	0%	0%	0%	0%	0%	0%	0%

Tabella 4.11: *Errori dei programmi (tabella da P_1 a P_4) della CTRNN addestrata in modalità* online *con tre labirinti di grandezza diversa, ovvero per $\lambda = 2$,$\lambda = 3$ e $\lambda = 4$ (parametro adimensionale proporzionale alle dimensioni del labirinto). L'apprendimento, osservando anche la tabella che riporta i risultati per gli altri quattro programmi, mostra chiaramente un'incapacità di raggiungere soluzioni ottimali per la rete con la conseguente tendenza della fitness a favorire l'ultimo programma appreso. Inoltre, anche in luogo dell'alto numero di iterazioni effettuate ($S = S_P \cdot N_P \cdot C = 20000$), non consente alcuna differenziazione dei risultati tra una prova e l'altra a causa di un'inefficienza dell'algoritmo utilizzato per l'apprendimento di comportamenti multipli.*

CTRNN (***online***)	**Errore del programma (Numero di svolte corrette)**											
	$\boldsymbol{P_1}$			$\boldsymbol{P_2}$			$\boldsymbol{P_3}$			$\boldsymbol{P_4}$		
	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$
1	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
2	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
3	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
4	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
5	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
6	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
7	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
8	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
9	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
10	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
11	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
12	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
13	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
14	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
15	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
16	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
17	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
18	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
19	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
20	45.447 (0/3)	60.597 (0/3)	75.747 (0/3)	31.307 (1/3)	41.407 (1/3)	51.507 (1/3)	30.297 (1/3)	40.397 (1/3)	50.497 (1/3)	16.157 (2/3)	21.207 (2/3)	26.257 (2/3)
Successo	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%

Tabella 4.12: *Errori dei programmi (tabella da P_5 a P_8) della CTRNN addestrata in modalità* online *con tre labirinti di grandezza diversa, ovvero per $\lambda = 2$,$\lambda = 3$ e $\lambda = 4$ (parametro adimensionale proporzionale alle dimensioni del labirinto). L'apprendimento, osservando anche la tabella che riporta i risultati per gli altri quattro programmi, mostra chiaramente un'incapacità di raggiungere soluzioni ottimali per la rete con la conseguente tendenza della fitness a favorire l'ultimo programma appreso. Inoltre, anche in luogo dell'alto numero di iterazioni effettuate ($S = S_P \cdot N_P \cdot C = 20000$), non consente alcuna differenziazione dei risultati tra una prova e l'altra a causa di un'inefficienza dell'algoritmo utilizzato per l'apprendimento di comportamenti multipli.*

CTRNN (***online***)	**Errore del programma (Numero di svolte corrette)**											
	$\boldsymbol{P_5}$			$\boldsymbol{P_6}$			$\boldsymbol{P_7}$			$\boldsymbol{P_8}$		
	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$
1	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
2	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
3	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
4	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
5	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
6	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
7	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
8	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
9	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
10	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
11	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
12	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
13	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
14	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
15	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
16	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
17	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
18	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
19	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
20	29.293 (1/3)	39.393 (1/3)	49.493 (1/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
Successo	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	100%	100%

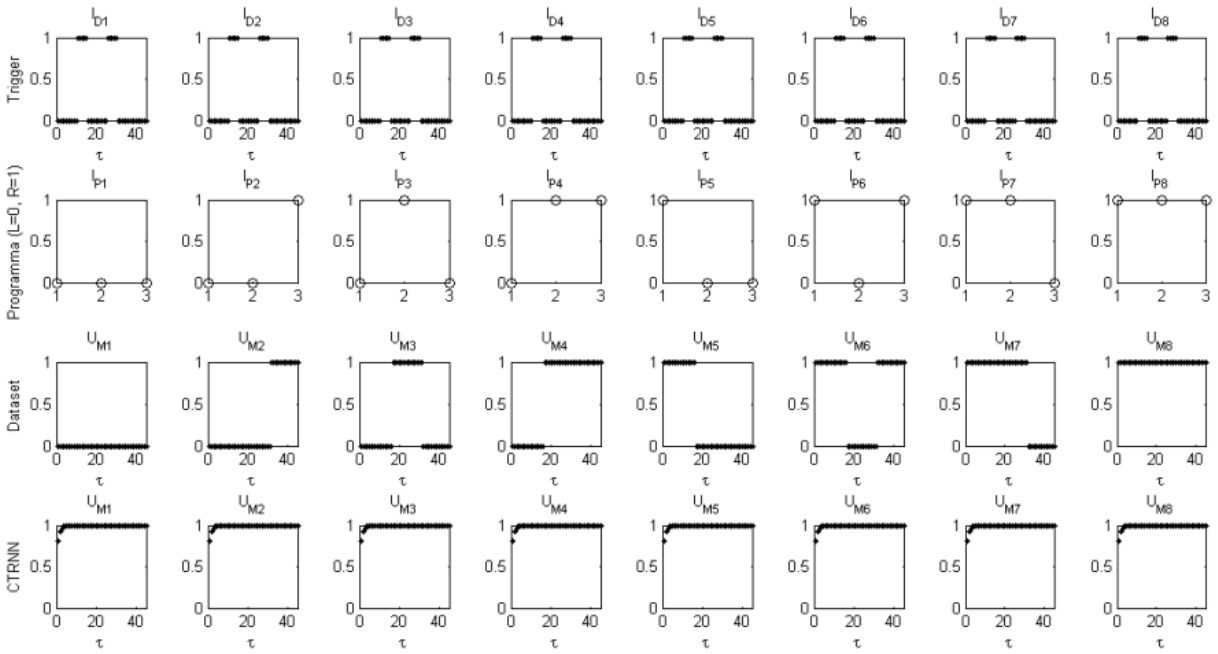


Figura 4.3.1: *Risposta della rete CTRNN numero 1 (*best/worst case*), addestrata in modalità* online*, con il* dataset *da tre labirinti, al labirinto con* $\lambda = 2$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). L'unico programma corretto è il* P_8*, per il quale le uscite* U_M *del dataset e della CTRNN - al di sotto e al di sopra della soglia a 0,5 - coincidono. La lettura di* U_M *come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5* τ *dall'istante iniziale e dopo ogni discesa del segnale di* trigger.

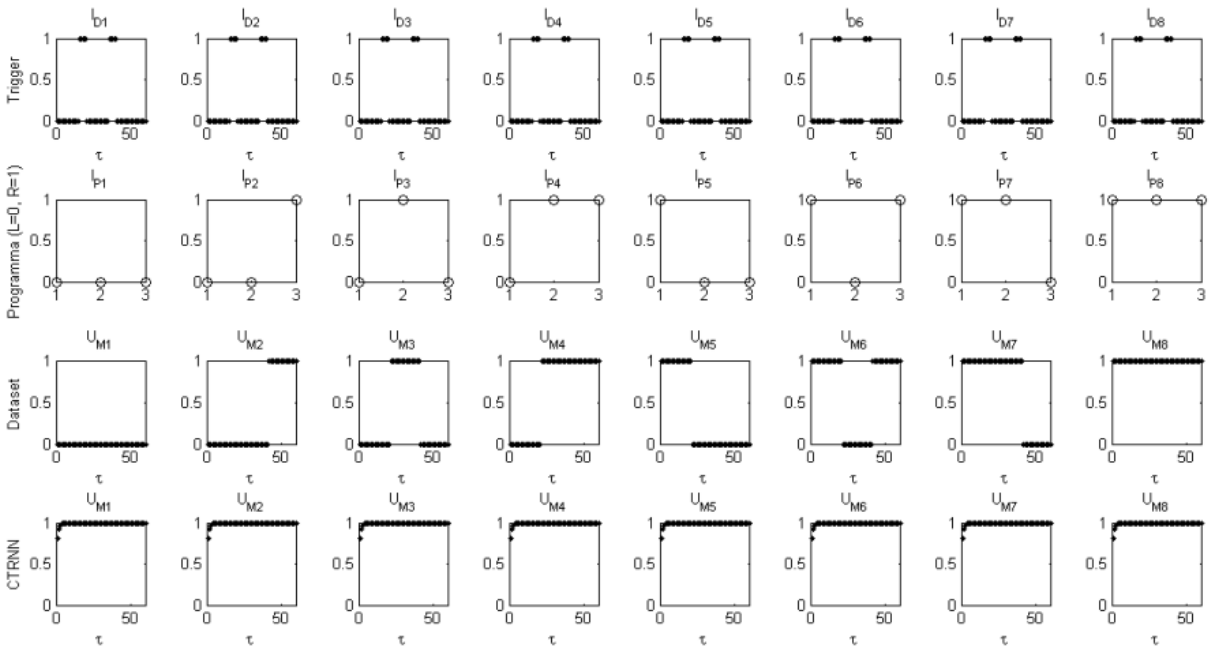


Figura 4.3.2: *Risposta della rete CTRNN numero 1 (*best/worst case*), addestrata in modalità* online*, con il* dataset *da tre labirinti, al labirinto con* $\lambda = 3$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). L'unico programma corretto è il* P_8*, per il quale le uscite* U_M *del dataset e della CTRNN - al di sotto e al di sopra della soglia a 0,5 - coincidono. La lettura di* U_M *come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5* τ *dall'istante iniziale e dopo ogni discesa del segnale di* trigger.

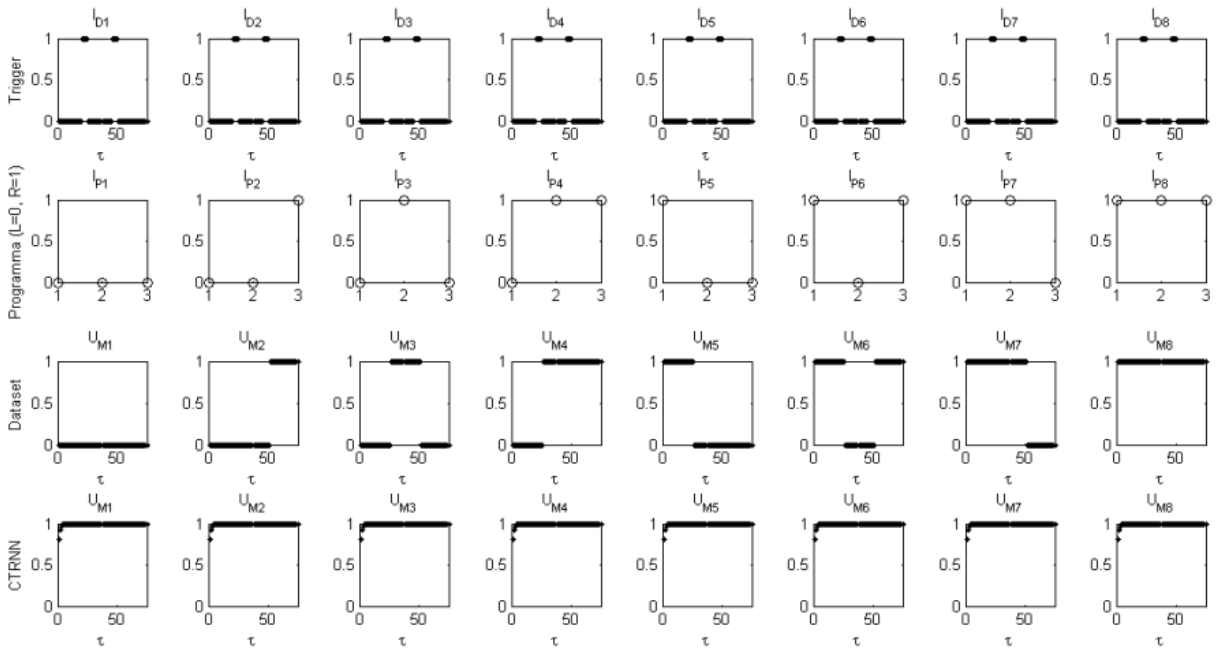


Figura 4.3.3: *Risposta della rete CTRNN numero 1 (*best/worst case*), addestrata in modalità* online*, con il* dataset *da tre labirinti, al labirinto con* $\lambda = 4$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). L'unico programma corretto è il* P_8*, per il quale le uscite* U_M *del dataset e della CTRNN - al di sotto e al di sopra della soglia a 0,5 - coincidono. La lettura di* U_M *come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5* τ *dall'istante iniziale e dopo ogni discesa del segnale di* trigger.

4.3.2 Architettura Non Programmabile in modalità *batch*

L'Architettura Non Programmabile (fig. 3.2.1) addestrata in modalità *batch* con tre labirinti di grandezza diversa, ovvero per $\lambda = 2$,$\lambda = 3$ e $\lambda = 4$ (parametro adimensionale proporzionale alle dimensioni del labirinto), presenta, rispetto alla modalità *online*, risultati di gran lunga migliori (tabb. 4.15, 4.16 e 4.17). Questa volta, infatti, i programmi P_1, P_5 e P_8 hanno una percentuale di successo del 100%. La CTRNN apprende almeno 4 su 8 programmi nel 100% dei casi e anche un quinto programma nell'70% dei casi (tab. 4.18). Come possiamo vedere dalle figure 4.3.4, 4.3.5 e 4.3.6 la CTRNN, nella quinta esecuzione che è il *best case*, presenta, per i programmi P_1, P_2, P_4, P_5 e P_8, output analoghi a quelli del *dataset* per il labirinto base.

Il *worst case*, corrispondente alla terza rete appresa, del resto, presenta giusto un programma in meno rispetto al *best case* (figg. 4.3.7, 4.3.8 e 4.3.9). Questo risultato è dovuto al tipo di algoritmo di apprendimento scelto: la modalità *batch*, infatti, consente di minimizzare gli errori dei singoli programmi, tenendo però sempre in memoria l'intera sequenza di uscite corrispondente a tutti i possibili ingressi. Un lavoro che porta a risultati non ottimali, ma significativamente buoni per quanto riguarda la differenziazione dei possibili comportamenti appresi.

Tabella 4.17: *Statistica dei risultati per la CTRNN addestrata in modalità* batch *con tre labirinti di grandezza diversa, ovvero per $\lambda = 2$,$\lambda = 3$ e $\lambda = 4$ (parametro adimensionale proporzionale alle dimensioni del labirinto). L'apprendimento mostra una buona capacità dell'ANP ad apprendere comportamenti multipli ma non sufficiente ad apprenderne 8 su 8 proposti. I risultati mostrano per tutti i programmi una deviazione standard più bassa rispetto alla media e ciò suggerisce che il presentarsi di un risultato migliore, anche a fronte di un numero superiore di prove, non sia auspicabile.*

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
Errore medio	0.016	36.360	45.518	9.205	2.661	60.918	57.583	0.012
Deviazione standard	0.013	23.197	21.472	5.500	2.975	0.934	0.004	0.002
Errore massimo	0.056	57.585	60.675	17.422	9.110	63.657	57.591	0.017
Errore minimo	0.010	2.484	6.158	0.087	0.076	60.610	57.580	0.010
Frequenza di successo	100%	35%	30%	95%	100%	0%	0%	100%

Tabella 4.15: *Errori dei programmi (tabella da P_1 a P_4) della CTRNN addestrata in modalità* batch *con tre labirinti di grandezza diversa, ovvero per $\lambda = 2$,$\lambda = 3$ e $\lambda = 4$ (parametro adimensionale proporzionale alle dimensioni del labirinto). Più programmi, ma non tutti, vengono appresi correttamente. Tra le 20 esecuzioni si possono isolare un* best case *(figg. 4.3.4, 4.3.5 e 4.3.6), con 5 su 8 programmi appresi, ed un* worst case*, con 4 su 8 programmi appresi (figg. 4.3.7, 4.3.8 e 4.3.9).*

CTRNN (*batch*)	**Errore del programma (Numero di svolte corrette)**											
	$\boldsymbol{P_1}$			$\boldsymbol{P_2}$			$\boldsymbol{P_3}$			$\boldsymbol{P_4}$		
	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$
1	0.004 (3/3)	0.004 (3/3)	0.004 (3/3)	1.065 (3/3)	11.131 (3/3)	16.181 (3/3)	15.161 (2/3)	20.214 (2/3)	25.268 (2/3)	1.029 (3/3)	1.031 (3/3)	1.034 (3/3)
2	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)	12.134 (2/3)	10.144 (2/3)	4.132 (2/3)	15.171 (2/3)	20.225 (2/3)	25.279 (2/3)	3.044 (3/3)	3.047 (3/3)	3.050 (3/3)
3 (*worst*)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)	14.144 (2/3)	19.194 (2/3)	24.244 (2/3)	15.159 (2/3)	20.210 (2/3)	25.261 (2/3)	1.062 (3/3)	0.074 (3/3)	0.087 (3/3)
4	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)	9.134 (2/3)	9.164 (2/3)	19.238 (2/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	3.050 (3/3)	3.054 (3/3)	3.056 (3/3)
5 (*best*)	0.011 (3/3)	0.015 (3/3)	0.019 (3/3)	2.120 (3/3)	3.155 (3/3)	2.189 (3/3)	15.158 (2/3)	20.209 (2/3)	25.261 (2/3)	1.094 (3/3)	1.114 (3/3)	1.136 (3/3)
6	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)	6.141 (3/3)	5.173 (3/3)	3.196 (3/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	4.086 (3/3)	4.109 (3/3)	5.133 (3/3)
7	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	4.054 (3/3)	4.058 (3/3)	4.061 (3/3)	3.041 (3/3)	3.044 (3/3)	3.047 (3/3)
8	0.006 (3/3)	0.007 (3/3)	0.008 (3/3)	3.121 (3/3)	1.157 (3/3)	5.207 (2/3)	15.155 (2/3)	20.205 (2/3)	25.256 (2/3)	7.118 (2/3)	5.141 (2/3)	5.162 (2/3)
9	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	8.112 (3/3)	8.137 (3/3)	8.164 (3/3)	5.070 (3/3)	5.081 (3/3)	5.093 (3/3)
10	0.008 (3/3)	0.010 (3/3)	0.013 (3/3)	0.136 (3/3)	1.177 (3/3)	3.223 (3/3)	15.156 (2/3)	20.208 (2/3)	25.259 (2/3)	4.095 (3/3)	5.126 (3/3)	5.159 (3/3)
11	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	2.050 (3/3)	2.053 (3/3)	2.055 (3/3)	4.055 (3/3)	4.057 (3/3)	4.059 (3/3)
12	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)	3.103 (3/3)	4.139 (3/3)	4.173 (3/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	4.089 (3/3)	4.110 (3/3)	4.132 (3/3)
13	0.015 (3/3)	0.019 (3/3)	0.022 (3/3)	0.127 (3/3)	1.161 (3/3)	1.196 (3/3)	15.157 (2/3)	20.208 (2/3)	25.259 (2/3)	1.090 (3/3)	2.112 (3/3)	2.136 (3/3)
14	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)	14.144 (2/3)	19.194 (2/3)	24.245 (2/3)	6.139 (3/3)	7.195 (3/3)	12.266 (2/3)	5.111 (3/3)	5.142 (3/3)	5.174 (3/3)
15	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	6.079 (3/3)	7.084 (3/3)	7.089 (3/3)	2.098 (3/3)	2.123 (3/3)	2.147 (3/3)
16	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	5.082 (3/3)	5.103 (3/3)	5.123 (3/3)	3.062 (3/3)	3.071 (3/3)	3.081 (3/3)
17	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)	14.144 (2/3)	19.194 (2/3)	24.244 (2/3)	15.156 (2/3)	20.207 (2/3)	25.257 (2/3)	0.040 (3/3)	0.045 (3/3)	0.049 (3/3)
18	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)	14.145 (2/3)	19.195 (2/3)	24.245 (2/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	0.027 (3/3)	0.029 (3/3)	0.031 (3/3)
19	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)	3.052 (3/3)	3.057 (3/3)	3.059 (3/3)	15.154 (2/3)	20.204 (2/3)	25.254 (2/3)	5.057 (3/3)	5.062 (3/3)	5.066 (3/3)
20	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)	14.144 (2/3)	19.194 (2/3)	24.244 (2/3)	6.094 (3/3)	6.112 (3/3)	6.131 (3/3)	4.096 (3/3)	4.127 (3/3)	4.167 (3/3)
Successo (3/3)	100%	100%	100%	40%	40%	35%	35%	35%	30%	95%	95%	95%

Tabella 4.16: *Errori dei programmi (tabella da P_5 a P_8) della CTRNN addestrata in modalità* batch *con tre labirinti di grandezza diversa, ovvero per $\lambda = 2$,$\lambda = 3$ e $\lambda = 4$ (parametro adimensionale proporzionale alle dimensioni del labirinto). Più programmi, ma non tutti, vengono appresi correttamente. Tra le 20 esecuzioni si possono isolare un* best case *(figg. 4.3.4, 4.3.5 e 4.3.6), con 5 su 8 programmi appresi, ed un* worst case*, con 4 su 8 programmi appresi (figg. 4.3.7, 4.3.8 e 4.3.9).*

CTRNN (*batch*)	**Errore del programma (Numero di svolte corrette)**											
	P_5			P_6			P_7			P_8		
	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$
1	2.042 (3/3)	2.047 (3/3)	2.053 (3/3)	15.154 (2/3)	20.204 (2/3)	25.254 (2/3)	14.145 (2/3)	19.195 (2/3)	24.245 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
2	2.026 (3/3)	2.028 (3/3)	2.030 (3/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.004 (3/3)	0.004 (3/3)	0.004 (3/3)
3 (*worst*)	0.035 (3/3)	0.038 (3/3)	0.042 (3/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.004 (3/3)
4	1.027 (3/3)	1.029 (3/3)	1.031 (3/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	14.144 (2/3)	19.194 (2/3)	24.244 (2/3)	0.004 (3/3)	0.004 (3/3)	0.005 (3/3)
5 (*best*)	1.034 (3/3)	1.038 (3/3)	1.041 (3/3)	15.154 (2/3)	20.204 (2/3)	25.254 (2/3)	14.144 (2/3)	19.195 (2/3)	24.245 (2/3)	0.004 (3/3)	0.004 (3/3)	0.004 (3/3)
6	0.029 (3/3)	0.031 (3/3)	0.034 (3/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.145 (2/3)	19.196 (2/3)	24.246 (2/3)	0.003 (3/3)	0.004 (3/3)	0.004 (3/3)
7	2.045 (3/3)	2.049 (3/3)	2.054 (3/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
8	3.035 (3/3)	3.037 (3/3)	3.038 (3/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.146 (2/3)	19.196 (2/3)	24.246 (2/3)	0.004 (3/3)	0.004 (3/3)	0.004 (3/3)
9	0.024 (3/3)	0.025 (3/3)	0.027 (3/3)	15.157 (2/3)	20.208 (2/3)	25.258 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.004 (3/3)	0.004 (3/3)	0.004 (3/3)
10	2.042 (3/3)	2.044 (3/3)	2.047 (3/3)	15.153 (2/3)	20.203 (2/3)	25.253 (2/3)	14.147 (2/3)	19.197 (2/3)	24.247 (2/3)	0.003 (3/3)	0.004 (3/3)	0.004 (3/3)
11	1.043 (3/3)	1.046 (3/3)	1.049 (3/3)	15.160 (2/3)	20.210 (2/3)	25.260 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.003 (3/3)	0.004 (3/3)	0.004 (3/3)
12	0.027 (3/3)	0.029 (3/3)	0.031 (3/3)	15.154 (2/3)	20.204 (2/3)	25.254 (2/3)	14.144 (2/3)	19.194 (2/3)	24.244 (2/3)	0.004 (3/3)	0.004 (3/3)	0.004 (3/3)
13	2.044 (3/3)	2.051 (3/3)	2.058 (3/3)	15.154 (2/3)	20.204 (2/3)	25.254 (2/3)	14.146 (2/3)	19.196 (2/3)	24.246 (2/3)	0.004 (3/3)	0.004 (3/3)	0.005 (3/3)
14	0.046 (3/3)	0.055 (3/3)	0.065 (3/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.004 (3/3)	0.004 (3/3)	0.004 (3/3)
15	0.027 (3/3)	0.029 (3/3)	0.031 (3/3)	15.158 (2/3)	20.209 (2/3)	25.259 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.004 (3/3)	0.004 (3/3)	0.004 (3/3)
16	0.028 (3/3)	0.030 (3/3)	0.032 (3/3)	16.168 (2/3)	21.219 (2/3)	26.270 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.004 (3/3)	0.005 (3/3)	0.005 (3/3)
17	0.040 (3/3)	0.045 (3/3)	0.049 (3/3)	15.153 (2/3)	20.203 (2/3)	25.254 (2/3)	14.144 (2/3)	19.194 (2/3)	24.244 (2/3)	0.003 (3/3)	0.003 (3/3)	0.003 (3/3)
18	0.025 (3/3)	0.028 (3/3)	0.030 (3/3)	15.154 (2/3)	20.204 (2/3)	25.254 (2/3)	14.146 (2/3)	19.197 (2/3)	24.247 (2/3)	0.005 (3/3)	0.005 (3/3)	0.006 (3/3)
19	1.026 (3/3)	1.030 (3/3)	1.039 (3/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	14.144 (2/3)	19.194 (2/3)	24.244 (2/3)	0.005 (3/3)	0.006 (3/3)	0.007 (3/3)
20	0.026 (3/3)	0.027 (3/3)	0.028 (3/3)	16.162 (2/3)	21.213 (2/3)	26.264 (2/3)	14.143 (2/3)	19.193 (2/3)	24.243 (2/3)	0.004 (3/3)	0.004 (3/3)	0.004 (3/3)
Successo (3/3)	100%	100%	100%	0%	0%	0%	0%	0%	0%	100%	100%	100%

Tabella 4.18: *Frequenza di successo per la CTRNN addestrata in modalità* batch *con tre labirinti di grandezza diversa, ovvero per* $\lambda = 2$ *,*$\lambda = 3$ *e* $\lambda = 4$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). Con questo tipo di addestramento l'ANP riesce ad apprendere 5 su 8 comportamenti nel* best case *(70% dei casi) e almeno 4 su 8 nel* worst case *(il restante 30%).*

Programmi appresi	1/8	2/8	3/8	4/8	5/8	6/8	7/8	8/8
Frequenza di successo	100%	100%	100%	100%	70%	0%	0%	0%

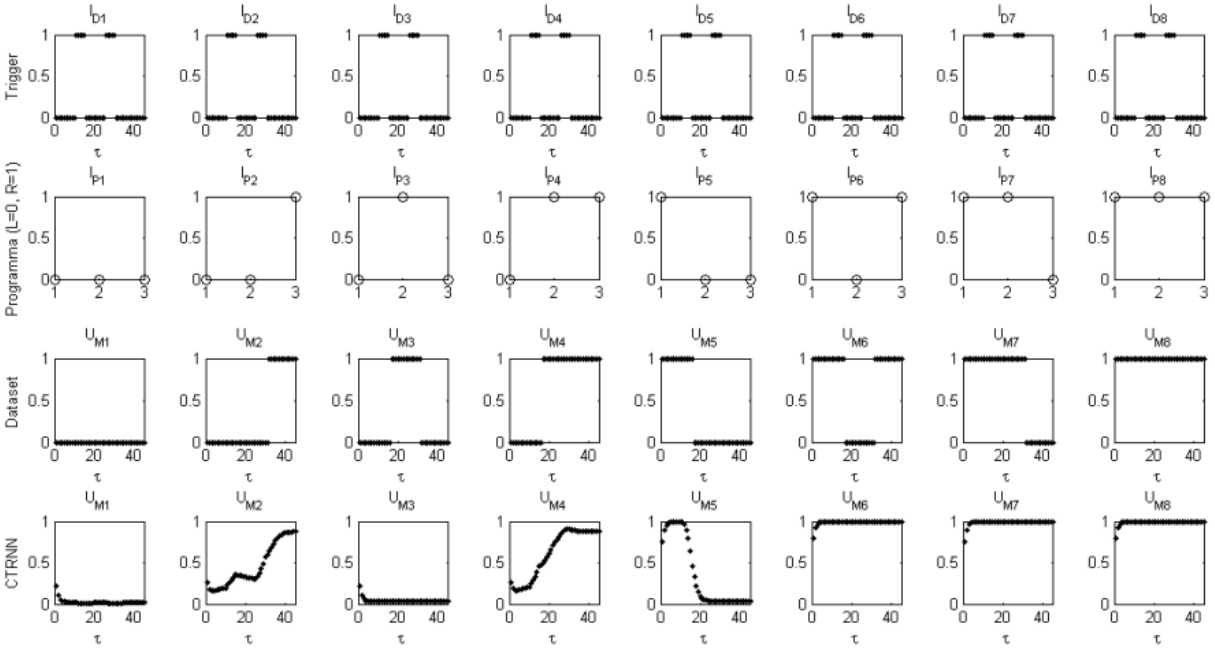


Figura 4.3.4: *Risposta della rete CTRNN numero 5 (*best case*), addestrata in modalità* batch*, con il* dataset *da tre labirinti, al labirinto con* $\lambda = 2$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). I programmi corretti sono* P_1*,* P_2*,* P_4*,* P_5 *e* P_8*, per i quali le uscite* U_M *del dataset e della CTRNN hanno andamenti - al di sotto e al di sopra della soglia a 0,5 - coincidenti. La lettura di* U_M *come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5* τ *dall'istante iniziale e dopo ogni discesa del segnale di* trigger*.*

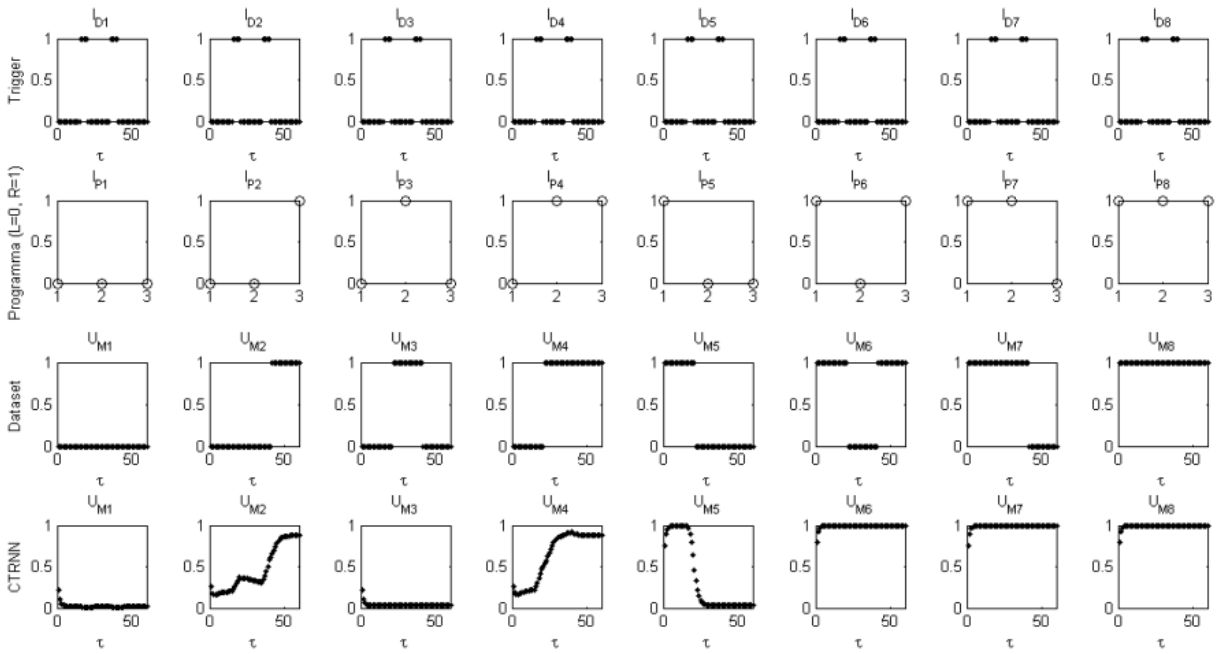


Figura 4.3.5: *Risposta della rete CTRNN numero 5 (*best case*), addestrata in modalità* batch*, con il* dataset *da tre labirinti, al labirinto con* $\lambda = 3$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). I programmi corretti sono* P_1*,* P_2*,* P_4*,* P_5 *e* P_8*, per i quali le uscite* U_M *del dataset e della CTRNN hanno andamenti - al di sotto e al di sopra della soglia a 0,5 - coincidenti. La lettura di* U_M *come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5* τ *dall'istante iniziale e dopo ogni discesa del segnale di* trigger*.*

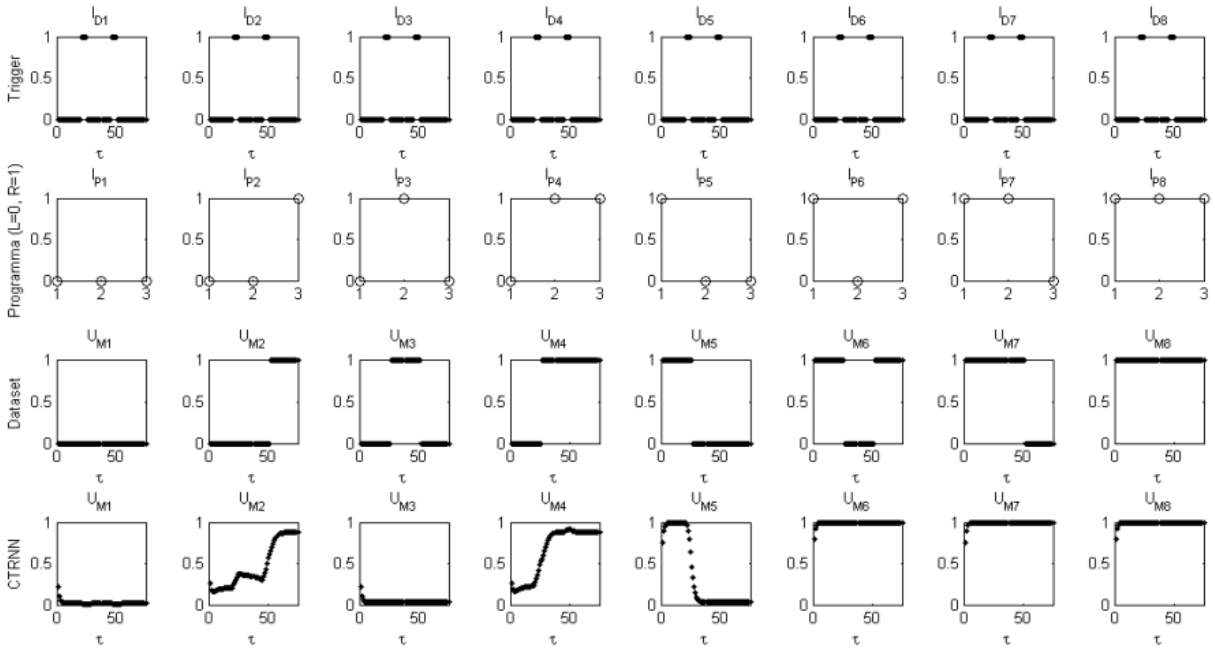


Figura 4.3.6: *Risposta della rete CTRNN numero 5 (*best case*), addestrata in modalità* batch*, con il* dataset *da tre labirinti, al labirinto con* $\lambda = 4$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). I programmi corretti sono* P_1*,* P_2*,* P_4*,* P_5 *e* P_8*, per i quali le uscite* U_M *del dataset e della CTRNN hanno andamenti - al di sotto e al di sopra della soglia a 0,5 - coincidenti. La lettura di* U_M *come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5* τ *dall'istante iniziale e dopo ogni discesa del segnale di* trigger*.*

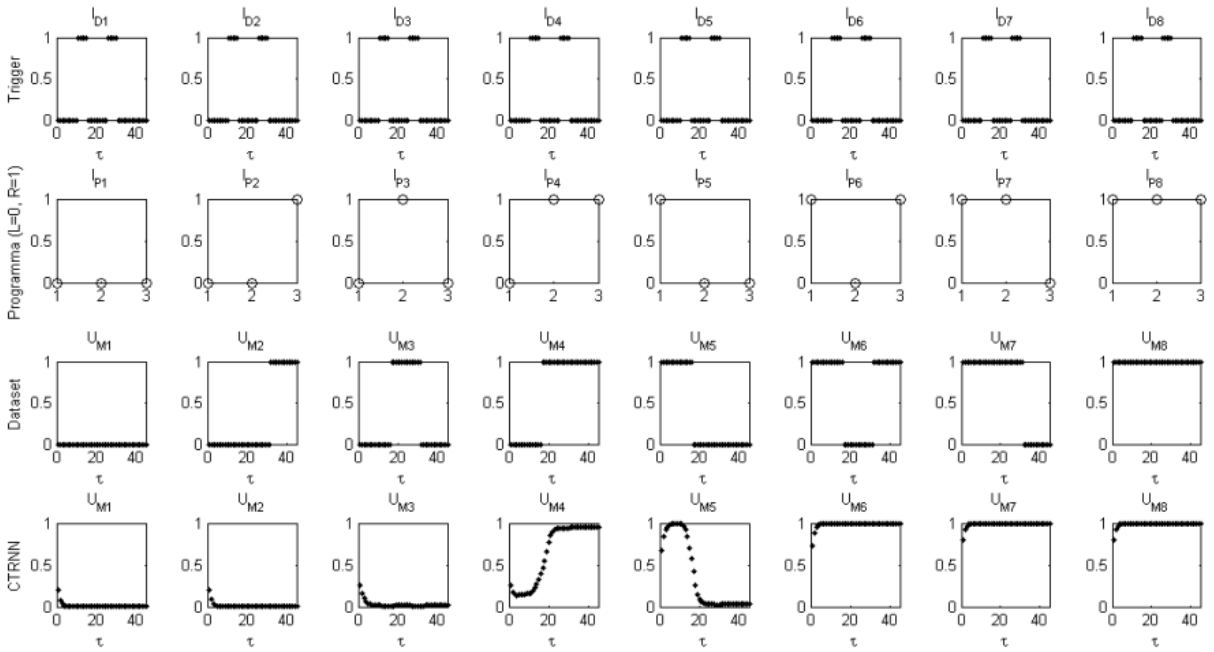


Figura 4.3.7: *Risposta della rete CTRNN numero 3 (*worst case*), addestrata in modalità* batch*, con il* dataset *da tre labirinti, al labirinto con* $\lambda = 2$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). I programmi corretti sono* P_1*,* P_4*,* P_5 *e* P_8*, per i quali le uscite* U_M *del dataset e della CTRNN hanno andamenti - al di sotto e al di sopra della soglia a 0,5 - coincidenti. La lettura di* U_M *come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5* τ *dall'istante iniziale e dopo ogni discesa del segnale di* trigger*.*

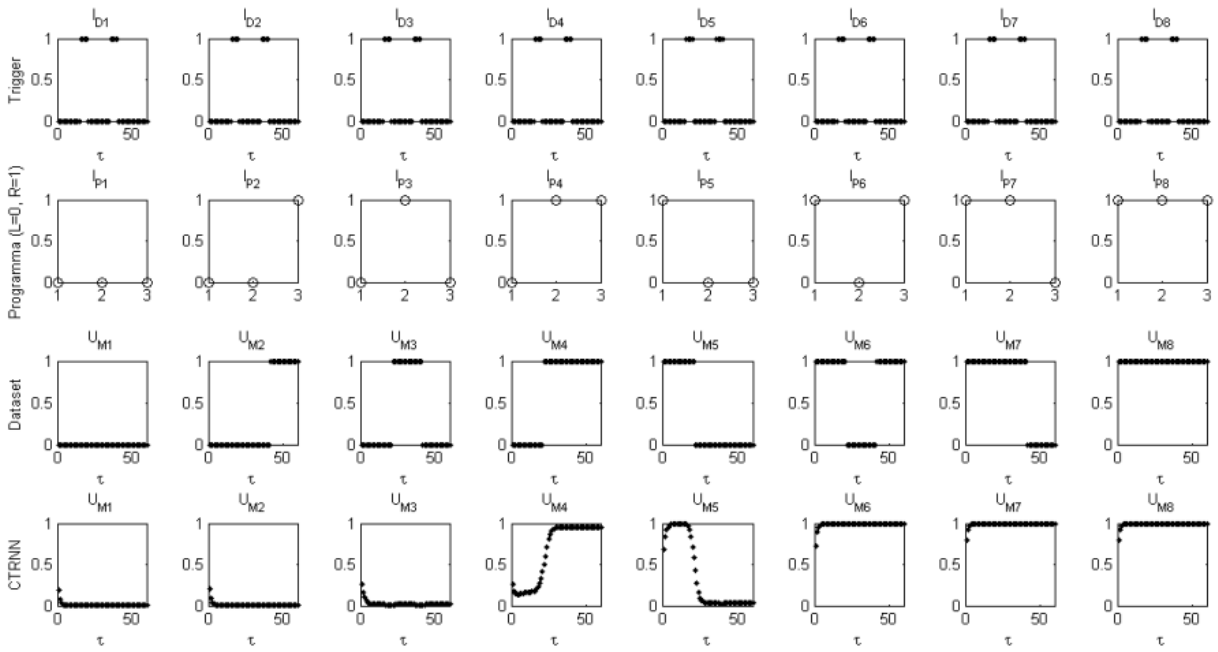


Figura 4.3.8: *Risposta della rete CTRNN numero 3 (*worst case*), addestrata in modalità* batch*, con il* dataset *da tre labirinti, al labirinto con* $\lambda = 3$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). I programmi corretti sono* P_1*,* P_4*,* P_5 *e* P_8*, per i quali le uscite* U_M *del dataset e della CTRNN hanno andamenti - al di sotto e al di sopra della soglia a 0,5 - coincidenti. La lettura di* U_M *come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5* τ *dall'istante iniziale e dopo ogni discesa del segnale di* trigger*.*

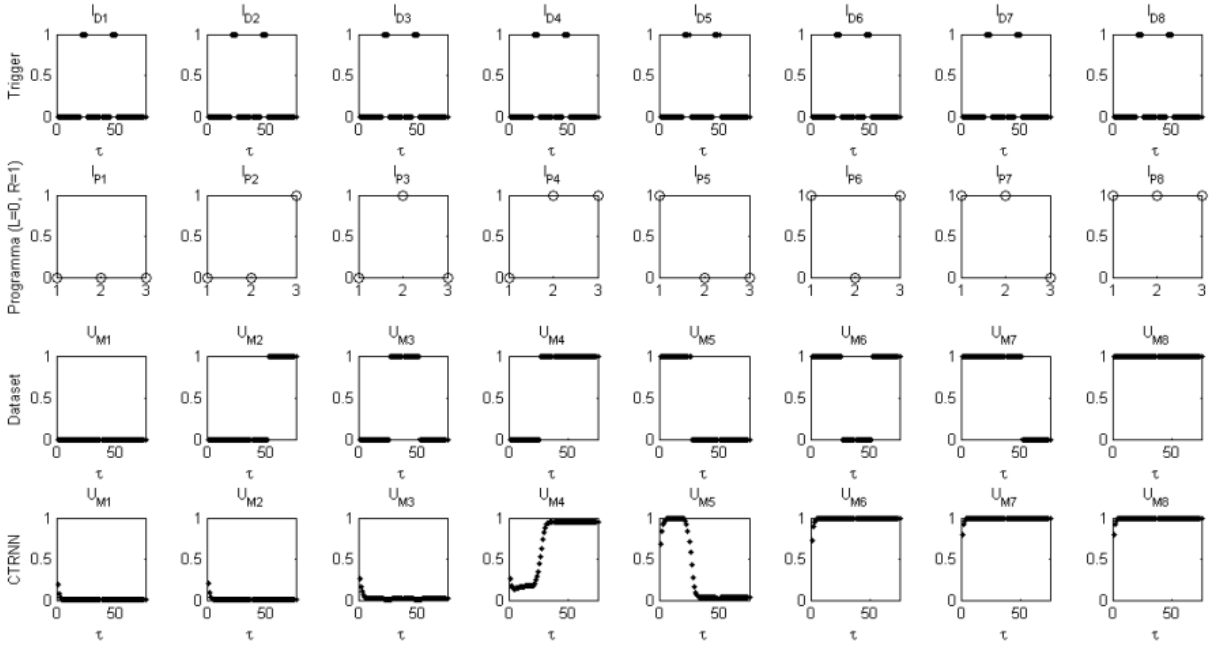


Figura 4.3.9: *Risposta della rete CTRNN numero 3 (*worst case*), addestrata in modalità* batch*, con il* dataset *da tre labirinti, al labirinto con* $\lambda = 4$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). I programmi corretti sono* P_1*,* P_4*,* P_5 *e* P_8*, per i quali le uscite* U_M *del dataset e della CTRNN hanno andamenti - al di sotto e al di sopra della soglia a 0,5 - coincidenti. La lettura di* U_M *come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5* τ *dall'istante iniziale e dopo ogni discesa del segnale di* trigger*.*

4.3.3 Architettura Programmabile

L'Architettura Programmabile (fig. 3.3.1) è stata addestrata per un campione totale di venti PNN, e relative FNN, anche su tre labirinti di grandezza diversa, ovvero per $\lambda = 2$,$\lambda = 3$ e $\lambda = 4$ (parametro adimensionale proporzionale alle dimensioni del labirinto).

La PNN, essendo una CTRNN dotata di programmabilità, così come spiegato nel capitolo 2, ha prestazioni confrontabili con una rete CTRNN full connected che costituisce l'ANP scelta per il raffronto. Come si può vedere dalle tabelle 4.19, 4.20 e 4.21, la rete PNN ha prestazioni di gran lunga superiori rispetto all'ANP e, infatti, riesce ad apprendere con successo otto programmi su otto, nel 25% dei casi, sei programmi su otto nel 64% dei casi e, sempre e comunque, almeno sei programmi, generalmente appresi solo nel 11% dei casi (tab. 4.9). I programmi che presentano più difficoltà ad essere appresi risultano essere P_3 e P_7, le cui frequenze di successo sono, rispettivamente, del 35% e del 80%.

Tabella 4.21: *Statistica dei risultati per la PNN addestrata con tre labirinti di grandezza diversa, ovvero per $\lambda = 2$,$\lambda = 3$ e $\lambda = 4$ (parametro adimensionale proporzionale alle dimensioni del labirinto). L'apprendimento mostra una buona capacità dell'AP ad apprendere comportamenti multipli e più che sufficiente ad apprenderne anche 8 su 8 proposti. La deviazione standard è leggermente alta per i programmi che mostrano una certa difficoltà ad essere appresi (P_2, P_3 e P_7), lo si può notare dal raffronto diretto con la loro frequenza di successo. P_2 e P_3 sono i programmi che hanno anche una media d'errore leggermente alta, ovvero tale da sbagliare al massimo solo una svolta delle tre previste per raggiungere la terminazione corretta del labirinto, ma comunque rendendo errato l'intero programma, concordemente con la loro frequenza di successo (3/3 svolte corrette) rispettivamente al 35% e 80%.*

	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8
Errore medio	0.014	3.493	39.818	0.195	0.178	0.203	11.945	0.014
Deviazione standard	0.000	12.759	29.087	0.057	0.097	0.076	23.423	0.000
Errore massimo	0.014	57.584	60.614	0.329	0.351	0.412	57.584	0.014
Errore minimo	0.014	0.177	0.277	0.093	0.055	0.125	0.102	0.014
Frequenza di successo	100%	95%	35%	100%	100%	100%	80%	100%

Tabella 4.22: *Frequenza di successo per la PNN addestrata con tre labirinti di grandezza diversa, ovvero per $\lambda = 2$,$\lambda = 3$ e $\lambda = 4$ (parametro adimensionale proporzionale alle dimensioni del labirinto). Con questo tipo di addestramento l'AP riesce ad apprendere 8 comportamenti su 8 nel* best case *(25% dei casi), almeno 7 su 8 nel 64% dei casi, fino a un minimo di 6 su 8 nel* worst case*, ovvero nel 11% dei casi.*

Programmi appresi	1/8	2/8	3/8	4/8	5/8	6/8	7/8	8/8
Frequenza di successo	100%	100%	100%	100%	100%	100%	85%	25%

Tabella 4.19: *Errori dei programmi (tabella da P_1 a P_4) della PNN addestrata con tre labirinti di grandezza diversa, ovvero per $\lambda = 2$,$\lambda = 3$ e $\lambda = 4$ (parametro adimensionale proporzionale alle dimensioni del labirinto). Grazie all'uso di un'AP, tutti i programmi possono essere appresi correttamente. Tra le 20 esecuzioni si possono isolare un* best case *(figg. 4.3.10, 4.3.11 e 4.3.12), con 8 su 8 programmi appresi, ed un* worst case*, con 6 su 8 programmi appresi (figg. 4.3.13, 4.3.14 e 4.3.15).*

PNN	**Errore del programma (Numero di svolte corrette)**											
	$\boldsymbol{P_1}$			$\boldsymbol{P_2}$			$\boldsymbol{P_3}$			$\boldsymbol{P_4}$		
	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$
1	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	0.090 (3/3)	0.115 (3/3)	0.139 (3/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	0.043 (3/3)	0.047 (3/3)	0.051 (3/3)
2	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	0.092 (3/3)	0.118 (3/3)	0.146 (3/3)	0.093 (3/3)	0.113 (3/3)	0.133 (3/3)	0.046 (3/3)	0.051 (3/3)	0.056 (3/3)
3 (*best*)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	0.048 (3/3)	0.059 (3/3)	0.070 (3/3)	0.077 (3/3)	0.092 (3/3)	0.108 (3/3)	0.058 (3/3)	0.066 (3/3)	0.074 (3/3)
4	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	0.124 (3/3)	0.171 (3/3)	0.218 (3/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	0.058 (3/3)	0.065 (3/3)	0.072 (3/3)
5	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	1.033 (3/3)	1.034 (3/3)	1.035 (3/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	0.056 (3/3)	0.065 (3/3)	0.075 (3/3)
6	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	0.124 (3/3)	0.169 (3/3)	0.214 (3/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	0.043 (3/3)	0.047 (3/3)	0.051 (3/3)
7	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	0.073 (3/3)	0.093 (3/3)	0.112 (3/3)	1.035 (3/3)	1.036 (3/3)	1.037 (3/3)	0.055 (3/3)	0.063 (3/3)	0.071 (3/3)
8	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	0.115 (3/3)	0.154 (3/3)	0.194 (3/3)	0.140 (3/3)	0.177 (3/3)	0.215 (3/3)	0.062 (3/3)	0.077 (3/3)	0.091 (3/3)
9	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	0.081 (3/3)	0.110 (3/3)	0.144 (3/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	0.076 (3/3)	0.097 (3/3)	0.117 (3/3)
10	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	0.077 (3/3)	0.099 (3/3)	0.117 (3/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	0.056 (3/3)	0.065 (3/3)	0.074 (3/3)
11	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	1.023 (3/3)	1.023 (3/3)	1.024 (3/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	0.046 (3/3)	0.049 (3/3)	0.052 (3/3)
12	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	0.050 (3/3)	0.061 (3/3)	0.071 (3/3)	1.047 (3/3)	1.049 (3/3)	1.051 (3/3)	0.054 (3/3)	0.060 (3/3)	0.066 (3/3)
13	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	0.104 (3/3)	0.135 (3/3)	0.164 (3/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	0.078 (3/3)	0.097 (3/3)	0.116 (3/3)
14 (*worst*)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	0.096 (3/3)	0.124 (3/3)	0.152 (3/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	0.056 (3/3)	0.064 (3/3)	0.073 (3/3)
15	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	0.098 (3/3)	0.132 (3/3)	0.166 (3/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	0.054 (3/3)	0.064 (3/3)	0.073 (3/3)
16	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	14.145 (2/3)	19.195 (2/3)	24.245 (2/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	0.041 (3/3)	0.048 (3/3)	0.054 (3/3)
17	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	0.116 (3/3)	0.157 (3/3)	0.197 (3/3)	0.136 (3/3)	0.170 (3/3)	0.205 (3/3)	0.060 (3/3)	0.068 (3/3)	0.077 (3/3)
18	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	0.097 (3/3)	0.130 (3/3)	0.162 (3/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	0.030 (3/3)	0.031 (3/3)	0.032 (3/3)
19	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	0.074 (3/3)	0.093 (3/3)	0.112 (3/3)	15.155 (2/3)	20.205 (2/3)	25.255 (2/3)	0.059 (3/3)	0.066 (3/3)	0.074 (3/3)
20	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)	0.092 (3/3)	0.117 (3/3)	0.142 (3/3)	0.118 (3/3)	0.150 (3/3)	0.182 (3/3)	0.091 (3/3)	0.110 (3/3)	0.128 (3/3)
Successo (3/3)	100%	100%	100%	95%	95%	95%	35%	35%	35%	100%	100%	100%

Tabella 4.20: *Errori dei programmi (tabella da P_5 a P_8) della PNN addestrata con tre labirinti di grandezza diversa, ovvero per $\lambda = 2$,$\lambda = 3$ e $\lambda = 4$ (parametro adimensionale proporzionale alle dimensioni del labirinto). Grazie all'uso di un'AP, tutti i programmi possono essere appresi correttamente. Tra le 20 esecuzioni si possono isolare un* best case *(figg. 4.3.10, 4.3.11 e 4.3.12), con 8 su 8 programmi appresi, ed un* worst case*, con 6 su 8 programmi appresi (figg. 4.3.13, 4.3.14 e 4.3.15).*

PNN	**Errore del programma (Numero di svolte corrette)**											
	$\boldsymbol{P_5}$			$\boldsymbol{P_6}$			$\boldsymbol{P_7}$			$\boldsymbol{P_8}$		
	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$	$\lambda = 2$	$\lambda = 3$	$\lambda = 4$
1	0.028 (3/3)	0.029 (3/3)	0.031 (3/3)	0.054 (3/3)	0.057 (3/3)	0.060 (3/3)	0.061 (3/3)	0.082 (3/3)	0.102 (3/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
2	0.021 (3/3)	0.022 (3/3)	0.023 (3/3)	0.065 (3/3)	0.070 (3/3)	0.074 (3/3)	14.145 (2/3)	19.195 (2/3)	24.245 (2/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
3 (*best*)	0.054 (3/3)	0.069 (3/3)	0.085 (3/3)	0.059 (3/3)	0.065 (3/3)	0.070 (3/3)	0.145 (3/3)	0.193 (3/3)	0.239 (3/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
4	0.053 (3/3)	0.063 (3/3)	0.073 (3/3)	0.053 (3/3)	0.061 (3/3)	0.068 (3/3)	1.118 (3/3)	1.166 (3/3)	1.216 (3/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
5	0.018 (3/3)	0.018 (3/3)	0.019 (3/3)	0.051 (3/3)	0.057 (3/3)	0.064 (3/3)	0.058 (3/3)	0.065 (3/3)	0.072 (3/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
6	0.044 (3/3)	0.050 (3/3)	0.056 (3/3)	0.070 (3/3)	0.083 (3/3)	0.098 (3/3)	0.119 (3/3)	0.155 (3/3)	0.191 (3/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
7	0.058 (3/3)	0.075 (3/3)	0.092 (3/3)	0.041 (3/3)	0.044 (3/3)	0.047 (3/3)	0.038 (3/3)	0.042 (3/3)	0.046 (3/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
8	0.031 (3/3)	0.034 (3/3)	0.038 (3/3)	0.083 (3/3)	0.087 (3/3)	0.093 (3/3)	14.145 (2/3)	19.195 (2/3)	24.245 (2/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
9	0.027 (3/3)	0.028 (3/3)	0.029 (3/3)	0.044 (3/3)	0.046 (3/3)	0.049 (3/3)	0.137 (3/3)	0.177 (3/3)	0.212 (3/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
10	0.081 (3/3)	0.097 (3/3)	0.112 (3/3)	0.068 (3/3)	0.080 (3/3)	0.095 (3/3)	0.053 (3/3)	0.056 (3/3)	0.059 (3/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
11	0.031 (3/3)	0.036 (3/3)	0.041 (3/3)	0.045 (3/3)	0.055 (3/3)	0.059 (3/3)	0.102 (3/3)	0.143 (3/3)	0.186 (3/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
12	0.091 (3/3)	0.109 (3/3)	0.126 (3/3)	0.048 (3/3)	0.053 (3/3)	0.058 (3/3)	0.146 (3/3)	0.197 (3/3)	0.247 (3/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
13	0.074 (3/3)	0.098 (3/3)	0.123 (3/3)	0.091 (3/3)	0.102 (3/3)	0.113 (3/3)	0.025 (3/3)	0.034 (3/3)	0.043 (3/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
14 (*worst*)	0.098 (3/3)	0.117 (3/3)	0.135 (3/3)	0.044 (3/3)	0.046 (3/3)	0.049 (3/3)	14.145 (2/3)	19.195 (2/3)	24.245 (2/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
15	0.082 (3/3)	0.101 (3/3)	0.121 (3/3)	0.041 (3/3)	0.045 (3/3)	0.049 (3/3)	0.133 (3/3)	0.177 (3/3)	0.209 (3/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
16	0.067 (3/3)	0.077 (3/3)	0.088 (3/3)	0.086 (3/3)	0.108 (3/3)	0.130 (3/3)	0.076 (3/3)	0.104 (3/3)	0.133 (3/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
17	0.046 (3/3)	0.048 (3/3)	0.051 (3/3)	0.048 (3/3)	0.053 (3/3)	0.059 (3/3)	0.082 (3/3)	0.106 (3/3)	0.132 (3/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
18	0.018 (3/3)	0.019 (3/3)	0.019 (3/3)	0.055 (3/3)	0.061 (3/3)	0.066 (3/3)	0.066 (3/3)	0.082 (3/3)	0.099 (3/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
19	0.052 (3/3)	0.060 (3/3)	0.069 (3/3)	0.037 (3/3)	0.041 (3/3)	0.047 (3/3)	14.145 (2/3)	19.195 (2/3)	24.245 (2/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
20	0.034 (3/3)	0.035 (3/3)	0.036 (3/3)	0.108 (3/3)	0.138 (3/3)	0.166 (3/3)	0.075 (3/3)	0.079 (3/3)	0.087 (3/3)	0.005 (3/3)	0.005 (3/3)	0.005 (3/3)
Successo (3/3)	100%	100%	100%	100%	100%	100%	80%	80%	80%	100%	100%	100%

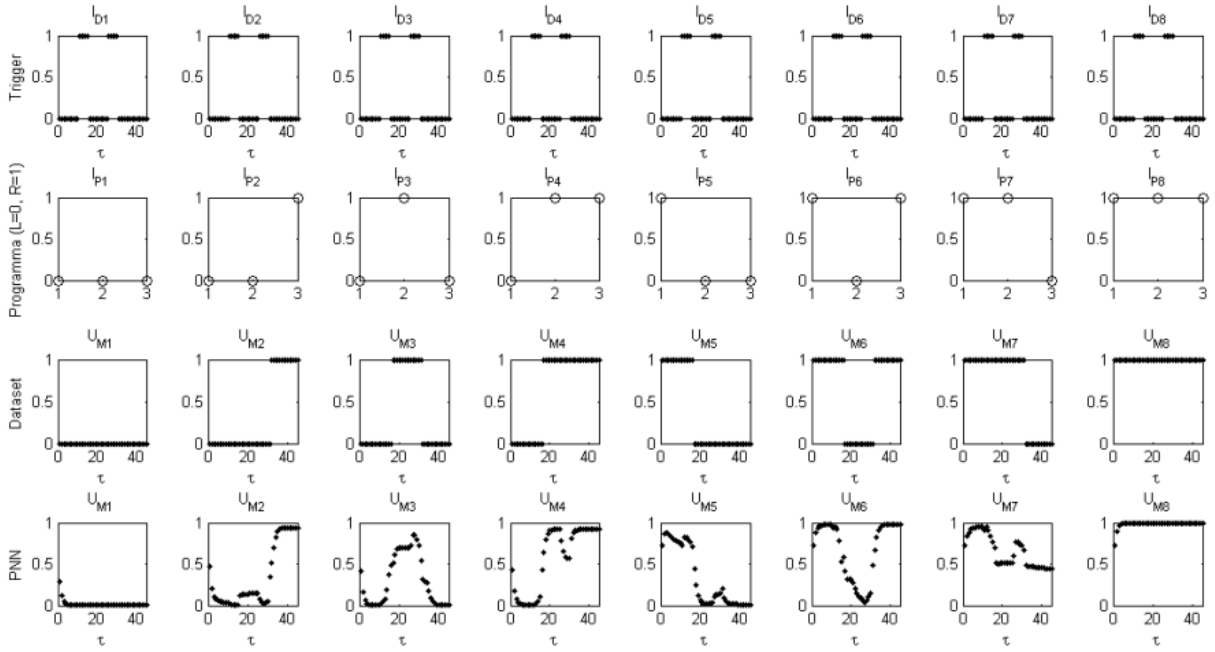


Figura 4.3.10: *Risposta della rete PNN numero 3 (*best case*), addestrata con il* dataset *da tre labirinti, al labirinto con* $\lambda = 2$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). I programmi sono tutti e otto corretti, infatti, tutte le uscite* U_M *del dataset e della PNN hanno andamenti - al di sotto e al di sopra della soglia a 0,5 - coincidenti. La lettura di* U_M *come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5* τ *dall'istante iniziale e dopo ogni discesa del segnale di* trigger.

Tabella 4.23: *Passi d'apprendimento della PNN con tre labirinti di grandezza diversa, ovvero per* $\lambda = 2$ *,*$\lambda = 3$ *e* $\lambda = 4$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). Il limite massimo di passi totali per l'apprendimento della PNN è impostato a* $S^{PNN} = S^{online} = N_P \cdot S^{batch} = 8 \cdot 2500 = 20000$ *passi totali, che corrispondono a 2500 passi massimi per ciascun programma. In realtà, come si vede dalla tabella, il massimo di passi raggiunto è di 7700 che è di gran lunga inferiore rispetto ai 20000 passi massimi previsti per l'apprendimento dell'ANP in modalità* online*. L'apprendimento a soglia ha ridotto da un minimo di* $20000 - 7700 = 12300$ *passi, ad un massimo di* $20000 - 2800 = 17200$ *passi, l'apprendimento* standard *per una PNN. La soglia massima di 2500 passi d'apprendimento per programma impostata era, questa volta, pari a quella impostata per l'ANP. L'algoritmo di apprendimento a soglia è dunque più efficiente sia di quello* online *che di quello* batch *perché ne acquista, di entrambi, i vantaggi: meno passi totali (modalità* batch*) e la possibilità, trattandosi di PNN, di apprendere i programmi singolarmente (modalità* online*), con maggior efficacia (minimizzazione dell'errore) sui singoli programmi.*

PNN	**Passi d'apprendimento per programma**								**Passi totali**
	P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	
1	50	400	2500	100	100	1100	600	50	4900
2	50	250	550	50	100	1500	2500	50	5050
3	50	250	1050	150	50	850	900	50	3350
4	50	450	2500	150	50	750	2500	50	6500
5	50	2500	2500	100	100	750	1550	50	7600
6	50	200	2500	150	50	550	150	50	3700
7	50	350	2500	200	50	650	1700	50	5550
8	50	200	900	150	50	1250	2500	50	5150
9	50	350	2500	150	50	1200	300	50	4650
10	50	200	2500	400	50	1000	1350	50	5600
11	50	2500	2500	150	50	1600	800	50	7700
12	50	350	2500	250	50	450	1250	50	4950
13	50	1450	2500	100	50	600	600	50	5400
14	50	250	2500	100	50	1600	2500	50	7100
15	50	300	2500	150	50	950	250	50	4300
16	50	2500	2500	150	50	950	550	50	6800
17	50	150	700	100	50	1100	600	50	2800
18	50	550	2500	150	100	450	400	50	4250
19	50	350	2500	100	100	1000	2500	50	6650
20	50	300	700	100	100	750	850	50	2900
Media	50	693	2070	148	65	953	1218	50	5245
Dev. standard	0	825	769	73	24	352	864	0	1472
Massimo	50	2500	2500	400	100	1600	2500	50	7700
Minimo	50	150	550	50	50	450	150	50	2800

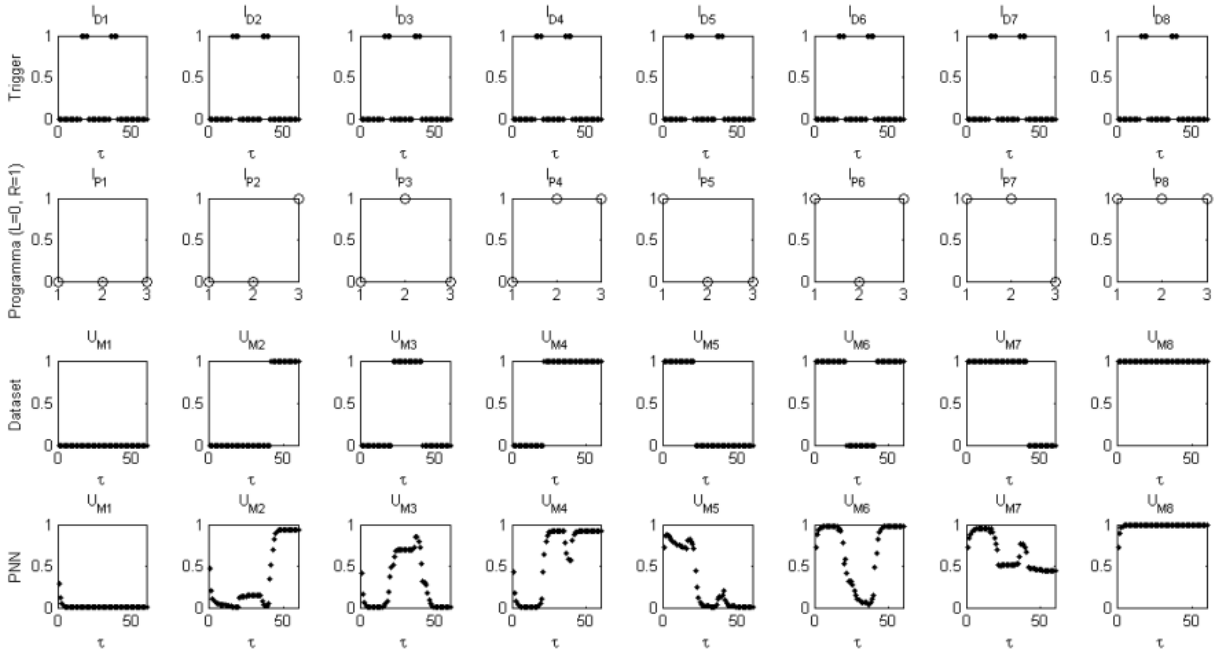


Figura 4.3.11: *Risposta della rete PNN numero 3 (*best case*), addestrata con il* dataset *da tre labirinti, al labirinto con* $\lambda = 3$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). I programmi sono tutti e otto corretti, infatti, tutte le uscite* U_M *del dataset e della PNN hanno andamenti - al di sotto e al di sopra della soglia a 0,5 - coincidenti. La lettura di* U_M *come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5* τ *dall'istante iniziale e dopo ogni discesa del segnale di* trigger.

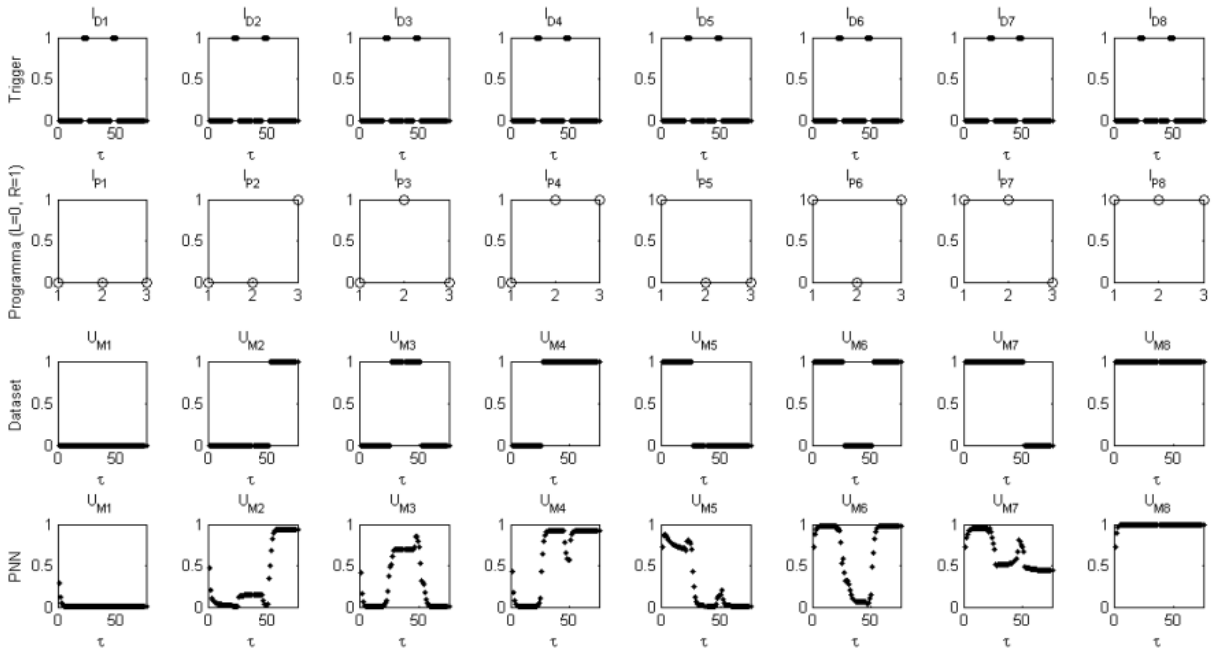


Figura 4.3.12: *Risposta della rete PNN numero 3 (*best case*), addestrata con il* dataset *da tre labirinti, al labirinto con* $\lambda = 4$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). I programmi sono tutti e otto corretti, infatti, tutte le uscite* U_M *del dataset e della PNN hanno andamenti - al di sotto e al di sopra della soglia a 0,5 - coincidenti. La lettura di* U_M *come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5* τ *dall'istante iniziale e dopo ogni discesa del segnale di* trigger.

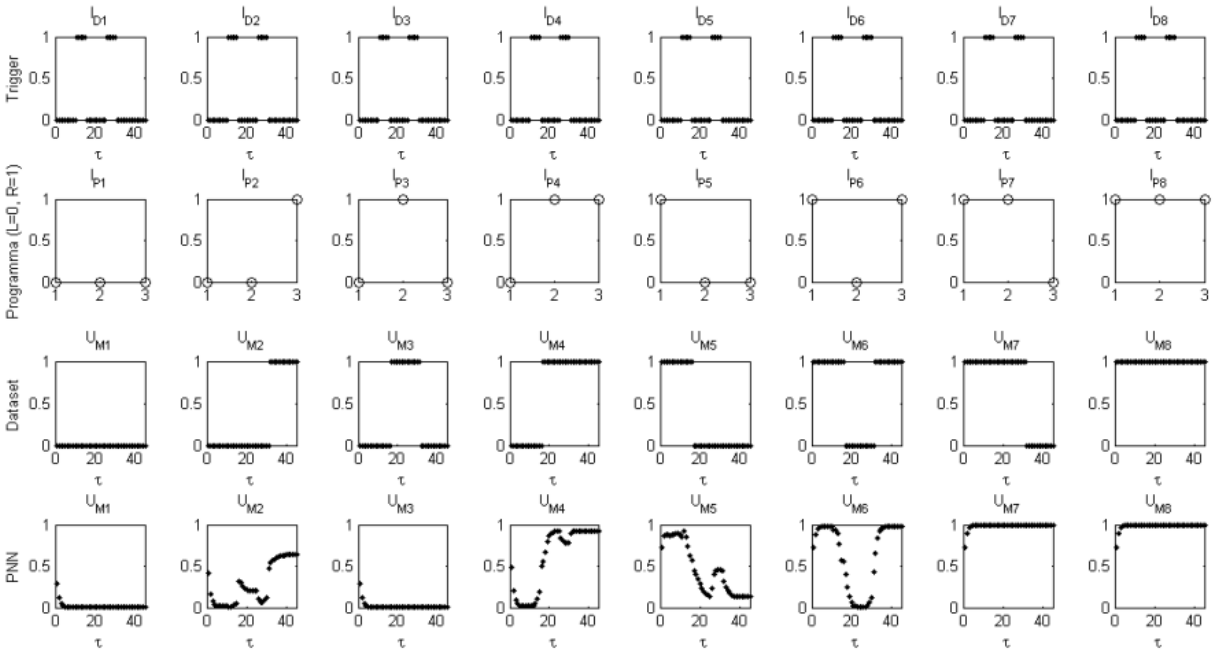


Figura 4.3.13: *Risposta della rete PNN numero 14 (*worst case*), addestrata con il* dataset *da tre labirinti, al labirinto con $\lambda = 2$ (parametro adimensionale proporzionale alle dimensioni del labirinto). I programmi corretti sono P_1, P_2, P_4, P_5, P_6 e P_8, per i quali le uscite U_M del dataset e della CTRNN hanno andamenti - al di sotto e al di sopra della soglia a 0,5 - coincidenti. La lettura di U_M come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5 τ dall'istante iniziale e dopo ogni discesa del segnale di* trigger.

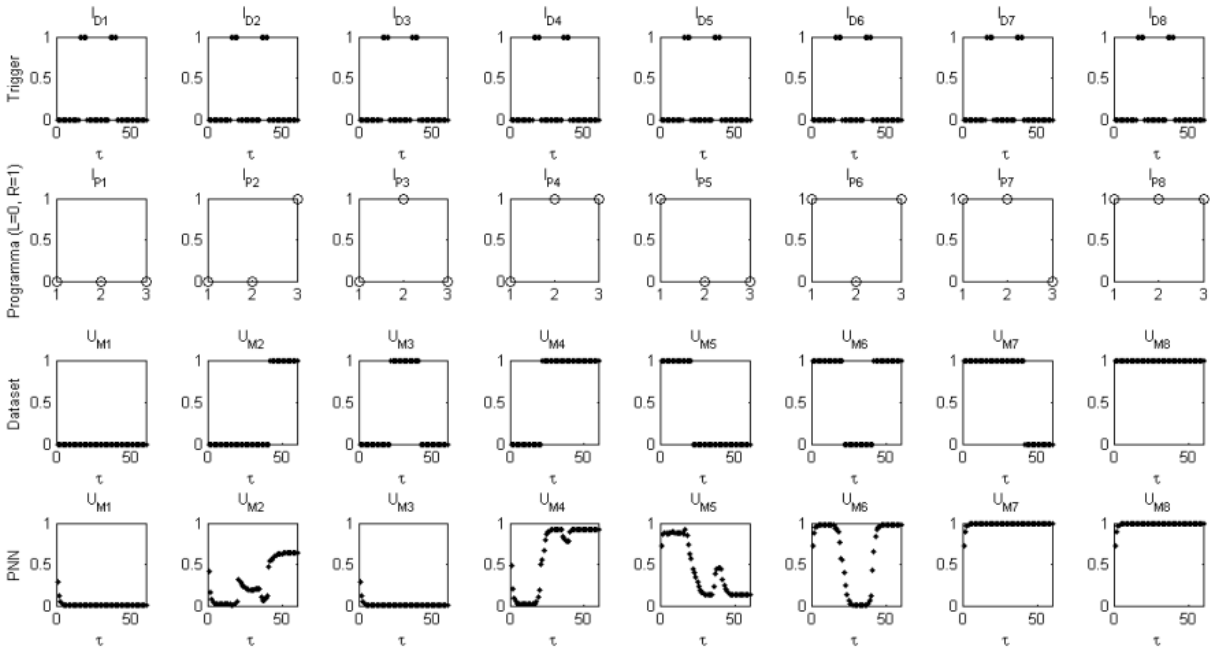


Figura 4.3.14: *Risposta della rete PNN numero 14 (*worst case*), addestrata con il* dataset *da tre labirinti, al labirinto con* $\lambda = 3$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). I programmi corretti sono* P_1*,* P_2*,* P_4*,* P_5*,* P_6 *e* P_8*, per i quali le uscite* U_M *del dataset e della CTRNN hanno andamenti - al di sotto e al di sopra della soglia a 0,5 - coincidenti. La lettura di* U_M *come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5* τ *dall'istante iniziale e dopo ogni discesa del segnale di* trigger.

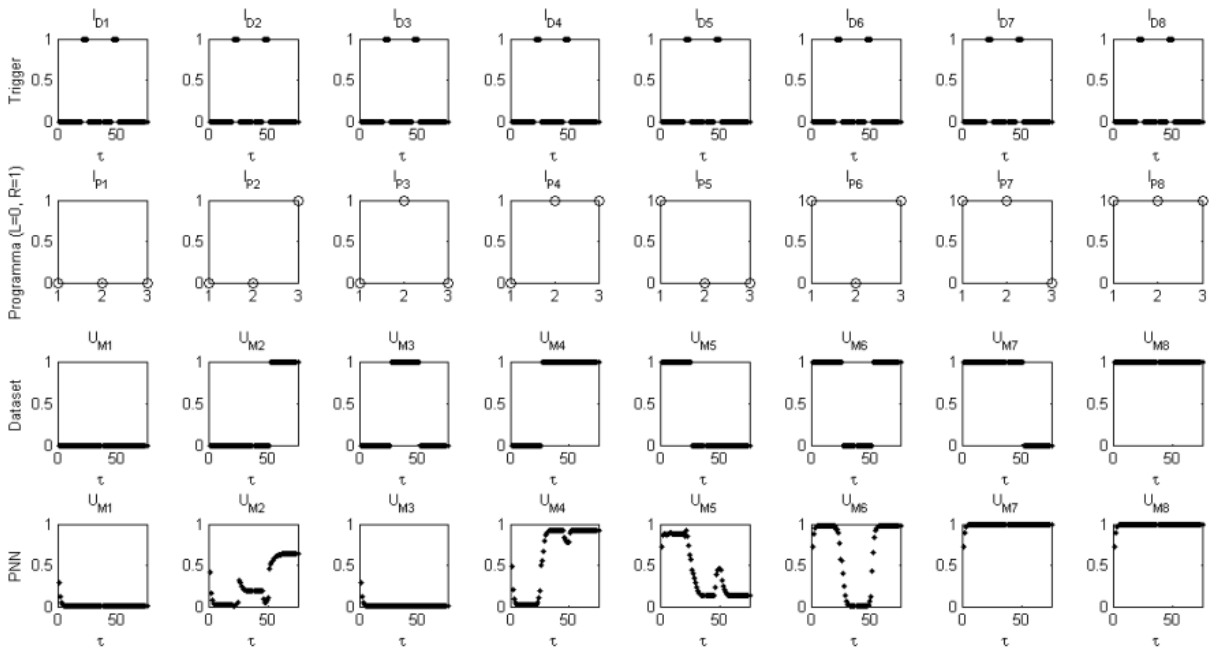


Figura 4.3.15: *Risposta della rete PNN numero 14 (*worst case*), addestrata con il* dataset *da tre labirinti, al labirinto con* $\lambda = 4$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). I programmi corretti sono* P_1*,* P_2*,* P_4*,* P_5*,* P_6 *e* P_8*, per i quali le uscite* U_M *del dataset e della CTRNN hanno andamenti - al di sotto e al di sopra della soglia a 0,5 - coincidenti. La lettura di* U_M *come segnale on-off, nel nostro caso, viene effettuata con un offset pari a 5* τ *dall'istante iniziale e dopo ogni discesa del segnale di* trigger.

4.3.4 Confronto dei risultati

Dai risultati dell'apprendimento con il *dataset* con tre labirinti di grandezza diversa, ovvero per $\lambda = 2$,$\lambda = 3$ e $\lambda = 4$ (parametro adimensionale proporzionale alle dimensioni del labirinto), risulta che i programmi della PNN vengono appresi tutti e otto nel 25% dei casi (tab. 4.22); nel 64% dei casi, vengono appresi comunque 7 programmi su 8 e, nel 11%, solo 6 su 8, il tutto mediamente in soli 656 passi d'apprendimento (tab. 4.23). Un risultato notevole rispetto a quello della CTRNN addestrata in modalità *batch* che ne apprende massimo 5, nell'70% dei casi, e almeno 4 nel restante 30% (tab. 4.18), con 2500 passi fissi d'apprendimento, oppure rispetto a quello della CTRNN addestrata in modalità *online* che riesce ad apprendere uno ed un solo programma nel 100% dei casi (tab. 4.14), con 20000 passi fissi d'apprendimento.

In sintesi, anche lo scenario con tre labirinti (§ 3.1.4.2) favorisce di gran lunga l'Architettura Programmabile composta da PNN e FNN, piuttosto che l'Architettura Non Programmabile composta da una CTRNN *full connected,* con due comportamenti in più appresi.

L'Architettura Programmabile si dimostra, anche in questo caso, l'unica in grado di apprendere e, stavolta di generalizzare pure, tutti e otto i comportamenti proposti.

4.4 Test delle architetture su labirinti di grandezza casuale

Una volta appresi tutti i comportamenti - o programmi - per ciascun tipo di architettura e con ogni modalità di apprendimento proposta, procediamo ad effettuare un test delle architetture di controllo su labirinti di grandezza casuale. Le dimensioni dei labirinti scelte per tali prove sono due e corrispondono ai parametri adimensionali $\lambda = 2.4$ e $\lambda = 3.6$, proporzionali alle dimensioni dei tratti rettilinei dei nuovi labirinti (eq. 3.1.2). In particolare, abbiamo scelto labirinti che nei nuovi dataset hanno segnale di *trigger* alto con intervallo $\triangle\tau_{I_DHI} = 5$ (durata di svolta) e segnale basso con intervallo, rispettivamente, di $\triangle\tau_{I_DLO} = \lambda \cdot \triangle\tau_{I_DHI} = 2.4 \cdot \triangle\tau_{I_DHI} = 12$ e $\triangle\tau_{I_DLO} = \lambda \cdot \triangle\tau_{I_DHI} = 3.6 \cdot \triangle\tau_{I_DHI} = 18$ unità di tempo (tempo di percorrenza a velocità costante del tratto rettilineo di lunghezza δ).

Delle reti neurali addestrate in precedenza (su labirinto base §. 4.2 e su tre labirinti §. 4.3) abbiamo scelto per la fase di test, per ognuna delle architetture presentate e relative modalità d'apprendimento, quelle che risultavano essere il *best case* ed il *worst case.*

I risultati mostrano che le architetture programmabili se addestrate con il solo labirinto base, a differenza delle architetture non programmabili, non sono in grado di generalizzare

i comportamenti per labirinti di diversa grandezza e hanno risposte peggiori man mano che le dimensioni del labirinto in analisi aumentano rispetto a quelle del labirinto base appreso (tab. 4.24).

D'altro canto, entrambe le architetture, se addestrate con più di un labirinto, hanno la capacità di generalizzare i comportamenti appresi per tutti i nuovi labirinti che sono stati scelti in fase di test (tab. 4.25).

Tabella 4.24: *Risposta di ciascun tipo di rete, addestrata con il* dataset *del labirinto base ($\lambda = 2$). Le due architetture testate con labirinti a $\lambda = 2.4$ e $\lambda = 3.6$ (parametro adimensionale proporzionale alle dimensioni del labirinto) non riescono a generalizzare i comportamenti appresi in precedenza su percorsi di dimensioni che si allontanano dalle misure del labirinto base.*

λ	**Rete (Addestr.)**	**B/W Case**	**Errore del programma (Numero di svolte corrette)**								**Successo Programmi**
			P_1	**P_2**	**P_3**	**P_4**	**P_5**	**P_6**	**P_7**	**P_8**	
2.4	CTRNN (*Online*)	*Best*	51.507 (0/3)	35.347 (1/3)	34.337 (1/3)	18.177 (2/3)	33.333 (1/3)	17.173 (2/3)	16.163 (2/3)	0.003 (3/3)	1/8
		Worst	51.507 (0/3)	35.347 (1/3)	34.337 (1/3)	18.177 (2/3)	33.333 (1/3)	17.173 (2/3)	16.163 (2/3)	0.003 (3/3)	1/8
	CTRNN (*Batch*)	*Best*	0.003 (3/3)	16.165 (2/3)	3.055 (3/3)	1.034 (3/3)	1.033 (3/3)	17.177 (2/3)	16.163 (2/3)	0.003 (3/3)	5/8
		Worst	0.003 (3/3)	16.164 (2/3)	17.173 (2/3)	0.048 (3/3)	0.024 (3/3)	17.174 (2/3)	16.164 (2/3)	0.003 (3/3)	4/8
	PNN	*Best*	0.005 (3/3)	7.137 (3/3)	0.065 (3/3)	0.045 (3/3)	0.028 (3/3)	1.064 (3/3)	2.088 (3/3)	0.005 (3/3)	8/8
		Worst	0.005 (3/3)	3.109 (3/3)	17.175 (2/3)	0.085 (3/3)	0.099 (3/3)	0.054 (3/3)	0.098 (3/3)	0.005 (3/3)	7/8
3.6	CTRNN (*Online*)	*Best*	69.687 (0/3)	47.467 (1/3)	46.457 (1/3)	24.237 (2/3)	45.453 (1/3)	23.233 (2/3)	22.223 (2/3)	0.003 (3/3)	1/8
		Worst	69.687 (0/3)	47.467 (1/3)	46.457 (1/3)	24.237 (2/3)	45.453 (1/3)	23.233 (2/3)	22.223 (2/3)	0.003 (3/3)	1/8
	CTRNN (*Batch*)	*Best*	0.003 (3/3)	22.226 (2/3)	3.062 (3/3)	1.037 (3/3)	1.036 (3/3)	23.237 (2/3)	22.223 (2/3)	0.003 (3/3)	5/8
		Worst	0.003 (3/3)	22.224 (2/3)	23.233 (2/3)	1.060 (3/3)	0.027 (3/3)	23.234 (2/3)	22.224 (2/3)	0.004 (3/3)	4/8
	PNN	*Best*	0.005 (3/3)	16.215 (2/3)	0.071 (3/3)	0.048 (3/3)	0.031 (3/3)	23.254 (2/3)	7.142 (3/3)	0.005 (3/3)	6/8
		Worst	0.005 (3/3)	20.258 (2/3)	23.235 (2/3)	0.103 (3/3)	0.122 (3/3)	0.065 (3/3)	6.149 (3/3)	0.005 (3/3)	6/8

Tabella 4.25: *Risposta di ciascun tipo di rete addestrata con il* dataset *da tre labirinti. Si vede chiaramente che la rete PNN ha prestazioni superiori alla rete CTRNN, addestrata sia in modalità* batch *che* online, *anche sui labirinti diversi da quello usato per l'addestramento, ovvero per labirinti con* $\lambda = 2.4$ *e* $\lambda = 3.6$ *(parametro adimensionale proporzionale alle dimensioni del labirinto). Grazie a questo apprendimento con* dataset *esteso, tutte e due le architetture riescono a generalizzare bene i comportamenti multipli appresi in precedenza su percorsi di grandezza casuale.*

λ	**Rete (Addestr.)**	**B/W Case**	**Errore del programma (Numero di svolte corrette)**								**Successo Programmi**
			P_1	P_2	P_3	P_4	P_5	P_6	P_7	P_8	
2.4	CTRNN (*Online*)	*Best*	51.507 (0/3)	35.347 (1/3)	34.337 (1/3)	18.177 (2/3)	33.333 (1/3)	17.173 (2/3)	16.163 (2/3)	0.003 (3/3)	1/8
		Worst	51.507 (0/3)	35.347 (1/3)	34.337 (1/3)	18.177 (2/3)	33.333 (1/3)	17.173 (2/3)	16.163 (2/3)	0.003 (3/3)	1/8
	CTRNN (*Batch*)	*Best*	0.013 (3/3)	2.134 (3/3)	17.178 (2/3)	0.102 (3/3)	1.036 (3/3)	17.174 (2/3)	16.164 (2/3)	0.004 (3/3)	5/8
		Worst	0.003 (3/3)	16.164 (2/3)	17.179 (2/3)	1.066 (3/3)	0.036 (3/3)	17.175 (2/3)	16.163 (2/3)	0.003 (3/3)	4/8
	PNN	*Best*	0.005 (3/3)	0.052 (3/3)	0.083 (3/3)	0.061 (3/3)	0.059 (3/3)	0.062 (3/3)	0.164 (3/3)	0.005 (3/3)	8/8
		Worst	0.005 (3/3)	0.107 (3/3)	17.175 (2/3)	0.059 (3/3)	0.106 (3/3)	0.045 (3/3)	16.165 (2/3)	0.005 (3/3)	6/8
3.6	CTRNN (*Online*)	*Best*	69.687 (0/3)	47.467 (1/3)	46.457 (1/3)	24.237 (2/3)	45.453 (1/3)	23.233 (2/3)	22.223 (2/3)	0.003 (3/3)	1/8
		Worst	69.687 (0/3)	47.467 (1/3)	46.457 (1/3)	24.237 (2/3)	45.453 (1/3)	23.233 (2/3)	22.223 (2/3)	0.003 (3/3)	1/8
	CTRNN (*Batch*)	*Best*	0.017 (3/3)	3.176 (3/3)	23.240 (2/3)	1.127 (3/3)	1.040 (3/3)	23.234 (2/3)	22.225 (2/3)	0.004 (3/3)	5/8
		Worst	0.003 (3/3)	22.224 (2/3)	23.240 (2/3)	0.082 (3/3)	0.041 (3/3)	23.235 (2/3)	22.223 (2/3)	0.003 (3/3)	4/8
	PNN	*Best*	0.005 (3/3)	0.066 (3/3)	0.102 (3/3)	0.071 (3/3)	0.078 (3/3)	0.068 (3/3)	0.221 (3/3)	0.005 (3/3)	8/8
		Worst	0.005 (3/3)	0.141 (3/3)	23.235 (2/3)	0.069 (3/3)	0.128 (3/3)	0.048 (3/3)	22.225 (2/3)	0.005 (3/3)	6/8

Capitolo 5

Conclusioni

In questa tesi ho affrontato un aspetto del problema evidenziato da alcuni recenti dati sperimentali: singole aree del sistema nervoso potrebbero essere capaci di sottendere a più di un singolo comportamento e di realizzare "transizioni" da un comportamento all'altro in maniera rapida e reversibile. Tale uso delle stesse aree per differenti funzioni è, in effetti, in contrasto con l'ipotesi maggiormente accettata in letteratura secondo cui il sistema nervoso centrale sia organizzato sulla base di aree dedicate a funzioni o compiti specifici.

Pubblicazioni più recenti mostrano che tali cambiamenti di comportamento debbano avvenire in maniera rapida e reversibile (Bargmann, 2012) e questo è in contrasto con l'idea, oggi largamente condivisa, che una variazione di comportamento debba sottendere a variazioni strutturali. Infatti, tali cambiamenti strutturali (cioè variazioni dell'efficacia sinaptica) si suppone che debbano avvenire in tempi molto più lunghi, compatibilmente con qualche processo d'apprendimento. Inoltre, i cambiamenti di struttura dovuti ad apprendimento in genere non sono reversibili.

Al fine di spiegare tale proprietà di transizione rapida e reversibile tra comportamenti diversi, molti autori hanno tentato differenti approcci. In questa tesi mi sono concentrato su un *framework* di programmabilità, ovvero un'architettura composta di due circuiti neurali. Il primo consiste in una rete FNN che sia in grado di apprendere le codifiche per ciascun comportamento e passarle in input all'altro circuito. Il secondo è l'interprete, realizzato da una PNN, ovvero una rete CTRNN dotata della proprietà di programmabilità, che riceve due input: quello "classico" e l'input prodotto dall'altro circuito, cioè la codifica che permette alla rete interprete di esibire il comportamento desiderato. Pertanto, l'architettura programmabile, per come è composta, simula la presenza di una architettura non programmabile (un'unica CTRNN *full connected*), gestendone gli stessi input e gli stessi output. Il *framework* di programmabilità utilizzato si basa sull'uso di una tecnica di estrazione di reti moltiplicative in grado di controllare soglie e pesi sinaptici. La presenza di queste reti moltiplicative all'interno del CNS è tutta da dimostrare. Tuttavia, è stata

più volte ipotizzata, ad esempio, in luogo di alcune operazioni elementari di elaborazione dei dati sensoriali che necessiterebbero proprio di operazioni moltiplicative. Inoltre, la struttura neurale preposta all'operazione di moltiplicazione potrebbe essere sia una sotto-rete neurale, così come implementata nel *framework* utilizzato, ma anche un meccanismo biologico presente a livello dendritico. Ipotesi, quest'ultima, che, se verificata sperimentalmente, porterebbe vantaggi immediati, anche in vista della necessità da parte delle reti moltiplicative di funzionare su una scala temporale di almeno un ordine di grandezza inferiore rispetto a quella delle reti principali in cui esse operano (§ 2.4.1).

Se confrontiamo questa idea con le deduzioni di Bargmann (2012) sul fatto che ogni singola rete neurale possa potenzialmente rappresentare l'insieme di più circuiti latenti, in grado di eseguire ora una funzione, ora un'altra, in base al fenomeno di neuromodulazione, è facile intuire che, dal punto di vista computazionale, risulti analoga la proposta di affidare tale meccanismo ad una codifica data in ingresso alla rete che ne riprogrammi il funzionamento.

La stessa proposta di Agnati *e altri* (2010) sulla presenza di due reti, una cablata e l'altra di volume, fittamente interconnesse, suppone che tale architettura possa essere in grado di realizzare la modifica del funzionamento della rete cablata da parte di quella di volume, e viceversa. Questo comportamento è in effetti molto simile all'idea, come avviene nel nostro caso, di riprogrammare un circuito neurale.

Nel contesto dell'interpretazione in termini di programmabilità, quel che ancora manca, purtroppo, sono le evidenze sperimentali di un'architettura simile o equivalente a quella proposta. L'esperimento che ho simulato però vuole mettere in evidenza la possibilità che ci possa essere un vantaggio nell'avere, nel biologico, circuiti neurali programmabili nel senso finora discusso. A tal fine, ho considerato un *toy problem* nel quale un *robot*, controllato prima da un'architettura non programmabile e poi da un'architettura programmabile, cerca di raggiungere la terminazione indicatagli, tra le otto possibili, in un labirinto costruito *ad hoc*. Il percorso consiste di una serie di tratti rettilinei di pari lunghezza seguiti da biforcazioni, per un totale di tre svolte successive da eseguire. Il *robot* può avanzare a velocità costante in due sole modalità: *Left follower*, cioè seguire il muro a sinistra o *Right Follower*, cioè seguire il muro a destra. In tal modo, l'obiettivo di raggiungere una tra le possibili terminazioni del labirinto diventa un comportamento costituito da una sequenza di tre primitive (L o R). Il livello di semplificazione del *toy problem* affrontato impone l'uso di una rete costituita da pochi neuroni. La scalabilità dell'approccio proposto dovrebbe essere ulteriormente testata. Quanto realizzato tramite simulazioni Matlab è però sufficiente per un'analisi delle prestazioni dei due tipi di architettura di controllo esaminati.

Nel lavoro che ho svolto, sono stati effettuati venti apprendimenti per ogni tipo di architettura, programmabile o meno, e per ciascun tipo di apprendimento possibile. I risultati sperimentali mostrano che un'architettura dotata della proprietà di programmabilità riesce ad esibire correttamente fino a otto su otto comportamenti tra quelli compatibili con i vincoli scelti per lo scenario. I comportamenti vengono appresi con una frequenza di successo variabile a seconda della grandezza del *dataset* scelto per l'apprendimento. In caso di successo parziale, è garantito un minimo di almeno sei su otto comportamenti appresi correttamente. Sotto le stesse condizioni, invece, la struttura non programmabile riesce ad esibire, al più, solamente cinque comportamenti degli otto possibili. Il tutto avviene anche per labirinti di dimensioni differenti non presenti nel *training set*, sui quali sono state testate le architetture di controllo dopo l'apprendimento.

Tali risultati mettono in risalto la peculiarità dell'architettura programmabile e quindi dell'uso di PNN nell'apprendimento di comportamenti multipli: apprendere le codifiche di programma tramite una sottostruttura preposta, affidando poi ad un interprete l'apprendimento dei diversi comportamenti (architettura programmabile, fig. 3.3.1), sembra essere più efficiente che apprendere direttamente più comportamenti, come accade nel caso di un'architettura non programmabile (fig. 3.2.1).

Per quanto riguarda gli sviluppi futuri sarebbe interessante ripetere gli stessi test in situazioni più complesse, ad esempio per labirinti con un maggior numero di svolte (sequenze di programma più lunghe) oppure più "primitive". Fino ad arrivare a scenari, anche in simulazione, in cui la rete controlli effettivamente un robot immerso in un ambiente dinamico e parzialmente conosciuto.

Un altro sviluppo futuro riguarda la possibilità di utilizzare reti CTRNN come reti che apprendano e inviino codifiche di comportamento, invece di utilizzare le PNN come è stato fatto in questa tesi.

In definitiva, la programmabilità si pone come una suggestiva ipotesi interpretativa per comprendere quei fenomeni di cambiamento di comportamento con transizioni rapide e reversibili che indubbiamente caratterizzano il nostro agire.

Bibliografia

Agnati L. F.; Guidolin D.; Guescini M.; Genedani S.; Fuxe K. (2010). Understanding wiring and volume transmission. *Brain Research Reviews*, **64 (1)**, 137–159.

Anderson M. L. (2010). Neural reuse: A fundamental organizational principle of the brain. *Behavioral and Brain Sciences*, **33(4)**, 245 – 266.

Bargmann C. I. (2012). Beyond the connectome: How neuromodulators shape neural circuits. *BioEssays*, **34 (6)**, 458–465.

Beer R. D. (1995). On the dynamics of small continuous-time recurrent neural networks. *Adaptive Behaviour*, **3**(4), 469–509.

Dehaene S.; Duhamel J. R.; Hauser M.; Rizzolatti G. (2004). Evolution of human cortical circuits for reading and arithmetic: The neuronal recycling hypothesis. In *From monkey brain to human brain: A Fyssen Foundation symposium*, pp. 133–157. Cambridge, Massachusetts: MIT Press.

Donnarumma F.; Prevete R.; Trautteur G. (2010). How and over what timescales does neural reuse actually occur? *Behavioral and Brain Sciences*, **33**(4), 272–273.

Donnarumma F.; Prevete R.; Trautteur G. (2012). Programming in the brain: A neural network theoretical framework. *Connection Science*, **24 (2-3)**, 71–90.

Dunn N.; Lockery S. R.; Pierce-Shimomura J. T.; Conery J. S. (2004). A neural network model of chemotaxis predicts functions of synaptic connections in the nematode cenorhabditis elegans. *Journal of Computational Neuroscience*, **17**(2), 137–147.

Eliasmith C. (2005). A unified approach to building and controlling spiking attractor networks. *Neural Computation*, **17**(6), 1276–1314.

Funahashi K. I.; Nakamura Y. (1993). Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Ne*, **6**(6), 801–806.

Fyhn M.; Hafting T.; Treves A.; Moser M. B.; Moser E. I. (2007). Hippocampal remapping and grid realignment in entorhinal cortex. *Nature*, **446**, 190–194.

Garzillo C.; Trautteur G. (2009). Computational virtuality in biological systems. *Theoretical Computer Science*, **410**, 323–331.

Giles C. L.; Miller C. B.; Chen D.; Chen H. H.; Sun G. Z.; Lee Y. C. (1992). Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, **4**(3), 393–405.

Goldberg E. (2004). Quando il leader è colpito. In *L'anima del cervello*, pp. 135–161. UTET Libreria.

Harris-Warrick R. M.; Marder E. (1991). Modulation of neural networks for behavior. *Annual Review of Neuroscience*, **14**, 39–57.

Hochreiter S.; Younger A. S.; Conwell P. R. (2001). Learning to learn using gradient descent. *Artificial Neural Networks - ICANN 2001 - Lecture Notes in Computer Science*, **2130**, 87–94.

Hopfield J. J.; Tank D. W. (1986). Computing with neural circuits: A model. *Science, New Series*, **233**(4764), 625–633.

Hopfield J. J.; W. T. D. (1985). Neural computation of decisions in optimization problems. *Biological Cybernetics*, **52**(3), 141–152.

Hurley S. (2008). The shared circuits model (scm): How control, mirroring, and simulation can enable imitation, deliberation, and mindreading. *Behavioral and Brain Sciences*, **31**(1), 1–22.

Ito M.; Tani J. (2004). Generalization in learning multiple temporal patterns using rnnpb. *Neural Information Processing - Lecture Notes in Computer Science*, **3316**, 592–598.

Jordan M. I. (1986). Attractor dynamics and parallellism in a connectionist sequential machine. In *Proceedings of the Eighth Annual Meeting of the Cognitive Science Society*, pp. 531–546.

Kandel E.; Schwartz J.; Jessell T. (2000). *Principles of Neural Science*, capitolo 2, pp. 21–23. McGraw-Hill.

Kier R. J.; Ames J. C.; Beer R. D.; Harrison R. R. (2006). Design and implementation of multipattern generators in analog vlsi. *IEEE Transactions on Neural Networks*, **17**(4), 1025–1038.

Nishimoto R.; Namikawa J.; Tani J. (2008). Learning multiple goal-directed actions through self-organization of a dynamic neural network model: A humanoid robot experiment. *Adaptive Behavior*, **16**(2-3), 166–181.

Noelle D. C.; Cottrell G. W. (1994). Towards instructable connectionist systems In *Computational Architectures Integrating Neural And Symbolic Processes*. A cura di Sun R., Bookman L. A., volume 292 di *The Springer International Series In Engineering and Computer Science*, pp. 187–221. Springer US.

Oztop E.; Arbib M. (2002). Schema design and implementation of the grasp-related mirror neuron system. *Biological Cybernetics*, **87**(2), 116–140.

Price K. V. (1999). An introduction to differential evolution. In *New ideas in optimization*, pp. 79–108. McGraw-Hill.

Prokhorov, D. V. andFeldkarnp L. A.; Tyukin I. Y. (2002). Adaptive behavior with fixed weights in rnn: An overview. In *Proceedings of the 2002 International Joint Conference on Neural Networks*, volume 3, pp. 2018–2022.

Roy A. (2008). Connectionism, controllers, and a brain theory. *IEEE Transactions on Systems, Man and Cybernetics*, **38**(6), 1434–1441.

Sauser E.; Billard A. (2006). Parallel and distributed neural models of the ideomotor principle: An investigation of imitative cortical pathways. *Neural Networks*, **19**(3), 285–298.

Schmidhuber J. (1992). Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Naural Computation*, **4**(1), 131–139.

Schneider W.; Oliver W. L. (1991). An instructable connectionist/control architecture: Using rule-based instructions to accomplish connectionist learning in a human time scale. In *Architectures for Intelligence: The 22nd Carnegie Mellon Symposium on Cognition*, pp. 113–145.

Siegelman H. T.; Ben-hur A.; Fishmann S. (2001). Comments on attractor computation. *International Journal of Computing Anticipatory Systems*, **6**, 161–170.

Touretzky D. S. (1990). Boltzcons: Dynamic symbol structures in a connectionist network. *Artificial Intelligence, 1990*, **46**(1-2), 5–46.

Indice analitico