SAPIENZA
UNIVERSITÀ DI ROMA

FACOLTA' DI INGEGNERIA

Tesi di Laurea Specialistica in Ingegneria Elettronica

Apprendimento per dimostrazione in un robot umanoide

Relatore Interno:
Prof. Giuseppe Oriolo
.....
Relatore Esterno:
Dott. Stefano Nolfi
.....

Laureando:
Francesca Pettine
Matr. 1051431

Anno Accademico 2009 - 2010

Alla mia famiglia

Indice

Ringraziamenti

Un sentito ringraziamento va al Dott. Stefano Nolfi, responsabile del laboratorio LARAL (*Laboratory of Autonomous Robotics and Artificial Life*) dell'ISTC, per la grande disponibilità e cortesia dimostratemi, e per tutto l'aiuto fornito durante la preparazione e la stesura di questo lavoro.

Al personale dell'Istituto e del laboratorio va la mia gratitudine per la gentile accoglienza e la fattiva collaborazione. Un grazie particolare va a Gianluca Massera che mi ha seguito e guidato con i suoi preziosi suggerimenti durante il lavoro di tesi.

Un doveroso ringraziamento va al Prof. Giuseppe Oriolo per i suoi insegnamenti e per l'attenzione che mi ha dedicato in questo lavoro di tesi.

Ringrazio inoltre con affetto i miei genitori per il loro incrollabile sostegno morale e per il grande aiuto che mi hanno dato durante tutto il percorso di studi. Grazie anche a Chiara e Lara per essermi state vicine ed avermi supportato (e sopportato) durante questo ultimo periodo e non solo.

Infine, un grazie di cuore va ad Alberto per essere entrato nella mia vita, per avermi sostenuto nei momenti di sconforto e per aver esultato con me nei momenti di gioia.

Introduzione

L'obiettivo di questa tesi è quello di realizzare un sistema di controllo per un robot umanoide che permetta al robot stesso di sviluppare, attraverso un processo di apprendimento guidato dall'utente, la capacità di esibire dei comportamenti complessi combinando in sequenza e/o in parallelo delle capacità comportamentali preesistenti.

Per perseguire tale scopo si è deciso di utilizzare il robot iCub[1], di dotare tale robot di un controllore neurale che opera elicitando una serie di primitive comportamentali, e di addestrare tale neuro-controllore attraverso un processo di apprendimento per dimostrazione. Durante il processo di apprendimento lo sperimentatore impartisce al robot dei comandi "linguistici" che elicitano l'esecuzione delle primitive motorie corrispondenti in modo che il robot apprenda ad associare allo stato senso-motorio corrente la primitiva motoria appropriata.

L'attività di ricerca condotta nell'ambito della tesi si inquadra dunque nel contesto della robotica adattiva, cioè di quegli approcci che cercano di progettare delle architetture di controllo per il robot che consentano al robot stesso di acquisire le proprie capacità attraverso un processo di apprendimento. In particolare l'obiettivo del progetto è quello di verificare se sia possibile estendere i modelli di apprendimento mediati dal linguaggio, proposti da Zhang e Weng in [1] e Dominay et al. in [2], all'acquisizione di capacità autonome cioè alla capacità di elicitare l'azione appropriata

[1] Cfr. http://www.robotcub.org/

o la sequenza di azioni appropriate nel contesto senso-motorio corrente riducendo progressivamente la dipendenza del robot dai comandi linguistici prodotti dal mediatore umano.

L'organizzazione del lavoro di questa tesi, con riferimento ai diversi capitoli che la compongono viene descritta sinteticamente di seguito.

Nel capitolo 1 vengono introdotti gli aspetti teorici fondamentali delle reti neurali artificiali e alcuni algoritmi di apprendimento. Particolare attenzione viene data alla *backpropagation*, algoritmo utilizzato per realizzare gli esperimenti.

Nel capitolo 2 vengono introdotti i concetti di base dell'apprendimento per dimostrazione in ambito robotico.

Nel capitolo 3 viene descritto il sistema sviluppato e la procedura di apprendimento. In particolare, vengono proposti due controllori neurali alternativi che si differenziano nelle modalità di codifica delle primitive.

Nel capitolo 4 vengono illustrati e confrontati i risultati ottenuti dall'utilizzo in simulazione di entrambi i controllori neurali oggetto di sperimentazione.

Nel capitolo 5 viene descritta la fase di verifica sperimentale sul robot iCub disponibile presso il laboratorio LARAL dell'istituto ISTC del CNR.

Nella sezione conclusiva, infine, vengono riassunti i risultati ottenuti e le implicazioni rispetto allo stato dell'arte.

Capitolo 1

Reti neurali artificiali

In questo Capitolo vengono presentati gli aspetti teorici fondamentali riguardanti le reti neurali artificiali e, a partire dal paragrafo 1.5, i principali algoritmi di apprendimento. In particolare, nel paragrafo 1.7.1, viene descritto dettagliatamente l'algoritmo di retropropagazione dell'errore, comunemente indicato con il nome di algoritmo di *backpropagation*.

1.1 Aspetti generali

Le reti neurali artificiali sono dei sistemi di elaborazione dell'informazione che cercano di simulare all'interno di un sistema informatico il funzionamento dei sistemi nervosi biologici, costituiti da un gran numero di cellule nervose dette neuroni, collegate tra di loro in una complessa rete. In altre parole le reti neurali artificiali sono quindi dei modelli computazionali che tentano di riprodurre e catturare i principi base di un sistema nervoso biologico. Ogni unità della rete implementa una semplice operazione che consiste nel diventare attiva se la somma di tutti i segnali entranti è più grande di una certa soglia. Un neurone attivo emette a sua volta un segnale che raggiunge tutte le unità al quale è connesso. La connessione opera come un filtro che moltiplica il

segnale per un peso positivo o negativo, noto come peso sinaptico. La risposta di una rete neurale artificiale ad un certo stimolo proveniente dall'ambiente dipende dalla sua architettura e dalle connessioni esistenti tra le varie unità.

Le reti neurali apprendono tramite un processo di modifica dei pesi sinaptici che si basa sull'esposizione della rete stessa ad un insieme di esempi (apprendimento per dimostrazione). Nella maggior parte dei casi infatti l'apprendimento richiede la presentazione ripetuta di un insieme di dati di ingresso, detti *pattern*. Esistono diversi tipi di regole di apprendimento, ognuna caratterizzata da specifiche funzionalità e applicabile a specifiche architetture ma, tipicamente, tutte le connessioni sinaptiche della stessa rete neurale artificiale cambiano in accordo alla stessa regola di apprendimento.

Generalmente ogni rete neurale ha caratteristiche strettamente dipendenti sia dal modello utilizzato che dal contesto in cui è inserita. Tuttavia vi sono alcune caratteristiche sufficientemente generali che valgono qualunque sia il modello utilizzato e che rendono le reti neurali artificiali interessanti nelle applicazioni ingegneristiche:

- **Robustezza.** Una rete neurale è tipicamente in grado di rispondere correttamente anche quando alcune delle sue connessioni vengono lesionate o nel caso sia presente del rumore agli ingressi o alle uscite dei neuroni della rete. All'aumentare dell'entità del rumore le prestazioni di norma subiscono un decadimento graduale e l'errore commesso dalla rete tende ad aumentare in modo lento e uniforme su tutto il campo di risposta.

- **Flessibilità.** Grazie alla possibilità di far apprendere alle reti neurali il compito desiderato in base all'esperienza, queste possono essere impiegate per un gran numero di finalità diverse senza dover conoscere approfonditamente le proprietà del dominio di applicazione. Naturalmente il tipo di modello di rete neurale che viene scelto tiene conto del compito che si vuole svolgere ma non è necessario che

l'utente conosca la soluzione analitica del problema.

- **Generalizzazione.** Nonostante la rete neurale venga tipicamente addestrata con un numero ridotto di esempi già risolti del problema posto, questa è in grado di percepire le somiglianze tra uno stimolo in ingresso mai visto e quelli inclusi nella fase di addestramento. Attraverso queste somiglianze la rete neurale è in grado di produrre una risposta soddisfacente. La motivazione è dovuta alla particolarità delle reti neurali di estrarre delle caratteristiche invarianti dagli stimoli in ingresso piuttosto che memorizzarli singolarmente. La potenza di generalizzazione di una rete dipende, quindi, oltre che dal modello scelto dall'entità delle caratteristiche invarianti che possono trovarsi negli stimoli.

1.2 Dal neurone biologico a quello artificiale

Le reti neurali artificiali sono una semplificazione di quello che è il sistema nervoso degli esseri viventi. In base a come il neurone biologico viene modellizzato si riconoscono diversi modelli di reti neurali. Un primo esempio è il modello HH [Hodgkin, Huxley 1952] sul quale sono basate le *Pulsed Neural Networks* [Maas, Bishop 1998], dove il neurone artificiale è molto fedele al comportamento dei neuroni biologici. Di seguito, invece, si presenta un modello molto semplificato del neurone che deriva dai primi studi svolti da McCulloch e Pitts sulle reti neurali artificiali [McCulloch, Pitts 1943]. I neuroni biologici sono le cellule del sistema nervoso nei quali, senza entrare nei dettagli, si possono individuare tre parti funzionali:

- i **dendriti**, solitamente più di uno, trasmettono gli stimoli in maniera centripeta, ovvero li convogliano verso il soma del neurone;

- il **soma** è la parte centrale del neurone nel quale vengono raccolti gli stimoli provenienti dai dendriti e, in base a questi, viene generato il segnale da trasmettere

all'assone;

- l'**assone** trasmette gli stimoli in maniera centrifuga, ovvero verso le altre cellule. Da esso partono delle piccole ramificazioni in grado di trasmettere il segnale ai dendriti degli altri neuroni tramite sinapsi, che può essere inibitoria o eccitatoria a seconda dell'effetto che induce sul dendrite.

1.3 Modello del neurone artificiale

Partendo dalla struttura di neurone biologico in Figura 1.1 sulla sinistra, è possibile ottenere un modello semplificato di neurone artificiale (in Figura 1.1 sulla destra). Come si evince dalla Figura 1.1, il complesso albero dendritico, le interazioni chimiche

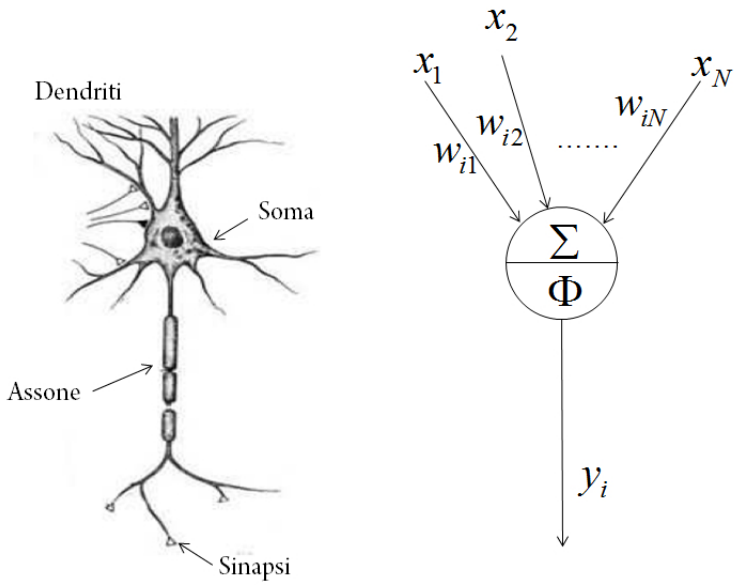

Figura 1.1: Confronto tra neurone biologico a sinistra e artificiale a destra.

tra sinapsi e dendriti ed altri dettagli vengono completamente ignorati ed il tutto è modellizzato con semplici connessioni pesate in ingresso. Anche il soma e l'assone

del neurone vengono semplificati e modellizzati tramite una funzione d'attivazione e una semplice variabile di uscita. Di conseguenza il neurone artificiale è generalmente caratterizzato da un insieme di connessioni pesate, una soglia e una funzione di attivazione e risulta un componente a N ingressi e un'uscita y.

Ignorando i ritardi di trasmissione, il *net input* o l'attivazione del neurone $i-esimo$, a_i, è la somma pesata di tutti gli N segnali di ingresso x_j connessi al neurone $i-esimo$ stesso:

$$a_i = \sum_{j=1}^{N} w_{ij} x_j \tag{1.1}$$

dove i w_{ij} sono i pesi sinaptici.

La risposta y_i del neurone $i-esimo$ viene calcolata attraverso una funzione di attivazione Φ:

$$y_i = \Phi(a_i) = \Phi\left(\sum_{j=1}^{N} w_{ij} x_j - \theta_i\right) \tag{1.2}$$

in cui una soglia θ_i viene tipicamente sottratta alla somma pesata degli ingressi del neurone.

La funzione d'attivazione risulta, quindi, la parte più critica del modello del neurone artificiale presentato in quanto la sua scelta determina le funzionalità del neurone e di conseguenza i limiti delle reti neurali risultanti. Alcune tipologie di funzioni di attivazione vengono brevemente descritte di seguito.

- Le funzioni gradino sono del tipo

$$\Phi(a_i) = \begin{cases} 1 & se \quad \sum_{j=1}^{N} w_{ij} x_j > \theta_i \\ 0 & altrimenti \end{cases} \tag{1.3}$$

  e permettono di distinguere tra due soli stati solitamente detti stato inibito e stato eccitato. Per tale motivo i neuroni caratterizzati da questo tipo di funzione di attivazione vengono detti neuroni con uscita binaria e le reti neurali che li usano sono caratterizzate da una logica binaria.

- Le funzioni lineari del tipo

$$\Phi(a_i) = ka_i \tag{1.4}$$

in cui k è una costante, permettono ai neuroni di avere in uscita un valore che è proporzionale all'intensità degli ingressi rendendo di conseguenza il comportamento della rete neurale di carattere tipicamente lineare.

- Le funzioni sigmoidali o logistiche sono caratterizzate dall'espressione seguente

$$\Phi(a_i) = \frac{1}{1 + e^{-ka_i}} \tag{1.5}$$

in cui k è un fattore di scala che determina l'inclinazione della funzione (v. Figura 1.2). Tali funzioni d'attivazione sono continue e non lineari e tendono asintoticamente a 0 e 1.

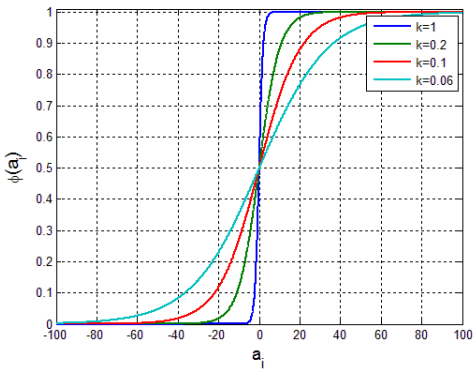

Figura 1.2: Andamento della funzione logistica al variare di k.

1.4 Architetture delle reti neurali

La modalità con cui i neuroni vengono interconnessi fra di loro influenza notevolmente il comportamento globale della rete. Per questo motivo la topologia con la quale i

neuroni vengono distribuiti e il modo in cui sono collegati tra loro rivestono particolare importanza.

I neuroni di una rete vengono divisi in tre categorie: neuroni di ingresso, interni e di uscita. I primi sono responsabili dell'analisi degli stimoli provenienti dall'ambiente e per questo sono spesso associati a sensori che misurano grandezze fisiche dell'ambiente, nel qual caso prendono il nome di neuroni sensoriali. I neuroni di uscita forniscono la risposta della rete agli stimoli ricevuti in ingresso e il loro valore viene tipicamente utilizzato per interagire con l'ambiente esterno. Infine i neuroni interni sono neuroni che non interagiscono con l'ambiente e sono in grado di comunicare solo con le altre unità all'interno della rete.

Le possibili topologie sono tantissime e limitate soltanto dalla fantasia. Tuttavia in tutta questa varietà si distinguono due classi molto importanti di architetture che devono il loro nome al modo in cui sono connessi i neuroni: *reti feedforward* e *reti ricorrenti*.

La tipologia architetturale più semplice è quella delle cosiddette *reti neurali feedforward* in cui l'uscita di ogni neurone dipende unicamente dall'uscita dei neuroni degli strati più bassi. Le reti feedforward sono dispositivi ingresso-uscita che, a causa della loro struttura, non possono produrre sequenze o dinamiche temporali, a meno che non siano composte da neuroni dinamici.

Un modo per dotare le reti neurali costituite da neuroni statici di dinamiche temporali è quello di aggiungere delle connessioni ricorrenti tra neuroni dello stesso strato o tra neuroni di strati differenti, dando così luogo alle cosiddette *reti neurali ricorrenti*. Nelle reti ricorrenti quindi la variabile temporale acquista un ruolo dominante, basti pensare al concetto di ritardo nella trasmissione di un segnale lungo una connessione. La presenza o meno di tale ritardo può giocare un ruolo decisivo nel momento in cui l'uscita del neurone è collegata a uno dei suoi ingressi, perché il ritardo temporale permette al neurone di poter avere in ingresso contemporaneamente una sua

decisione passata e stimoli correnti. In questo caso, la risposta al tempo t di un neurone con connessioni ricorrenti è data da

$$y_i^t = \Phi\left(\sum_j^N w_{ij}x_j^t + \sum_l^M r_{il}q_l^{t-1} - \theta_i\right) \tag{1.6}$$

dove q_l^{t-1} sono le uscite dei neuroni nello stesso strato (incluso anche il neurone i) al passo precedente e r_{il} sono i pesi delle connessioni ricorrenti. Per questo motivo alcune reti ricorrenti possono essere viste come delle memorie associative, come accade nel modello di Hopfield [Hopfield 1982].

1.5 Plasticità sinaptica

La capacità di adattarsi è una delle caratteristiche più importanti di un sistema nervoso, permettendo agli organismi di modificare e sviluppare i propri comportamenti con lo scopo di mantenere o migliorare la loro capacità di sopravvivenza in ambienti parzialmente sconosciuti e dinamici. Le reti neurali artificiali possono riprodurre tale capacità di adattamento tramite l'adozione di diversi algoritmi. Questi algoritmi possono fondamentalmente dividersi in due grandi famiglie:

- **Apprendimento non supervisionato**: include algoritmi che permettono alla rete di estrarre informazioni statisticamente significanti dalla distribuzione di pattern di ingresso o di memorizzare e ricostruire i pattern di ingresso stessi.

- **Apprendimento supervisionato**: include algoritmi che guidano il processo di addestramento della rete tenendo conto della risposta desiderata che la rete dovrebbe fornire per un dato insieme di *training pattern*. L'apprendimento supervisionato inoltre comprende gli algoritmi di apprendimento per rinforzo [4], che modificano la rete a seconda di feedback positivi o negativi ricevuti dall'ambiente ad intervalli irregolari.

Tutti gli algoritmi di apprendimento mostrano alcune caratteristiche comuni, indipendentemente da come vengono implementati. Primo tra tutti, l'apprendimento consiste nella presentazione ripetuta di un insieme di pattern di ingresso, conosciuti anche come training pattern. Nell'apprendimento non supervisionato, i training pattern sono i vettori di ingresso che vengono presentati alla rete, mentre nell'apprendimento supervisionato sono coppie di vettori che rappresentano l'ingresso e l'uscita desiderata corrispondente. L'apprendimento di una rete neurale consiste nel calcolare una modifica dei suoi pesi sinaptici, Δw_{ij}, secondo un qualche algoritmo e ciò può essere fatto online o offline a seconda che tale modifica venga calcolata dopo ogni presentazione di un training pattern e la corrispondente attivazione della rete oppure dopo la presentazione di tutti i training pattern e le relative attivazioni della rete rispettivamente. I nuovi pesi sinaptici vengono tipicamente calcolati aggiungendo la modifica Δw_{ij} ai pesi sinaptici correnti:

$$w_{ij}^{t+1} = w_{ij}^{t} + \Delta w_{ij}^{t}. \tag{1.7}$$

Per tale motivo gli algoritmi di apprendimento si concentrano solamente sul calcolo di Δw_{ij}. La relazione (1.7) mostra come il processo di apprendimento nelle reti neurali consista fondamentalmente nell'aggiungere nuova conoscenza Δw_{ij} ad una conoscenza preesistente w_{ij}. Al fine di ridurre l'interferenza tra la conoscenza precedente e quella nuova, l'apprendimento viene effettuato in modo incrementale presentando gli stessi training pattern più volte e aggiungendo ai vecchi pesi solo una piccola frazione della modifica calcolata dalla regola di apprendimento. L'ampiezza di questa frazione è un valore $0 \leq \eta \leq 1$, detto *learning rate*, che controlla la velocità dell'apprendimento della rete ed è utilizzato per assicurare la stabilità del processo di apprendimento.

Va infine fatto notare che la maggior parte degli algoritmi di apprendimento è basata, o comunque ispirata dalla legge di Hebb sulla plasticità sinaptica che, per

neuroni senza dinamica temporale può essere formalizzata come segue

$$\Delta w_{ij} = \eta y_i x_j \tag{1.8}$$

in cui Δw_{ij} è la modifica che deve essere effettuata sul peso w_{ij}, η è il learning rate e y_i, x_j sono i valori di attivazione delle unità postsinaptiche e presinaptiche rispettivamente.

1.6 Apprendimento non supervisionato

Nell'apprendimento non supervisionato tipicamente la rete neurale impara alcune proprietà caratteristiche dell'insieme di pattern di ingresso, senza alcun feedback proveniente dall'ambiente o dall'utente. In altre parole, in questo caso l'apprendimento consiste nell'estrazione di alcune caratteristiche comuni o distintive che permettono alla rete di classificare i pattern di ingresso. Da un punto di vista matematico, l'apprendimento non supervisionato realizza operazioni statistiche come il calcolo di indici di correlazione, stima di parametri delle funzioni di densità di probabilità dei pattern in ingresso, etc. Al fine di effettuare tali operazioni, la distribuzione di pattern di ingresso deve essere ridondante per permettere il riconoscimento della struttura [Barlow 1989].

Tra gli esempi di algoritmi di apprendimento non supervisionato si ricordano la *regola di Oja* [Oja 1989] e la *regola di Sanger* [1989], proposte come modifiche alla regola di Hebb per il riconoscimento e l'estrazione di caratteristiche dai dati. Per quanto concerne algoritmi di formazione di memoria si fa riferimento a Hopfield [Hopfield 1982] che mostrò come la regola di Hebb, quando applicata a reti di neuroni interconnessi, genera reti neurali risultanti in grado di memorizzare i pattern e di ricostruirli a partire da versioni incomplete o corrotte. Infine, degne di nota sono le mappe auto-organizzanti o reti SOM (*Self-Organizing Maps*), elaborate da Teuvo Kohonen dell'Università Tecnologica di Helsinki. L'algoritmo di apprendimento in questione è senza dubbio

una brillante formulazione di apprendimento non supervisionato e ha dato luogo a un gran numero di applicazioni nell'ambito dei problemi di classificazione.

1.7 Apprendimento supervisionato

L'apprendimento supervisionato è caratterizzato dalla presenza di un addestratore che fornisce la risposta desiderata richiesta dalla rete per ogni training pattern in ingresso. Secondo l'approccio originariamente proposto da Rosenblatt [Rosenblatt 1962], i pesi sinaptici vengono modificati in modo da ridurre l'errore tra la risposta desiderata e la risposta fornita dalla rete.

Si consideri una rete neurale feedforward con un solo strato di pesi sinaptici tra le unità di ingresso e di uscita e con funzione di attivazione lineare. Dato un insieme di training pattern composto da M coppie di vettori di ingresso $\vec{x}^{\mu}$ e di vettori delle corrispondenti risposte desiderate $\vec{t}^{\mu}$, con $\mu = 1, .., M$, si vuole trovare l'insieme di pesi sinaptici che facciano in modo che la risposta della rete corrisponda a quella desiderata per tutti gli M pattern. Le prestazioni di una rete neurale possono essere descritte dalla funzione d'errore

$$E_W = \frac{1}{2} \sum_{\mu} \sum_{i} (t_i^{\mu} - y_i^{\mu})^2 = \frac{1}{2} \sum_{\mu} \sum_{i} \left(t_i^{\mu} - \sum_{j} w_{ij} x_j^{\mu} \right)^2 \tag{1.9}$$

che rappresenta l'errore quadratico medio tra la risposta desiderata e la risposta fornita dalla rete in esame. La funzione d'errore dipende unicamente dai pesi sinaptici e può essere minimizzata cambiando i pesi nella direzione opposta al gradiente della funzione di errore rispetto ai pesi stessi. Derivando la funzione di errore rispetto ai pesi si ottiene la regola per l'apprendimento:

$$\Delta w_{ij} = \sum_{\mu} (t_i^{\mu} - y_i^{\mu}) x_j^{\mu} \tag{1.10}$$

che, quando applicata dopo la presentazione di ogni training pattern μ, diventa

$$\Delta w_{ij} = \eta \, (t_i - y_i) \, x_j \tag{1.11}$$

dove η è il learning rate.

Tipicamente i pesi sinaptici iniziali vengono inizializzati con piccoli valori casuali centrati intorno allo zero e i training pattern vengono presentati più volte al fine di minimizzare la funzione d'errore. Spesso risulta utile monitorare la E_W totale dopo ogni presentazione di un training set completo M. L'algoritmo appena descritto è conosciuto come *regola di Widrow-Hoff* dal nome dei suoi autori [Widrow, Hoff 1960], o più spesso come *regola delta* per enfatizzare il ruolo della differenza $\delta = t - y$ tra la risposta desiderata e quella prodotta dalla rete. La regola delta viene spesso scritta nel modo seguente

$$\Delta w_{ij} = \eta \delta_i x_j \tag{1.12}$$

e può essere matematicamente dimostrato che l'algoritmo riesce a trovare un insieme di pesi sinaptici che minimizzano la funzione d'errore se i training pattern sono linearmente separabili (due insieme di punti si dicono linearmente separabili se è possibile tracciare una linea tra questi). Poiché reti costituite da un solo strato di connessioni possono apprendere la relazione tra ingressi ed uscite solamente nei casi linearmente separabili, è necessario in tutti gli altri casi utilizzare architetture più complesse che comprendano anche strati di neuroni interni con funzioni di uscita non lineari. A tal proposito, le reti neurali con unità interne e funzioni di uscita non lineari possono in linea di principio realizzare ogni mapping tra ingresso e uscita, e necessitano di una generalizzazione della regola delta per l'apprendimento.

Se l'addestramento ha successo la rete impara a riconoscere la relazione incognita che lega le variabili d'ingresso a quelle d'uscita ed è quindi in grado di fare previsioni anche laddove l'uscita non è nota a priori. In altri termini, l'obiettivo finale dell'apprendimento supervisionato è la previsione del valore dell'uscita per ogni valore valido dell'ingresso, basandosi soltanto su un numero limitato di esempi di corrispondenza (vale a dire, coppie di valori ingresso-uscita). Per fare ciò la rete deve

essere quindi dotata di un'adeguata capacità di generalizzazione, con riferimento a casi ad essa ignoti.

1.7.1 Retropropagazione dell'errore

Addestrare una rete con unità interne e funzioni d'uscita non lineari richiede due modifiche alla regola delta: considerazioni sul comportamento non lineare introdotto dalla funzione d'uscita e un metodo per calcolare il contributo delle unità interne all'errore misurato alle unità di uscita. Il metodo di *retropropagazione dell'errore* (*backpropagation of error*)[Rumelhart et al. 1986], anche noto come *generalizzazione della regola delta*, fornisce una soluzione che può essere applicata ad ogni rete con un numero arbitrario di neuroni e di strati interni di connessione.

Il cuore dell'algoritmo consiste nel calcolare il contributo dell'errore delle unità interne propagando l'errore calcolato alle unità di uscita indietro alle unità interne tramite le stesse connessioni pesate utilizzate nella connessione diretta dalle unità interne a quelle di uscita (da qui il motivo del nome *backpropagation of error*). L'algoritmo di backpropagation dell'errore implementa un metodo gradiente discendente della funzione d'errore E_W (v. equazione (1.9)) per reti di un numero arbitrario di strati e neuroni. L'algoritmo è talmente potente e generale che è diventato uno dei metodi più utilizzati per risolvere problemi computazionali e ingegneristici che fanno uso di reti neurali artificiali. Il metodo in realtà era già stato proposto precedentemente in contesti differenti [Bryson and Ho 1969; D.B.Parker 1985; Werbos 1974].

Per spiegare l'algoritmo si consideri la semplice rete neurale multistrato in Figura 1.3 e i simboli associati ai suoi elementi. I valori delle unità di ingresso sono determinati dai pattern di ingresso. La risposta delle unità interne e di uscita è calcolata utilizzando la funzione sigmoidale descritta nell'equazione (1.5). La versione online dell'algoritmo si sviluppa come segue.

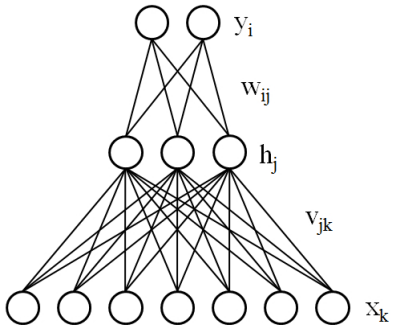

Figura 1.3: Rete neurale feedforward multistrato.

1. Inizializzare tutti i pesi con valori piccoli casuali centrati intorno allo zero.

2. Settare i valori delle unità di ingresso $\vec{x}$ con quelli del training pattern corrente $\vec{s}$:

$$x_k^\mu = s_k^\mu$$

3. Calcolare i valori delle unità interne:

$$h_k^\mu = \Phi\left(\sum_k v_{jk} x_k^\mu\right)$$

dove $\Phi(\cdot)$ è la funzione di attivazione sigmoidale (cfr. equazione 1.5).

4. Calcolare i valori delle unità di uscita:

$$y_i^\mu = \Phi\left(\sum_j w_{ij} h_j^\mu\right)$$

5. Calcolare l'errore delta tra output desiderato e output calcolato, per ogni unità di uscita della rete. Si noti che l'errore è moltiplicato per la derivata prima della funzione d'errore di quel nodo perché si stanno utilizzando funzioni non lineari (ciò non è stato necessario nella regola delta mostrata nell'equazione (1.11) perché la funzione d'errore in quel caso è lineare).

$$\delta_i^\mu = \dot{\Phi}\left(\sum_j w_{ij} h_j^\mu\right)(t_i^\mu - y_i^\mu)$$

La derivata prima della funzione sigmoidale può essere convenientemente espressa in termini dell'uscita dell'unità stessa. Nel caso delle unità di uscita, per esempio,

$$\dot{\Phi}\left(\sum_j w_{ij} h_j^\mu\right) = y_i^\mu \left(1 - y_i^\mu\right)$$

6. Propagando l'errore suddetto dalle unità di uscita alle unità interne tramite i pesi di connessione è possibile calcolare l'errore tra le unità interne desiderate e quelle calcolate. Si noti che l'indice nella somma dei delta pesati è i e si riferisce alle unità di uscita. Come prima, l'errore delta deve essere moltiplicato dalla derivata prima della funzione di uscita dell'unità:

$$\delta_j^\mu = \dot{\Phi}\left(\sum_k v_{jk} x_k^\mu\right) \sum_i w_{ij} \delta_i^\mu$$

7. Calcolare quindi le modifiche da effettuare sui pesi sinaptici dei due strati moltiplicando gli errori delta alle unità postsinaptiche con le uscite delle unità presinaptiche:

$$\Delta w_{ij}^\mu = \delta_i^\mu h_j^\mu$$
$$\Delta v_{jk}^\mu = \delta_j^\mu x_k^\mu$$

8. Infine, aggiornare i valori dei pesi aggiungendo una porzione η delle modifiche calcolate al passo precedente:

$$w_{ij}^t = w_{ij}^{t-1} + \eta \Delta w_{ij}^\mu$$
$$v_{jk}^t = v_{jk}^{t-1} + \eta \Delta v_{jk}^\mu$$

dove η è il learning rate ed è tipicamente minore di 1. Se necessario, una volta calcolati i pesi modificati, è possibile normalizzare tali pesi secondo le relazioni:

$$\begin{aligned} w_{ij}^t &= \beta \frac{w_{ij}^t}{\|w_j^t\|_2} \\ v_{jk}^t &= \beta \frac{v_{jk}^t}{\|v_k^t\|_2} \end{aligned} \tag{1.13}$$

in cui β è un coefficiente moltiplicativo determinabile empiricamente.

Ogni coppia di valori di ingresso e corrispondente uscita desiderata nel training set viene presentato più volte in ordine casuale fino a che l'errore quadratico medio totale calcolato su tutte le unità di uscita i e su tutti i training pattern μ,

$$TSS = \frac{1}{M}\sum_{\mu}^{M}\left(\frac{1}{N}\sum_{i}^{N}(t_i^\mu - y_i^\mu)^2\right)$$

raggiunge un valore piccolo.

Nel caso in cui la rete neurale di interesse fosse caratterizzata anche da connessioni ricorrenti, per applicare l'algoritmo di backpropagation occorre utilizzare alcune strategie. Una di queste consiste nell'aggiungere degli ulteriori neuroni di ingresso che mantengano una copia delle attivazioni dei neuroni ricorrenti al tempo precedente. Nell'architettura proposta da Elman nel 1990 in Figura 1.4, le unità di memoria mantengono una copia dei valori delle unità interne al tempo precedente e sono connesse alle unità interne stesse con connessioni dirette.

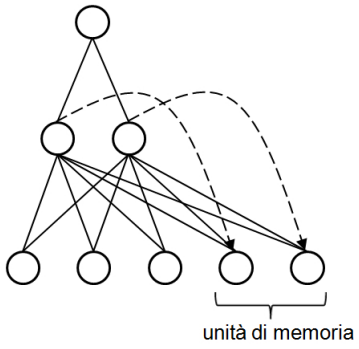

Figura 1.4: Architettura di Elman.

Capitolo 2

Apprendimento per dimostrazione

In questo Capitolo vengono presentati i concetti base dell'apprendimento per dimostrazione (LfD, *Learning from Demonstration*) in ambito robotico. In particolare, nella prima sezione di introduzione vengono descritte brevemente le motivazioni che hanno portato alla nascita di tali metodi di apprendimento, gli sviluppi storici di tali metodi e i fondamenti della formulazione del problema. Nella sezione 2.2 si descrivono le principali scelte di progetto che devono essere affrontate nella fase di dimostrazione, ovvero le modalità con cui effettuare le dimostrazioni e la scelta di cosa imitare. A seguire, nella sezione 2.3 vengono illustrate brevemente le tecniche maggiormente utilizzate per derivare la strategia d'azione a partire dagli esempi forniti e per concludere, nella sezione 2.4 vengono riportate le principali cause di limitazione che tipicamente si riscontrano nei metodi di apprendimento per dimostrazione.

2.1 Introduzione

L'apprendimento per dimostrazione [6],[7],[8],[9], è un argomento che di recente ha suscitato grande interesse in ambito robotico, abbracciando diverse aree di ricerca tra cui quella relativa all'apprendimento automatico, all'interazione tra uomo e robot e molte altre. La motivazione principale che ha portato alla nascita dei metodi di apprendimento per dimostrazione, spostandosi dalla pura preprogrammazione dei robot verso interfacce più flessibili basate sull'interazione tra uomini e robot risiede nella volontà di minimizzare, se non addirittura eliminare, l'esplicita e complicata programmazione dei compiti da parte di un utente umano esperto del settore. Dotare i robot della capacità di apprendere attraverso le dimostrazioni dei compiti di interesse consente un maggior orientamento verso una componente di integrazione sociale e culturale della robotica, permettendo anche ad utenti non specialisti del settore di insegnare ai robot ad eseguire determinati compiti. La speranza è che tramite questi metodi i robot mostrino un più alto grado di flessibilità e versatilità che permetta loro di interagire naturalmente con utenti umani e di dimostrare abilità simili.

In robotica, il problema di apprendere la trasformazione tra lo spazio degli stati e quello delle azioni è alla base di molte applicazioni. L'identificazione di tale trasformazione tra i due spazi, nel seguito indicata con il termine di strategia, rende il robot in grado di scegliere un'azione sulla base dello stato corrente. L'apprendimento per dimostrazione si colloca in questo contesto in quanto è uno degli approcci tramite cui è possibile risolvere il problema dell'apprendimento della strategia. Nell'apprendimento per dimostrazione la strategia è appresa attraverso le dimostrazioni effettuate da un insegnante del comportamento desiderato che si vuole ottenere dal robot.

La caratteristica fondamentale che distingue i metodi di apprendimento per dimostrazione dagli altri metodi di apprendimento presenti in letteratura, tra cui apprendimento per rinforzo [4] e algoritmi di apprendimento evolutivi [3], consiste nel

fatto che le dimostrazioni consentono di fornire informazioni dettagliate in grado di guidare il processo di apprendimento. Le dimostrazioni permettono inoltre di limitare l'apprendimento alle aree dello spazio degli stati incontrate durante l'esecuzione del compito nella fase di dimostrazione e consentono quindi di velocizzare il processo di apprendimento stesso rispetto agli altri metodi. Ciò nonostante numerosi sono i tentativi di combinare le diverse tecniche di apprendimento al fine di ottenere sistemi più efficienti. In questo contesto un lavoro interessante è quello recentemente proposto da Kormushev et al. in [10], in cui viene sviluppato un metodo di apprendimento che prevede una prima fase di dimostrazione del compito e fasi di apprendimento incrementale che utilizzano tecniche di rinforzo.

2.1.1 Cenni storici

La nascita dell'apprendimento per dimostrazione si può far risalire all'inizio degli anni ′80 nel contesto della robotica industriale. Tale approccio apparve come un metodo promettente per automatizzare il tedioso lavoro di programmazione manuale e per ridurre i costi necessari allo sviluppo e al mantenimento dei robot nelle industrie.

Ispirati dagli sviluppi dell'intelligenza artificiale, i primi lavori in questo campo prevedevano l'adozione del ragionamento simbolico in combinazione con operazioni di teleoperazione, tramite le quali è possibile far muovere a distanza il robot con l'utilizzo di joystick o tastiere apposite. In questi lavori le informazioni relative agli stati e alle azioni registrate venivano segmentate in un numero discreto di obiettivi da raggiungere e in appropriate azioni: il controllore era fondamentalmente una macchina a stati finiti basata su semplici regole del tipo *if-then*.

Fondamentale fu il contributo di Muench et al. [11], che suggerirono l'utilizzo di tecniche di apprendimento automatico nell'ambito dell'apprendimento per dimostrazione e stabilirono molti dei concetti chiave del LfD in ambito robotico. Tra questi ultimi si citano il fatto che fosse necessario un qualche criterio di generalizzazione

che consentisse di riprodurre le capacità apprese in situazioni nuove, il fatto che occorresse un sistema per valutare i tentativi di riproduzione, e il ruolo dell'utente durante l'apprendimento. Tale influsso dell'apprendimento automatico nell'ambito dell'apprendimento per dimostrazione ha consentito l'utilizzo di validi strumenti tra i quali reti neurali artificiali, logica fuzzy e modelli di Markov nascosti per lo sviluppo di nuovi metodi, ma i progressi fondamentali effettuati nel tempo in questo ambito riguardano soprattutto lo sviluppo di interfacce più facilmente utilizzabili da parte degli utenti.

Il recente aumento di interesse verso lo sviluppo di robot mobili e umanoidi ha reso l'apprendimento per dimostrazione in ambito robotico un campo interdisciplinare, coinvolgendo sempre più diversificate aree di ricerca tra cui quelle che si occupano di studiare i meccanismi neurali che entrano in gioco nei processi di imitazione visuo-motoria nei primati e quelle che si interessano dello studio degli stadi dello sviluppo delle capacità di imitazione nei bambini.

L'apprendimento per dimostrazione è attualmente un argomento discusso regolarmente nelle due maggiori conferenze di robotica IROS e ICRA, ed in alcune conferenze minori.

2.1.2 Definizione del problema

I metodi di apprendimento per dimostrazione possono essere inquadrati nell'ambito dell'apprendimento supervisionato, in quanto il robot apprende la strategia tramite l'insieme di dimostrazioni del compito desiderato fornite dall'insegnante.

Formalmente un problema di apprendimento per dimostrazione viene definito nel modo seguente. Il mondo è costituito da una serie di stati S ed azioni A, ed il passaggio tra gli stati avviene per mezzo delle azioni. Tipicamente lo stato non è completamente osservabile e quindi il robot alunno ha accesso solo ad una parte di esso. L'insieme degli stati disponibili al robot viene indicato con Z, ed è ricavabile tramite una trasformazione

$M : S \rightarrow Z$. Il problema dell'apprendimento per dimostrazione consiste nello stabilire la strategia $\pi : Z \rightarrow A$ in grado di selezionare le azioni da effettuare sulla base degli stati osservati. In generale la strategia può essere funzione di diversi parametri ovvero, $\pi = \pi(z(t), t, \alpha)$ in cui $z(t) \in Z$ è il vettore dello stato osservato al tempo t, t è l'istante di tempo considerato ed α è il vettore di parametri non noti che devono essere aggiustati durante l'apprendimento (ad esempio i pesi di una rete neurale). Nel caso la dipendenza dal tempo non sia esplicitamente presente si ha $\pi = \pi(z(t), \alpha)$.

Per ricavare la strategia π vengono sfruttate le dimostrazioni, durante l'esecuzione delle quali da parte dell'insegnante vengono memorizzati gli stati e le azioni corrispondentemente intraprese. Indicato con D l'insieme delle dimostrazioni, la $i - esima$ dimostrazione $d_i \in D$ è rappresentata dall'insieme delle k coppie di stati osservati e azioni, ovvero $d_i = (z_i^j, a_i^j)$, con $z_i^j \in Z, a_i^j \in A, j = 0...k$. La strategia derivata da queste dimostrazioni deve consentire al robot di selezionare l'azione appropriata sulla base dello stato corrente.

2.2 Dimostrazione

Alla base dei metodi di apprendimento per dimostrazione esiste sempre una fase di dimostrazione in cui il robot acquisisce l'insieme di esempi da cui derivare la strategia da utilizzare. Tale fase può essere realizzata in modalità interattiva o non interattiva: nel primo caso la strategia viene aggiornata in modo incrementale man mano che vengono fornite le dimostrazioni, nel secondo caso invece la strategia viene appresa solo dopo che sia stata conclusa la fase di dimostrazione.

In questa fase le decisioni chiave che devono essere prese da un eventuale sviluppatore di un nuovo metodo di apprendimento per dimostrazione, sono principalmente due: la scelta della modalità con cui effettuare le dimostrazioni (v. paragrafo 2.2.1) e la scelta di cosa imitare (v. paragrafo 2.2.2).

2.2.1 Come effettuare le dimostrazioni

La maggior parte dei lavori proposti in letteratura si dividono in due categorie a seconda di come vengono effettuate le dimostrazioni. In questo paragrafo vengono illustrate le principali modalità con cui è possibile effettuare le dimostrazioni e i pregi e difetti di ciascuna di esse.

Muovere il corpo del robot alunno

L'opzione più semplice da utilizzare è quella di muovere direttamente il corpo del robot alunno, in modo tale che il robot sia in grado di memorizzare gli stati e le azioni esperiti durante la sua stessa esecuzione della dimostrazione, senza dover risolvere nessun problema di corrispondenze tra alunno e dimostratore.

Un primo metodo tramite cui muovere il corpo del robot è quello della teleoperazione: il robot viene guidato tramite joystick o tastiera dall'insegnante nell'esecuzione del compito. Nonostante la teleoperazione fornisca il metodo più semplice e diretto per trasferire l'informazione durante la fase di dimostrazione, essa tipicamente non viene utilizzata quando si ha a che fare con robot che presentano un gran numero di gradi di libertà, come ad esempio i robot umanoidi. La motivazione di ciò sta nel fatto che in questi casi muovere il robot tramite joystick per dimostrare movimenti di basso livello può diventare piuttosto complicato e difficile da gestire.

Una variante della teleoperazione tradizionale è costituita dal *kinesthetic teaching*. Il *kinesthetic teaching* consiste nel controllare passivamente i giunti del robot, guidandoli manualmente nei movimenti verso configurazioni desiderate e si differenzia dalla teleoperazione in cui i giunti vengono guidati attivamente tramite joystick o tastiere.

L'utilizzo di insegnanti umani permette inoltre di avvalersi anche di altre tecniche, tra cui l'uso del linguaggio naturale. Nel caso in cui la dimostrazione venga mediata dal linguaggio, come accade in questa tesi, l'insegnante ha il compito di dire al

robot quali comportamenti elementari eseguire nei diversi stati al fine di completare il compito desiderato. L'utilizzo del linguaggio ha lo svantaggio di richiedere che il sistema possegga una serie di comportamenti elementari preesistenti, detti tipicamente primitive, e permette solo lo sviluppo di comportamenti ottenibili a partire dalla combinazione di tali primitive.

Osservare il comportamento del dimostratore

In questi approcci la dimostrazione è eseguita da una piattaforma diversa da quella del robot alunno, sia che essa sia un essere umano o un altro robot. In questo caso, poiché tipicamente il robot alunno e il dimostratore differiscono a livello di sensori e struttura fisica, è necessario risolvere il cosiddetto problema delle corrispondenze, ovvero ricavare la trasformazione tra lo spazio dell'insegnante e quello dell'alunno che permetta di trasferire le informazioni dall'uno all'altro [8],[12].

In questo contesto si collocano gli approcci che vedono i sensori localizzati direttamente sul corpo dell'insegnate in modo da memorizzarne l'esecuzione delle dimostrazioni e quelli che invece utilizzano sensori esterni alla piattaforma che esegue le dimostrazioni, non necessariamente localizzati sul robot alunno.

Nel primo caso, nonostante il vantaggio di poter fornire delle misure precise degli esempi eseguiti, spesso sono richiesti costi elevati in termini di sensori specialistici e strutture apposite. Infatti tipicamente sul corpo dell'insegnante umano che esegue le dimostrazioni vengono applicati dei sensori al fine di memorizzare le informazioni utili delle dimostrazioni. Tali metodi possono risultare vantaggiosi specialmente quando si lavora con robot umanoidi o antropomorfi, in virtù della somiglianza tra il corpo del robot e quello dell'essere umano.

Per quanto concerne invece gli approcci che si basano sull'osservazione del comportamento dimostrato (per esempio attraverso l'utilizzo di una telecamera), poiché il robot non memorizza direttamente gli stati e le azioni esperiti dall'insegnante durante

l'esecuzione, questi devono essere dedotti introducendo di conseguenza una potente sorgente di incertezza per il robot alunno. In confronto all'approccio precedentemente illustrato, i dati memorizzati con questa tecnica sono meno precisi e meno affidabili ma il metodo risulta più generale ed economico.

2.2.2 Cosa imitare

Una volta stabilito come effettuare le dimostrazioni, occorre scegliere cosa imitare del compito dimostrato. In questo contesto la letteratura si scinde fondamentalmente in due grandi categorie: le tecniche che si concentrano sull'apprendimento dei micromovimenti e quindi delle traiettorie e i metodi che invece si focalizzano sull'apprendimento dei comportamenti elementari. Per maggiore chiarezza da ora in avanti si indicherà con il termine di azione o micromovimento un singolo comando agli attuatori del robot, mentre con il termine di comportamento elementare una sequenza completa di azioni finalizzate ad un certo scopo, come ad esempio afferrare un oggetto con la mano o raggiungere un punto dello spazio peripersonale. Si noti che la scelta di cosa imitare influenza le modalità con cui codificare gli stati e le azioni esperite durante le dimostrazioni e di conseguenza gli algoritmi di apprendimento che possono essere utilizzati.

2.2.2.1 Micromovimenti

Nel caso si vogliano riprodurre i micromovimenti dimostrati, diventa cruciale la scelta delle variabili con le quali codificare un particolare movimento e spesso si rendono necessarie tecniche di riduzione della dimensionalità dello spazio che proiettino i segnali registrati in uno spazio di moto fittizio di dimensionalità ridotta, al fine di ridurre il carico computazionale del problema (per un esempio di tali tecniche cfr. [13]).

Inoltre, poiché si tiene traccia dei micromovimenti che compongono il compito e poiché le ripetizioni dello stesso compito da parte di un dimostratore umano danno

luogo a micromovimenti che potrebbero non essere gli stessi eseguiti precedentemente, le informazioni memorizzate relative a tutte le dimostrazioni potrebbero essere ambigue e contrastanti. Per gestire tale variabilità presente nelle diverse dimostrazioni dello stesso compito sono stati utilizzati diversi metodi tra cui alcuni comprendenti tecniche di apprendimento statistico ed altri basati su equazioni differenziali.

La maggior parte dei lavori realizzati in questo contesto codifica i movimenti del dimostratore nello spazio dei giunti, memorizzando ad esempio le traiettorie di alcune delle variabili di giunto umane, o nello spazio cartesiano, memorizzando ad esempio la traiettoria tridimensionale dell'end-effector o di altre parti del sistema.

Uno dei primi lavori che si sono occupati della riproduzione dei micromovimenti è quello di Ijspeer et al. [14], che hanno progettato un sistema per la codifica e la riproduzione dei movimenti a partire da diverse condizioni iniziali, basato su equazioni differenziali. Il sistema d'apprendimento in questione sfrutta sensori applicati sull'insegnante per memorizzare le informazioni relative alle dimostrazioni e si ripropone di apprendere le traiettorie delle variabili nello spazio dei giunti. L'essenza dell'approccio è quella di partire da un semplice sistema dinamico parametrico, cioè un insieme di equazioni differenziali lineari, e di trasformarlo successivamente in un sistema non lineare con dinamiche d'attrazione generate per mezzo di termini di forzamento ricavati in seguito a un processo di apprendimento. Per apprendere i parametri del sistema appena descritto Ijspeer et al. adottano la regressione pesata locale (*Locally weighted regression*, LWR), che combina la semplicità della regressione lineare ai minimi quadrati e la flessibilità della regressione non lineare. L'efficacia di questo metodo di apprendimento per dimostrazione è stata dimostrata su un robot umanoide che ha appreso una serie di movimenti tipici del tennis tra cui il dritto e il rovescio (Figura 2.1).

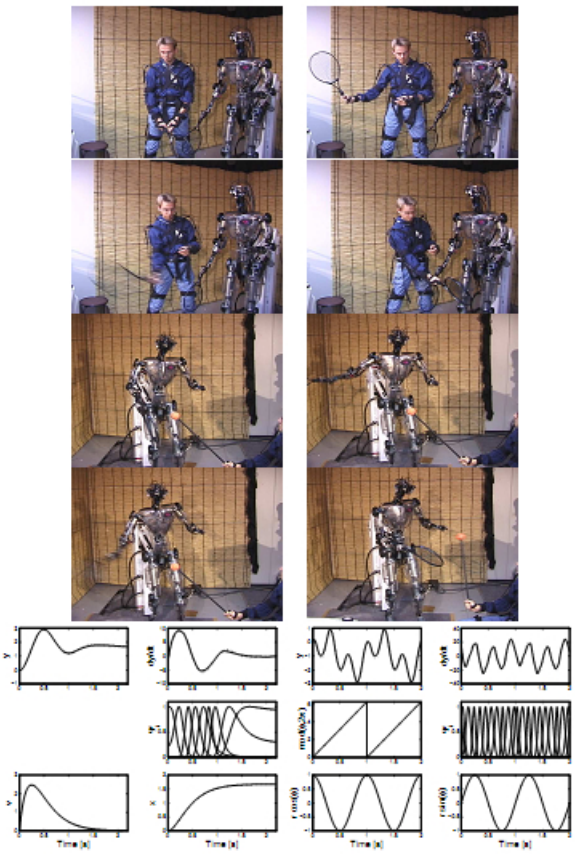

Figura 2.1: In alto: sequenza di immagini che descrivono l'apprendimento di un dritto a partire dalla dimostrazione fornita dall'insegnante umano. In basso: esempi di traiettorie per la riproduzione di movimenti discreti (sinistra) e periodici (destra).

2.2.2.2 Comportamenti elementari

In questa classe ricadono tutti quegli approcci che hanno l'obiettivo di apprendere comportamenti formati dalla combinazione di comportamenti elementari. Tali comportamenti elementari, o primitive motorie, possono essere preesistenti nel sistema [7],[15] o ricavabili tramite segmentazione del flusso di dati acquisiti durante la fase di dimostrazione, come accade ad esempio in [16].

A livello concettuale, nella Figura 2.2 vengono illustrati gli elementi più importanti

di un sistema di imitazione basato sull'esistenza di primitive motorie. Le informazioni

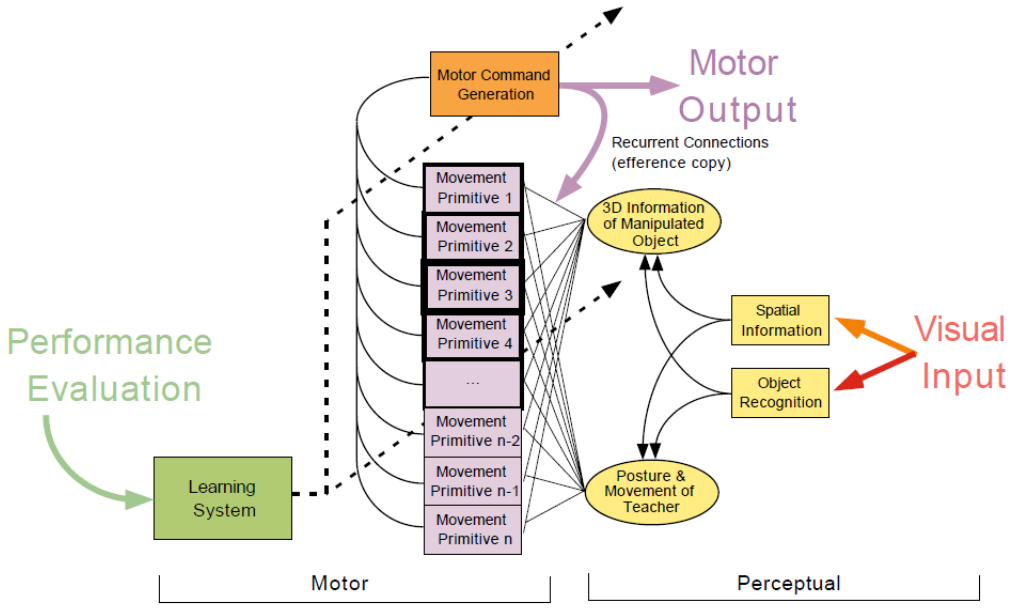

Figura 2.2: Schema concettuale di un sistema di imitazione [6],[7]. La parte destra della figura illustra gli elementi percettivi del sistema ed indica come le informazioni visive vengono trasformate in informazioni spaziali e relative al riconoscimento di oggetti; la parte destra invece si focalizza sugli elementi del controllo motorio, alla base del quale esistono una serie di primitive motorie preesistenti, in lilla.

visive devono essere trasformate in informazioni spaziali in un sistema di coordinate scelto per essere poi utilizzate per decidere quali primitive motorie attivare. La Figura assume che le azioni dell'insegnante percepite dal sistema vengano mappate in un insieme di primitive motorie preesistenti in una fase preliminare di assimilazione, e che queste siano attivate in una fase successiva, al fine di riprodurre il compito dimostrato.

In questo contesto si collocano i lavori di Zhang e Weng in [1] e Dominay et al. in [2], in cui l'obiettivo è quello di insegnare ad un robot comportamenti complessi a partire dalla combinazione di una serie di comportamenti elementari. In particolare, in [2] il dimostratore utilizza dei comandi vocali appositi per indicare l'inizio e la fine di un

nuovo compito ed il robot si limita a memorizzare e a riprodurre la sequenza di primitive dimostrate. Il sistema proposto in questo lavoro consente ad un robot di apprendere una serie di comportamenti utili nella costruzione di un tavolino. In [1] invece, viene presentato un modello tramite cui un robot possa imparare ad associare una sequenza di comportamenti elementari precedentemente appresi ad un nuovo comando linguistico tramite un sistema di predizione in grado di associare al contesto senso-motorio corrente un contesto senso-motorio futuro. In realtà però anche in questo lavoro l'esperimento effettivamente condotto sul robot è piuttosto semplice e limitato, in quanto tiene conto solo degli stimoli sensoriali acustici e non di quelli relativi all'interazione del robot con l'ambiente durante l'esecuzione delle azioni, dando luogo ad un sistema che analogamente a quello di Dominay et al. di fatto si basa solo sulla riproduzione della sequenza di primitive. Il robot in questo caso impara a disegnare un fiore (v. Figura 2.3), associando alla nuova parola "start" la sequenza dei quattro comportamenti elementari precedentemente appresi, ciascuno relativo al disegno di uno dei quattro petali del fiore.

Figura 2.3: Il robot SAIL mentre riproduce il compito appreso.

2.3 Apprendimento

La scelta di un algoritmo per generare la strategia in grado di associare allo stato corrente l'azione corrispondente è tipicamente influenzata dalla codifica scelta per lo spazio degli stati e delle azioni e dalla presenza o meno della possibilità di migliorare le prestazioni del sistema ottimizzando i comportamenti dimostrati. Infatti, poiché in molti casi è possibile che le prestazioni dell'insegnante siano inferiori se comparate

con le abilità dell'alunno, molti sistemi di apprendimento per dimostrazione tengono in considerazione la possibilità di ottimizzare i comportamenti appresi, migliorando le capacità di riproduzione del robot tramite ad esempio tecniche di apprendimento incrementale.

Le principali tecniche tramite cui ricavare la strategia da utilizzare vengono brevemente illustrate di seguito e comprendono fondamentalmente l'utilizzo di reti neurali artificiali apprese con algoritmi di apprendimento supervisionato, equazioni differenziali e modelli di apprendimento statistici.

- **Reti neurali artificiali e apprendimento supervisionato.** Le reti neurali artificiali costituiscono un potente mezzo tramite cui derivare la strategia in grado di associare allo stato corrente l'azione o il comportamento elementare da eseguire ma hanno lo svantaggio di richiedere lunghi tempi di convergenza. Le dimostrazioni in questo caso vengono utilizzate per apprendere i pesi delle connessioni tra i neuroni tramite algoritmi di apprendimento supervisionato, quali *backpropagation* e *backpropagation through time*[1]. In questo contesto si colloca ad esempio il lavoro di Ito et al. [20] in cui viene addestrata una rete neurale ricorrente con una variante dell'algoritmo di *backpropagation through time*. In questo caso le dimostrazioni vengono effettuate tramite *kinesthetic teaching* e l'obiettivo è quello di insegnare ad un piccolo robot umanoide alcuni comportamenti di manipolazione di oggetti.

- **Equazioni differenziali.** Le equazioni differenziali vengono tipicamente utilizzate per apprendere i micromovimenti dimostrati e forniscono spesso soluzioni robuste ai cambiamenti dinamici che avvengono nell'ambiente. In questo contesto si collocano i lavori proposti da Schaal et al., tra cui il già citato [14], in cui vengono utilizzati sistemi di equazioni differenziali parametrici i cui parametri

[1] Per maggiori informazioni cfr. [19].

vengono appresi tramite l'utilizzo delle dimostrazioni.

Sistemi di equazioni differenziali possono inoltre essere utilizzati per ricavare le primitive motorie, come proposto in [21].

- **Modelli di apprendimento statistici.** In questo contesto si collocano le tecniche di apprendimento statistico che comprendono approcci di approssimazione tramite spline, tecniche basate sulla generazione di traiettorie generalizzate ottenute semplicemente calcolando valori medi e varianze delle variabili memorizzate per ogni unità di tempo in ogni dimostrazione, tecniche probabilistiche che fanno uso di *Gaussian Mixture Model* (GMM), e tecniche che utilizzano i modelli di Markov nascosti[2] nei quali le dimostrazioni vengono utilizzate per calcolare le probabilità di transizione tra gli stati [16], [23].

2.4 Limitazioni dell'apprendimento per dimostrazione

I sistemi di apprendimento per dimostrazione sono strettamente collegati alle informazioni fornite tramite le dimostrazioni e di conseguenza le loro prestazioni sono pesantemente influenzate dalla quantità e dalla qualità di tali informazioni.

Le principali motivazioni connesse a basse prestazioni dell'alunno in sistemi d'apprendimento per dimostrazione sono fondamentalmente due: da una parte l'esistenza di aree dello spazio che non sono state dimostrate, dall'altra la qualità degli esempi dimostrati, dipendente dalla capacità dell'insegnante di eseguire il compito ottimamente.

Il fatto che l'insegnante non possa dimostrare l'azione corretta per ogni possibile stato, data la vastità dello spazio degli stati, pone di fronte al problema di decidere come il robot dovrebbe agire quando si trova di fronte a stati mai esperiti durante le

[2]Per maggiori informazioni sui Modelli di Markov applicati all'apprendimento si può far riferimento a [22].

dimostrazioni. In questi casi è possibile seguire due strade: generalizzare a partire dalle dimostrazioni fornite, o acquisire nuove informazioni tramite dimostrazioni aggiuntive.

Il fatto che le dimostrazioni fornite dall'insegnante possano essere ambigue e non ottimali influenza pesantemente la qualità della strategia appresa. In questo contesto l'ambiguità nasce quando ci sono più azioni che possono essere elicitate in corrispondenza dello stesso stato, tra le quali l'insegnante sceglie arbitrariamente o quando non è possibile identificare correttamente lo stato a causa di rumore dei sensori o a causa del fatto che le variabili rilevanti del compito non sono direttamente osservabili. Il risultato in entrambi i casi è che stati identici, o praticamente identici, sono associati ad azioni o comportamenti elementari differenti. Anche in questo caso, in cui il robot non è in grado di stabilire con certezza quale azione scegliere in corrispondenza di uno degli stati esperiti durante le dimostrazioni precedenti, può essere tenuta in considerazione la possibilità di richiedere all'insegnante nuove dimostrazioni del compito. Un approccio alternativo per gestire la bassa qualità delle informazioni derivanti dalle dimostrazioni prevede che lo studente possa apprendere o raffinare le proprie capacità attraverso un processo di apprendimento per prove ed errori basato sul raggiungimento di un obiettivo dato, come ad esempio avviene in [10].

Capitolo 3

Modello Sperimentale

Questo Capitolo descrive il modello sviluppato ed è strutturato essenzialmente in due parti. Nella prima parte vengono presentati gli strumenti utilizzati in questa tesi: il robot iCub, il simulatore e lo scenario sperimentale che si è scelto di utilizzare. Nella seconda parte del Capitolo, che ha inizio a partire dal paragrafo 3.3, vengono invece descritti il sistema di controllo del robot, composto dalle primitive motorie implementate e da un controllore neurale, e le fasi del processo di apprendimento del controllore neurale stesso. In accordo con i metodi di apprendimento per dimostrazione, il processo di apprendimento è organizzato fondamentalmente in due fasi. Nella prima fase, descritta nella sezione 3.4, il dimostratore produce dei comandi linguistici costituiti dai nomi delle diverse primitive motorie ed il robot si limita ad eseguirli attivando le primitive motorie ad essi corrispondenti. In questa fase gli stati sensoriali esperiti dal robot durante l'esecuzione delle dimostrazioni e il tipo di primitive eseguite in ogni unità di tempo vengono utilizzati per generare un training set, necessario per procedere nella fase successiva con l'apprendimento supervisionato del sistema. Nella seconda fase infatti, come descritto nella sezione 3.5, il training set viene utilizzato per addestrare una rete neurale ad associare agli stati sensoriali correnti le primitive motorie corrispondenti. Alla fine del processo di apprendimento viene verificata la capacità

del robot di esibire un comportamento simile a quello dimostrato autonomamente, in assenza cioè dei comandi linguistici prodotti dal dimostratore (cfr. sezione 3.6).

Il sistema oggetto di questo studio ricade nell'ambito dei sistemi d'apprendimento per dimostrazione mediati dal linguaggio, in quanto nella fase di dimostrazione il robot viene guidato dall'utente tramite opportuni comandi "linguistici". Analogamente ai lavori di Zhang e Weng in [1] e Dominay et al. in [2], l'obiettivo è quello di insegnare ad un robot ad eseguire comportamenti complessi combinando in sequenza una serie di comportamenti elementari. Tuttavia, mentre nei lavori sopra citati il processo di apprendimento riguarda semplicemente la capacità di combinare una serie di comportamenti elementari in sequenza indipendentemente dal contesto senso-motorio, per il sistema presentato in questa tesi il processo di apprendimento riguarda la capacità di associare dei comportamenti elementari direttamente agli stati sensoriali esperiti e di conseguenza la capacità di determinare quando devono avvenire le transizioni tra le primitive sulla base del contesto senso-motorio. In altre parole il sistema di apprendimento per dimostrazione presentato ha lo scopo di consentire ad un robot umanoide, dotato di una serie di primitive motorie preesistenti, di apprendere il compito dimostrato acquisendo la capacità di elicitare la primitiva corretta sulla base dello stato sensoriale corrente, ed eventualmente, degli stati sensoriali precedenti.

3.1 L'iCub

Il robot utilizzato per sviluppare questo lavoro è l'iCub (v. Figura 3.3), un robot umanoide sviluppato dall'Istituto Italiano di Tecnologia (IIT) e da altri laboratori nell'ambito del progetto europeo RobotCub. La motivazione principale che ha portato alla realizzazione di questa piattaforma informatica e relativo hardware è stata quella di sviluppare una piattaforma avanzata comune per la ricerca nella robotica umanoide con un'enfasi sulla robotica *developmental* [24], ovvero che cerca sviluppare robot che

acquisiscano le proprie capacità attraverso un processo di apprendimento simile a quello umano.

Il progetto dell'iCub è *open-source* sia per quanto riguarda il software che per la parte hardware, approfonditamente descritta nelle sue componenti e con pezzi reperibili sul mercato.

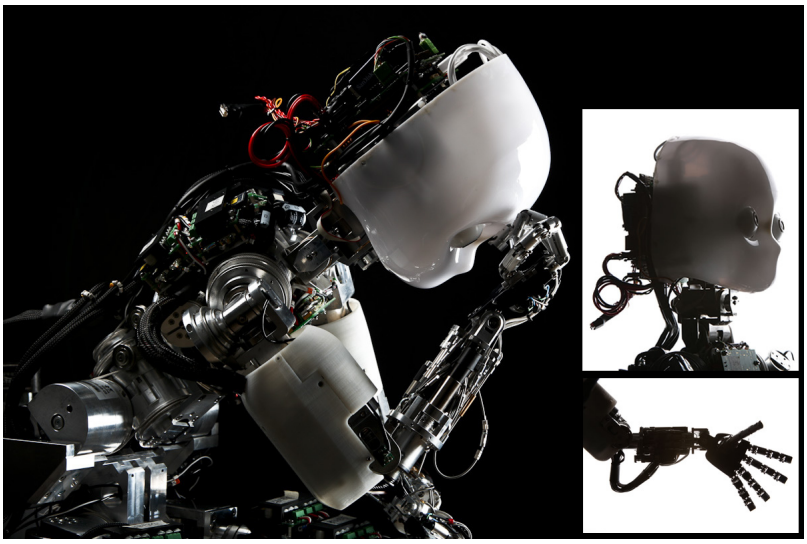

Figura 3.1: iCub

3.1.1 Specifiche meccaniche ed elettroniche

Alto $104cm$ e pesante $22kg$, l'iCub presenta le sembianze di un bambino di circa 3 anni. Il robot è controllato da un controllore $PC104$ a bordo, che comunica con gli attuatori e i sensori usando lo standard seriale CANbus. L'iCub, che sin dalla sua prima realizzazione è stato sottoposto a numerose revisioni e miglioramenti, presenta nella sua versione finale una struttura comprendente complessivamente 53 gradi libertà, distribuiti nel modo seguente:

- 7 gradi di libertà per ogni braccio: 3 per la spalla, 1 per il gomito e 3 per il polso;

- 9 gradi di libertà per ogni mano: 3 per il pollice, 2 per l'indice, 2 per il dito medio, 1 per l'anulare e il mignolo e 1 per l'abduzione/adduzione delle dita;
- 6 gradi di libertà per i movimenti della testa;
- 3 gradi di libertà per il torace e la colonna vertebrale;
- 6 gradi di libertà per ogni gamba.

Gli angoli di giunto sono misurati tramite l'utilizzo di encoder incrementali e sensori realizzati appositamente, basati sull'effetto Hall. Attualmente le dita possono essere equipaggiate con sensori di tatto e in futuro, quando sarà realizzata la pelle artificiale attualmente in fase di studio, potranno essere rivestite da tale pelle ed acquistare capacità sensoriali distribuite con un miglioramento della capacità percettiva complessiva. La testa possiede un sistema di due telecamere orientabili per la visione stereo, posizionate al posto degli occhi e due microfoni posizionati lateralmente al posto delle orecchie. Alcuni LED rossi montati dietro il pannello del volto consentono di simulare alcune espressioni facciali.

3.1.2 Architettura software

Il software dell'iCub è fondamentalmente scritto in linguaggio C++ e usa il protocollo di comunicazione YARP per interfacciarsi tramite rete Gbit Ethernet con software fuori bordo, necessario per un controllo di più alto livello. YARP[1] (*Yet Another Robot Platform*) è una struttura open-source multi-piattaforma che supporta il calcolo distribuito orientato al controllo di robot. Per spiegare concettualmente il funzionamento di YARP si può far riferimento alla Figura 3.2. Come si evince dalla Figura esiste un server YARP che consente di gestire la comunicazione tra i processi che girano sul robot, sia che esso sia reale o simulato, e i processi relativi al software di

[1] Cfr. http://eris.liralab.it/yarp/ e [26]

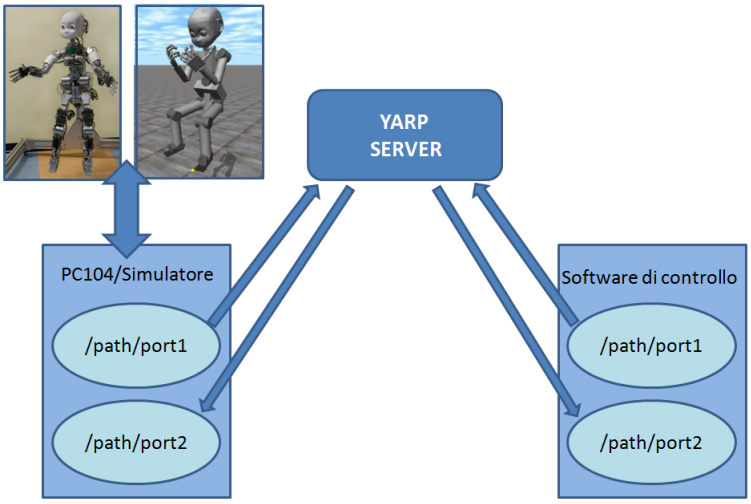

Figura 3.2: Funzionamento concettuale di YARP.

controllo implementato. La comunicazione avviene tramite porte che consentono l'invio e la ricezione di messaggi che tipicamente contengono le letture dei sensori e i comandi per gli attuatori.

La maggior parte degli esperimenti condotti in questa tesi è stata effettuata in simulazione, utilizzando il simulatore open source dell'iCub, sviluppato da Gianluca Massera. Tale software permette di creare un "mondo" tridimensionale virtuale nel quale il modello dell'iCub può eseguire le azioni dettate dal proprio controllore interagendo con l'ambiente, e consente di utilizzare YARP come infrastruttura per la comunicazione in modo da poter utilizzare lo stesso software sviluppato in simulazione per gli esperimenti sull'iCub reale.

Il linguaggio di programmazione mediante il quale viene progettato il controllore per la movimentazione del robot nel mondo è il C++; nel caso in esame è stato utilizzato il software *Microsoft Visual Studio 2008*. Parte integrante del simulatore sono le librerie *Newton Dynamics*[2] che consentono di simulare accuratamente le dinamiche e

[2]Cfr. http://newtondynamics.com

le collisioni tra corpi rigidi. Ulteriori librerie che sono state utilizzate per la realizzazione del controllore sono:

- librerie YARP: utilizzate per interfacciarsi con i dispositivi hardware del robot e per la comunicazione tra i processi;
- librerie QT[3]: utilizzate per gestire la grafica del simulatore, le strutture dati dinamiche, i processi *multithread* e la comunicazione tramite slot e segnali nel codice;
- librerie NNFW[4]: utilizzate per implementare le reti neurali e l'algoritmo di apprendimento supervisionato.

3.2 Scenario Sperimentale

Lo scenario sperimentale che si è scelto di utilizzare prevede che il robot sia posto di fronte ad un tavolo, sul quale viene posizionato un oggetto di forma variabile tra cubo, cilindro o sfera in posizione nota e un cestino a terra alla destra del robot stesso. Tale

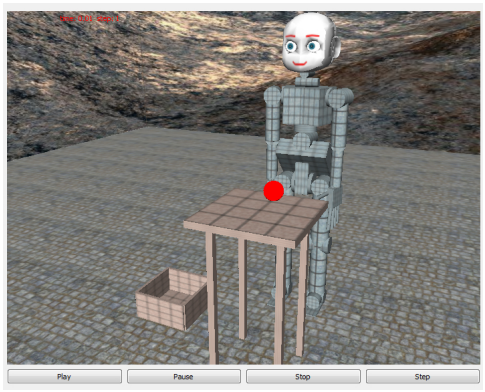

Figura 3.3: Scenario sperimentale.

[3]Cfr. http://qt.nokia.com/
[4]Cfr. http://www.nnfw.org/index.php

scenario è stato pensato per consentire la realizzazione di compiti di manipolazione di oggetti, e per tale motivo sono stati utilizzati i sensori propriocettivi del robot che restituiscono le posizioni dei giunti del braccio, della mano e i contatti della mano con gli oggetti presenti nell'ambiente.

La scelta di uno scenario piuttosto semplice è legata al fatto che l'interesse principale di questo lavoro si focalizza sul processo di apprendimento del robot, piuttosto che sulla complessità dei compiti realizzabili.

3.3 Il sistema di controllo del robot

In questa sezione viene descritto nel dettaglio il sistema di controllo del robot.
Come è stato già detto, l'obiettivo di questa tesi è quello di progettare un sistema in grado di permettere ad un robot umanoide, dotato di una serie di capacità comportamentali elementari, di sviluppare attraverso un processo di apprendimento guidato dall'utente nuove capacità ottenute combinando in sequenza o in parallelo le capacità preesistenti, acquisendo la capacità di elicitare il comportamento appropriato al contesto corrente. Per perseguire questo obiettivo si è quindi reso necessario:

- da una parte identificare e implementare una serie di primitive motorie che consentano al robot di esibire un certo numero di comportamenti elementari e che utilizzate in combinazione tra loro possano consentire al robot di esibire potenzialmente un'ampia varietà di comportamenti complessi;

- dall'altra sviluppare un'architettura e un algoritmo di apprendimento che consentano al robot di acquisire la capacità di esibire una serie di comportamenti complessi scelti arbitrariamente dall'utente nello spazio dei comportamenti generabili a partire dalle primitive implementate.

In questo ambito con il termine di comportamenti "complessi" si intende l'insieme di comportamenti che risultano dalla composizione di comportamenti elementari, tipicamente indicati con il termine di primitive.

A tale proposito vengono illustrate le primitive motorie implementate e successivamente proposte due architetture alternative di controllore neurale che si differenziano tra loro per la codifica delle primitive motorie utilizzata: localistica (cfr. paragrafo 3.3.2.1) o distribuita (cfr. paragrafo 3.3.2.2).

Per capire il funzionamento del sistema di controllo si può far riferimento alla Figura esplicativa 3.4. Come si evince dalla Figura, il controllore neurale prende in ingresso

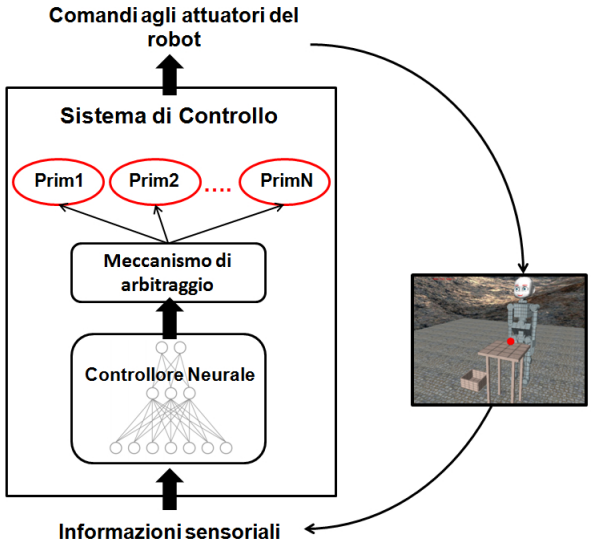

Figura 3.4: Struttura del sistema di controllo.

le informazioni sensoriali provenienti dal robot e fornisce in uscita i valori in base ai quali un meccanismo di arbitraggio opportunamente realizzato stabilisce quale primitiva attivare. In questo contesto, con meccanismo di arbitraggio si intende la parte di codice che definisce le modalità di attivazione delle primitive: a seguire nel testo verranno descritte le varie modalità oggetto di sperimentazione (cfr. sezione 3.6, paragrafo

4.1.1.3 e 4.2.1). Le primitive motorie, raffigurate nei moduli evidenziati in rosso, sono in generale N e costituiscono i blocchi base per la costruzione di comportamenti complessi, in quanto individuano i comportamenti elementari a disposizione del robot. L'attivazione di una o più delle primitive in questione fornisce in uscita i comandi per gli attuatori del robot che consentono al robot stesso di interagire con l'ambiente modificando di conseguenza il contesto senso-motorio.

3.3.1 Le primitive motorie

Per motivi analoghi a quelli che hanno portato alla scelta dello scenario sperimentale, si è scelto di implementare delle semplici primitive posturali, ovvero delle routine che modificano la posizione di una serie di giunti fino ad ottenere una postura desiderata. Nell'ipotesi di utilizzare l'iCub per compiti di manipolazione di oggetti, sono state implementate le seguenti cinque primitive motorie:

1. **REACH** : spostamento del braccio destro del robot da una posizione iniziale ad una desiderata in corrispondenza di un oggetto posto sul tavolino;
2. **CLOSE-HAND**: chiusura delle dita della mano destra del robot;
3. **LIFT**: spostamento del braccio destro del robot da una posizione iniziale ad una desiderata in corrispondenza del cesto posizionato a terra;
4. **OPEN-HAND**: apertura delle dita della mano destra del robot;
5. **MOVE**: spostamento del braccio destro del robot da una posizione iniziale ad una desiderata coincidente con la posizione di partenza del braccio.

Tali primitive sono state implementate tramite un semplice controllo proporzionale in velocità, basato sull'errore tra la posizione iniziale acquisita tramite sensori e la posizione desiderata nota. Per ricavare la posizione desiderata è stato usato il modulo *RobotMotorGui.exe*, fornito col simulatore open source dell'iCub sviluppato all'IIT di

Genova, tramite il quale è possibile agire manualmente sui singoli valori delle variabili di giunto per spostare il robot.

3.3.2 Il controllore neurale

Parte integrante del sistema di controllo del robot è costituita dal controllore neurale, ovvero da una rete neurale che riceve in ingresso lo stato corrente di alcuni dei sensori del robot e la posizione in coordinate cartesiane del palmo della mano destra e produce in uscita dei valori che determinano la primitiva o le primitive motorie che devono essere attivate. In Figura 3.5 vengono riportate le due architetture scelte per

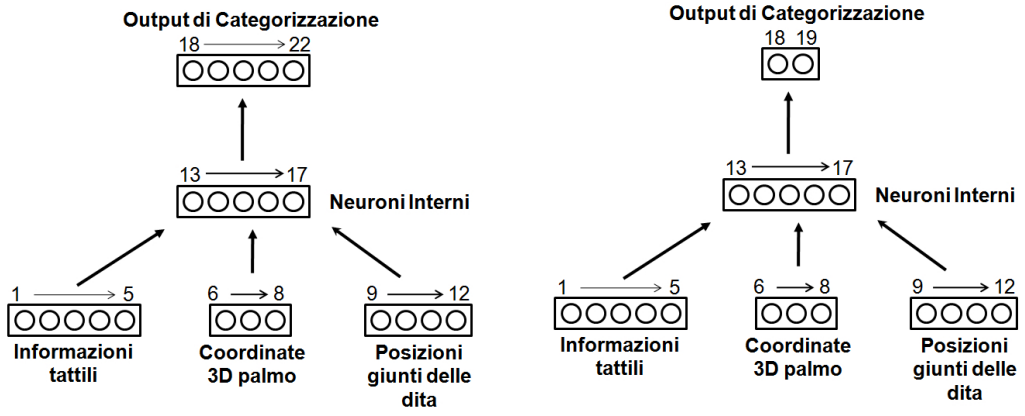

Figura 3.5: A sinistra: Architettura 1 del controllore neurale. A destra: Architettura 2 del controllore neurale. I blocchi evidenziati rappresentano i gruppi di neuroni che hanno le stesse caratteristiche; le frecce indicano le connessioni tra i blocchi: ciascun neurone del blocco di partenza è connesso con tutti i neuroni del blocco di arrivo.

l'implementazione del controllore. Per comodità, nel seguito l'architettura di sinistra verrà indicata con il nome di Architettura 1, mentre quella di destra con il nome di Architettura 2. Come si evince dalla Figura, entrambe le reti neurali sono costituite da 12 neuroni di ingresso e 5 neuroni interni, e l'unica differenza tra le due consiste nella

scelta del numero di neuroni di uscita e quindi nella modalità con cui sono codificate le primitive da attivare.

Ad ogni unità di tempo la risposta y_i del neurone di ingresso $i - esimo$ viene aggiornata nel modo seguente:

- per $i = 1, .., 5$, y_i è aggiornata sulla base dello stato dei sensori tattili distribuiti sulle dita della mano e scalata linearmente nell'intervallo $[0, 1]$. I sensori di tatto restituiscono il valore 1 se la parte della mano su cui sono applicati è in contatto con un altro corpo rigido, altrimenti danno 0. In questo caso si è scelto di associare un neurone a ciascun dito della mano in modo tale da tenere traccia della somma dei contatti di ciascun dito con un altro corpo rigido;

- per $i = 6, .., 8$, y_i è aggiornata sulla base della posizione del palmo della mano destra in coordinate cartesiane, nel sistema di riferimento del robot, normalizzate tra 0 e 1 in base allo spazio che il palmo stesso può raggiungere mantenendo il robot fermo;

- per $i = 9, .., 12$, y_i è aggiornata sulla base dello stato dei sensori propriocettivi della mano che codificano l'estensione/flessione relativa alle 5 dita, ovvero sulla base delle posizioni del primo giunto di ciascun dito. Tali valori sono scalati linearmente nell'intervallo $[0, 1]$ in modo che lo 0 corrisponda al dito completamente steso ed 1 corrisponda al dito completamente piegato. Come si può notare, in realtà il numero di neuroni appartenente a questo blocco è 4 anzichè 5 in quanto l'anulare e il mignolo sono comandati tramite un unico giunto.

Ad ogni unità di tempo la risposta y_i dei neuroni interni e di uscita viene aggiornata sulla base della relazione (1.2), in cui $\theta_i = 0$ e la funzione d'attivazione $\Phi(\cdot)$ è la funzione logistica in (1.5). I valori di k della funzione logistica in (1.5), che come è stato detto determinano l'inclinazione della curva, sono stati scelti in modo che la pendenza della stessa sia tale da distinguere significanti variazioni del net input producendo di

conseguenza uscite differenti. Nello specifico, per i neuroni interni è stato scelto $k = 0.1$, mentre per i neuroni di uscita $k = 0.2$. In particolare, per effettuare tale scelta è stato tenuto in considerazione l'intervallo di inizializzazione dei pesi delle reti neurali, scelto pari a $[-6, 6]$. Il fatto che il range di inizializzazione dei pesi scelto sia abbastanza grande riflette d'altra parte il ricorso a funzioni logistiche con minori caratteristiche di non linearità (valori di k più bassi, Figura 1.5). La scelta del numero dei neuroni interni è stata presa a seguito di numerose prove in cui tale numero è stato aumentato progressivamente.

Come si evince dalla Figura 3.5, in entrambe le reti non vi sono né connessioni dirette tra i neuroni di ingresso e i neuroni di uscita né connessioni ricorrenti: le reti neurali risultano puramente reattive, reagiscono cioè sempre allo stesso modo in corrispondenza di un determinato stimolo, indipendentemente dagli stimoli precedenti.

In entrambe le architetture i neuroni di uscita codificano gli output di categorizzazione, ovvero i valori in base ai quali decidere le primitive che devono essere attivate a seconda del contesto corrente. In base al numero di neuroni di uscita del controllore neurale sono stati adottati due tipi di codifiche differenti: una codifica localistica (Architettura 1) e una codifica distribuita (Architettura 2).

3.3.2.1 Codifica localistica

Nel caso dell'Architettura 1 si utilizza una codifica localistica delle primitive, ovvero ciascun neurone di uscita codifica una primitiva motoria corrispondente. In questo caso il controllore presenta un numero di neuroni di uscita pari al numero di primitive implementate.

3.3.2.2 Codifica distribuita

Nel caso dell'Architettura 2 si utilizza una codifica distribuita delle primitive motorie, in quanto la rete presenta solo due neuroni di uscita indipendentemente dal numero

di primitive implementate. Tale tipo di codifica presenta rispetto alla precedente due caratteristiche interessanti: la prima è che il numero di unità di uscita può essere inferiore al numero di primitive implementate, la seconda è che la rete neurale è lasciata libera di stabilire come rappresentare ciascuna primitiva.

Poiché i neuroni di uscita sono variabili con continuità nell'intervallo $[0, 1]$, ad ogni training pattern in ingresso alla rete neurale corrisponde una risposta che cade nella porzione di piano cartesiano $[0, 1] \times [0, 1]$. Per semplicità indichiamo con x e y le risposte del primo e del secondo neurone di uscita della rete neurale in Figura 3.5 a destra, rispettivamente.

In questo secondo approccio, l'idea è quella di individuare delle zone nello spazio cartesiano $[0, 1] \times [0, 1]$ associabili a ciascuna delle primitive. Per evitare di fissare a priori queste zone, perdendo di conseguenza la flessibilità di questo approccio, si procede analizzando e processando le risposte fornite dalla rete per gli stimoli associati a ciascuna primitiva motoria. In questo modo, come è stato detto precedentemente, ad ogni training pattern in ingresso la rete associa un punto caratterizzato dalle coordinate x e y sul piano cartesiano. Di conseguenza, ad ogni primitiva è quindi associato un gruppo di punti sul piano, a partire dal quale è possibile costruire un rettangolo che contenga i punti del gruppo stesso. Per fare ciò è sufficiente ricavare per ogni gruppo di punti relativo ad una certa primitiva i valori di x e y minimi e massimi, ottenendo così i vertici dei rettangoli contenenti tali punti. Per motivi che risulteranno chiari in seguito, per ogni gruppo di punti associati alle diverse primitive, viene ricavato il centro di massa (x_{CM}, y_{CM}), con

$$x_{CM} = \frac{\sum_{i=1}^{N} x_i}{N}$$

$$y_{CM} = \frac{\sum_{i=1}^{N} y_i}{N},$$

in cui N è il numero di punti appartenenti al gruppo associato alla primitiva che si sta considerando.

La collocazione dei rettangoli che si ottiene inizialmente dipende fortemente dalla rete neurale inizializzata in modo casuale e non assicura che i rettangoli associati alle diverse primitive siano non sovrapposti. Tipicamente infatti, tali rettangoli sono parzialmente o completamente sovrapposti e per questo motivo è necessario calcolare dei teaching input che spingano la rete a modificare le risposte prodotte per gli stimoli ambigui, quelli cioè che producono risposte che si trovano all'interno delle intersezioni tra i rettangoli. A tal fine, è stata realizzata una funzione di processamento dei rettangoli, il cui obiettivo è semplicemente quello di separare, tramite spostamenti minimi, i rettangoli relativi alle diverse primitive in modo da non avere sovrapposizioni. Poiché l'algoritmo di apprendimento supervisionato utilizzerà come teaching input i punti processati associati alle diverse primitive, nello specifico è necessario spostare tali punti sul piano cartesiano. Per calcolare gli spostamenti da applicare ai punti, sono state considerate le zone di sovrapposizione tra i rettangoli.

Per capire meglio la modalità con cui sono stati processati i rettangoli, si faccia riferimento all'esempio riportato in Figura 3.6, in cui sono stati considerati 5 rettangoli, ciascuno associato ad una diversa primitiva, con i rispettivi centri di massa (le croci). Come si può osservare dalla Figura, ogni regione di sovrapposizione tra rettangoli è a sua volta un rettangolo, caratterizzato da una certa larghezza ed altezza. Per ogni rettangolo $p-esimo$, con $p = 1, .., 5$ si considerano le regioni di intersezione con tutti gli altri rettangoli k, con $k = 1, .., 5$ e $k \neq p$, e si calcolano i valori di larghezza ed altezza di tali zone.

Per i rettangoli che si intersecano con un solo altro rettangolo, lo spostamento minimo coincide proprio con il valore minore tra larghezza dx ed altezza dy della zona rettangolare di sovrapposizione. In particolare, nel caso in cui l'altezza della zona di intersezione abbia valore minore rispetto alla larghezza, lo spostamento minimo è dato dall'altezza dy, con segno positivo se la coordinata y del centro di massa del rettangolo $p-esimo$ è maggiore della coordinata y del centro di massa del rettangolo $k-esimo$,

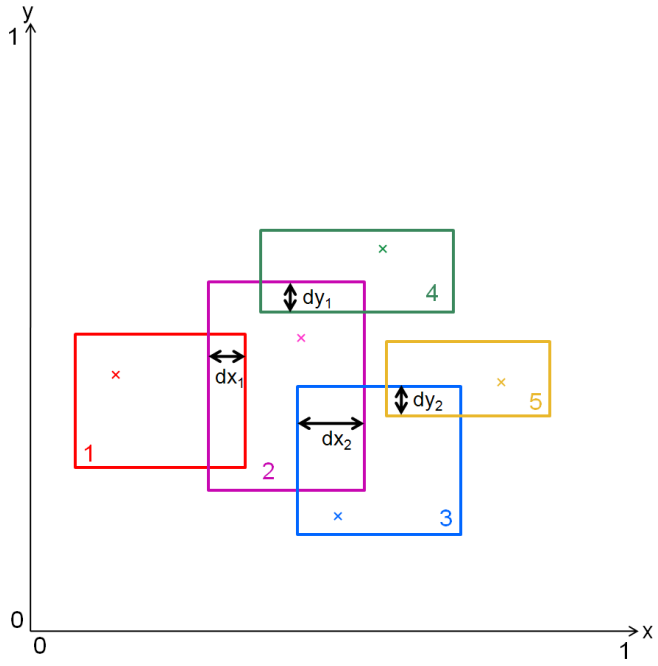

Figura 3.6: Esempio di dislocazione dei rettangoli relativi a generiche primitive motorie nella porzione di spazio cartesiano $[0,1] \times [0,1]$. I numeri contenuti nei rettangoli identificano le primitive motorie relative e le croci i corrispondenti centri di massa.

viceversa con segno negativo. Analogamente accade nel caso in cui la larghezza abbia valore minore rispetto all'altezza: lo spostamento minimo è dato in questo caso dalla larghezza dx, con segno positivo o negativo a seconda che la coordinata x del centro di massa del rettangolo $p-esimo$ sia maggiore o minore della coordinata x del centro di massa del rettangolo $k-esimo$, rispettivamente.

Nel caso in cui il rettangolo $p-esimo$ che si sta considerando sia sovrapposto a più di un rettangolo, viene invece calcolato una sorta di spostamento minimo medio risultante dalla media dei valori minori dei lati delle diverse zone di sovrapposizione.

Nel caso in Figura ad esempio, il rettangolo relativo alla primitiva 2 (fucsia) è sovrapposto al rettangolo relativo alla primitiva 1 (rosso), a quello relativo alla primitiva 3 (blu) e infine a quello relativo alla primitiva 4 (verde). Per maggiore chiarezza, in

Figura 3.6 sono indicati solo i valori minori dei lati delle zone sovrapposizione: nel caso in esame occorre considerare i valori dx_1, dx_2, dy_1. In particolare, considerando lo spostamento da applicare ai punti del rettangolo relativo alla primitiva 2, per quanto precedentemente esposto dy_1 è da considerare con segno negativo, dx_1 con segno positivo e dx_2 con segno negativo. In questo caso, lo spostamento lungo l'asse y coincide proprio con dy_1, mentre lo spostamento lungo l'asse x è pari a $(dx_1 + dx_2)/2$, in cui dx_1 e dx_2 sono considerati con segno.

In generale quindi, per calcolare lo spostamento minimo medio lungo l'asse delle x da applicare ai punti del rettangolo $p - esimo$, è necessario fare il rapporto tra la somma algebrica di tutti i valori dx con segno relativi ai lati minori dei rettangoli di intersezione e il numero di intersezioni che producono tali dx. Analogamente, per calcolare lo spostamento minimo medio lungo l'asse delle y da applicare ai punti del rettangolo $p - esimo$, occorre fare il rapporto tra la somma algebrica di tutti i valori dy con segno e il numero dei dy che concorrono alla somma.

In realtà però i punti del rettangolo $p - esimo$ non vengono spostati uniformemente dello spostamento calcolato, ma secondo un fattore che tiene conto della loro distanza dal centro di massa del rettangolo relativo. A tal proposito viene calcolata la distanza massima di un punto appartenente al rettangolo dal centro di massa dello stesso e lo spostamento di ciascun punto viene moltiplicato per il rapporto tra la sua distanza dal centro di massa e la distanza massima dal centro di massa stesso. In questo modo i punti prossimi al baricentro si spostano di poco, viceversa quelli lontani dal baricentro si spostano di una quantità maggiore.

Tutti i punti che una volta spostati cadono fuori dallo spazio $[0, 1] \times [0, 1]$ vengono collassati sul lato più vicino del perimetro dello spazio: valori di $x < 0$ dei punti processati vengono posti a 0, valori di $x > 1$ vengono posti a 1 e discorso analogo per i valori di y.

Poiché gli spostamenti in gioco sono generalmente piccoli, è necessario richiamare

più volte la funzione di processamento al fine di ottenere rettangoli di categorizzazione completamente separati.

3.4 Dimostrazione

L'utente interagisce con il robot attraverso comandi "linguistici" ed in particolare si suppone che esista un comando linguistico per ogni primitiva implementata. Per poter procedere con l'apprendimento è necessario che vi sia una prima fase di interazione in tempo reale tra robot e utente durante la quale il robot si limita ad eseguire i comandi "linguistici" che il mediatore umano produce per ottenere il comportamento complesso desiderato. Tale fase comprende l'esecuzione ripetuta dello stesso compito al fine di generare un training set, necessario per poter addestrare attraverso un algoritmo di apprendimento supervisionato il sistema di controllo del robot, costituito da una rete neurale che fornisce in uscita i valori in base ai quali determinare quali primitive attivare.

Per conferire maggiore flessibilità alle reti neurali addestrate, sono stati generati training set in cui il compito desiderato viene ripetuto più volte e in condizioni di variabilità. In particolare, si è scelto di costruire training set che contengano un numero minimo di 5 esempi significativi di ciò che il robot dovrà fare, con differenti temporizzazioni nelle transizioni tra le diverse primitive, variando di volta in volta la forma dell'oggetto sul tavolo (cubo, cilindro o sfera) e aggiungendo una piccola randomizzazione sulla posizione desiderata finale di alcune primitive, concordemente con i limiti di giunto.

Nel caso del controllore neurale di Figura 3.5 a sinistra, i teaching input sono ricavati direttamente durante la fase preliminare di interazione tra robot e utente, in cui il robot è guidato dall'utente ad eseguire un certo compito. Il training set generato in questo modo, per ogni training pattern, contiene la risposta desiderata della rete, codificata

come riportato nella Tabella 3.1.

	Teaching Input				
REACH	1	0	0	0	0
CLOSE-HAND	0	1	0	0	0
LIFT	0	0	1	0	0
OPEN-HAND	0	0	0	1	0
MOVE	0	0	0	0	1

Tabella 3.1: Teaching Input per le varie primitive nel caso del controllore realizzato con l'Architettura 1.

Per quanto riguarda invece il controllore di Figura 3.5 a destra, durante la fase iniziale di interazione tra robot e utente è possibile salvare solo gli stati sensoriali esperiti durante le dimostrazioni. I teaching input infatti verranno calcolati successivamente durante l'addestramento della rete, in modo da seguire la naturale tendenza dell'algoritmo di apprendimento supervisionato a spostare i punti modificando i pesi della rete neurale stessa. Poiché quindi in questo secondo caso la costruzione del training set e la scelta dell'algoritmo di addestramento risultano profondamente intrecciati, si rimanda al paragrafo successivo per una spiegazione più dettagliata.

Inoltre, per utilità della funzione di processamento dei rettangoli descritta nel paragrafo 3.3.2.2, durante questa fase viene anche mantenuta memoria in un file della primitiva associata ad ogni training pattern in ingresso.

Infine, per entrambi i controllori, per ogni training pattern in ingresso vengono salvati anche cinque valori compresi tra 0 e 1 che codificano la percentuale di completamento di ciascuna delle primitive: prossimi a 0 se la primitiva corrispondente è appena stata lanciata, viceversa prossimi a 1 se la primitiva ha quasi completato il suo compito. Tali valori non entrano in gioco nel processo di addestramento vero e

proprio ma si sono rivelati utili per i motivi che saranno esposti nel Capitolo successivo (v. paragrafo 4.1.3).

3.5 Addestramento

Il processo di addestramento viene effettuato utilizzando l'algoritmo di apprendimento supervisionato noto con il nome di backpropagation (cfr. 1.7.1). Tale processo risulta disaccoppiato dal robot e dall'ambiente e viene effettuato dopo la fase di dimostrazione.

Per chiarezza di esposizione, da ora in avanti si indicherà con il nome di *ciclo* ogni unità di tempo, pari a $0.01s$ e coincidente con il passo di simulazione. In corrispondenza ad ogni unità di tempo viene aggiornato lo stato dei sensori, dei neuroni interni e dei neuroni di uscita, la posizione dei giunti del robot e degli oggetti posti nell'ambiente. Con il nome di *trial* verrà indicato un insieme di cicli durante i quali il robot esegue un comportamento desiderato e con il nome di *epoca* ogni iterazione dell'algoritmo di backpropagation.

Nel caso di apprendimento del controllore neurale con codifica localistica delle primitive motorie, come è stato detto, l'algoritmo di addestramento utilizza direttamente i comandi linguistici prodotti dal mediatore umano nella fase iniziale di interazione con il robot come teaching input per addestrare il sistema di controllo del robot a generare comandi linguistici analoghi al momento opportuno anche senza l'intervento dello sperimentatore umano.

Per quanto concerne invece il controllore neurale con codifica distribuita delle primitive, come accennato nel paragrafo precedente, la generazione del training set per l'addestramento di tale controllore è strettamente legata all'algoritmo d'addestramento. Infatti, nella fase di dimostrazione in cui il robot è forzato dall'utente ad eseguire le primitive relative al compito scelto, non si è ancora in grado di ricavare quali siano i teaching input associati ad ogni training pattern in ingresso: il training set contenente le

risposte desiderate viene rigenerato ad ogni epoca, mantenendo invariate le informazioni sensoriali esperite durante la fase di dimostrazione. Tale scelta è stata effettuata per fare in modo che la distribuzione finale dei rettangoli segua in qualche modo la naturale tendenza dell'algoritmo di backpropagation a spostare i punti sul piano cartesiano, modificando i pesi della rete neurale. In particolare, si è deciso di richiamare ad ogni epoca di backpropagation la funzione che processi i punti associati a ciascuna primitiva (v. paragrafo 3.3.2.2) fino a che non si ottengono rettangoli completamente separati. In altre parole ad ogni iterazione dell'algoritmo vengono ricalcolati i punti che genera la rete per ogni training pattern in ingresso e su questi viene richiamata più volte, se necessario, la funzione di processamento dei rettangoli, ricavando di conseguenza i teaching input. Tali teaching input vengono utilizzati dall'algoritmo di backpropagation, con l'accortezza di addestrare la stessa rete neurale che è stata utilizzata per la generazione dei training pattern nella fase iniziale di interazione con l'utente. Tale tipologia di apprendimento è sicuramente più ragionevole di quella statica in cui vengono prima processati i punti dei rettangoli fino a che non si ottengono configurazioni con rettangoli completamente separati, per poi procedere con tutte le epoche dell'algoritmo di backpropagation, minimizzando l'errore tra i punti che fornisce la rete in corrispondenza ai training pattern e quelli ottenuti alla fine del processo di separazione dei rettangoli.

Per motivi che saranno chiari nel Capitolo successivo, si è deciso di implementare l'algoritmo di apprendimento tenendo conto della possibilità di normalizzare i pesi delle connessioni in modo da evitare, se necessario, di raggiungere zone di funzionamento saturato delle funzioni d'attivazione dei neuroni (v. equazioni in (1.13)). Tale ragionamento ha luogo in particolare per i neuroni interni e di uscita che vengono aggiornati tramite funzioni d'attivazione logistiche.

3.6 Analisi del comportamento prodotto dal robot

Durante o alla fine del processo di apprendimento viene poi osservato il comportamento prodotto autonomamente dal robot situato nell'ambiente. In questa fase il robot è lasciato libero di muoversi senza l'intervento del mediatore umano, eseguendo dei comandi "linguistici" prodotti autonomamente, attraverso l'acquisizione di una capacità di elicitare autonomamente in un certo contesto senso-motorio le azioni elicitate precedentemente dal mediatore umano attraverso comandi linguistici. Se questa fase di verifica viene effettuata alla fine del processo di apprendimento quello che si desidera è che il robot sia in grado di portare a termine autonomamente il comportamento complesso precedentemente dimostrato.

Poiché l'obiettivo del processo di apprendimento è quindi quello di ottenere un controllore neurale che riesca a elicitare autonomamente la primitiva corretta per ciascun contesto senso-motorio, producendo in uscita comandi "linguistici" analoghi ai teaching input forniti dall'utente, per testare il comportamento delle due architetture sono state fatte le scelte di seguito riportate.

Nel caso del controllore neurale con Architettura 1, si è deciso di attivare ad ogni passo la primitiva corrispondente al neurone con valore più prossimo a 1.

Nel caso del controllore con Architettura 2 invece, è possibile stabilire quali primitive attivare a seconda della posizione del punto relativo all'output di categorizzazione sulla porzione di spazio cartesiano $[0, 1] \times [0, 1]$. A tal fine, in corrispondenza dell'ultima epoca di backpropagation, è necessario salvare su file le informazioni relative ai rettangoli generati dalle uscite fornite dalla rete per ogni ciclo. Nello specifico, occorre salvare gli estremi e il centro di massa di ciascun rettangolo per riuscire a elicitare correttamente le primitive. Il fatto che alla fine del processo di addestramento i rettangoli possano risultare parzialmente sovrapposti è ammissibile in quanto il processo di appredimento, per come è stato implementato in questo secondo caso, tiene conto

della tendenza dell'algoritmo di backpropagation a spostare i punti sul piano. In particolare, la vicinanza o la parziale sovrapposizione dei rettangoli alla fine del processo di apprendimento può risultare utile per facilitare la transizione tra le varie primitive in fase di verifica. In questo secondo caso quindi, per associare a ciascun contesto senso-motorio, ovvero a ciascun punto dello spazio, le primitive corrispondenti, si agisce come segue:

- se l'output di categorizzazione è contenuto in un rettangolo, viene attivata la primitiva corrispondente al rettangolo in cui è contenuto;
- se l'output di categorizzazione è in un punto che non è contenuto in nessun rettangolo, viene attivata la primitiva corrispondente al rettangolo con il centro di massa più vicino al punto;
- se l'output di categorizzazione è contenuto in una zona di sovrapposizione tra due rettangoli, viene attivata la primitiva relativa al rettangolo dell'intersezione con il centro di massa più vicino.

3.6.1 Apprendimento Incrementale

Nel caso la fase di verifica dimostri che la rete non è riuscita ad apprendere correttamente o completamente il compito, si è pensato di sviluppare un sistema di apprendimento incrementale che consenta al robot di raffinare le capacità acquisite durante una serie di sessioni di addestramento aggiuntive effettuate in interazione con l'utente. Durante tali sessioni, il mediatore umano ha la possibilità di adattare il proprio contributo in modo tale da focalizzare il processo di apprendimento sulle singole fasi in cui il robot esegue un comportamento parzialmente o totalmente scorretto.

3.7 Interfaccia Grafica

Per realizzare il processo di apprendimento è stata sviluppata un'interfaccia che consente allo sperimentatore di fornire al robot i comandi "linguistici" mentre il robot interagisce con l'ambiente, di salvare il training set, di effettuare il processo di apprendimento e di osservare il comportamento autonomo del robot alla fine o durante il processo di apprendimento stesso.

Per semplicità, i comandi linguistici non vengono forniti attraverso il linguaggio naturale e un riconoscitore del parlato ma attraverso un'interfaccia a bottoni in cui l'utente può scegliere quale primitiva attivare premendo il corrispondente bottone, mentre osserva il robot e l'ambiente (si veda la Figura 3.7).

Nella parte alta della finestra di interfaccia grafica è presente il gruppo di comandi che interessano nel caso si voglia salvare l'insieme di dati che costituiscono il training set. A questo proposito si procede in maniera differente nel caso in cui si voglia generare un training set per l'Architettura 1 o per l'Architettura 2. Mentre per l'Architettura 1, qualora l'utente volesse generare un nuovo training set è sufficiente spuntare la casella *Salva il Training set* presente sull'interfaccia, scegliere il nome del file sul quale salvare il training set e far eseguire al robot la combinazione di primitive motorie di interesse tramite i bottoni sottostanti, nel caso dell'Architettura 2 occorre prima caricare una nuova rete neurale inizializzata in modo casuale e poi procedere con la combinazione di primitive di interesse. In tal modo, ad ogni unità di tempo viene richiamata un'apposita funzione che salva sul file specificato dall'utente i valori attuali dei segnali di ingresso connessi ai neuroni di ingresso della rete e, nel caso dell'Architettura 1 i valori dei teaching input corrispondenti, mentre nel caso dell'Architettura 2 le uscite che fornisce la rete neurale caricata per ogni training pattern in ingresso. Come è stato già detto, sarà poi compito della funzione di processamento realizzata per implementare la codifica distribuita quello di generare i teaching input corrispondenti ad ogni training pattern

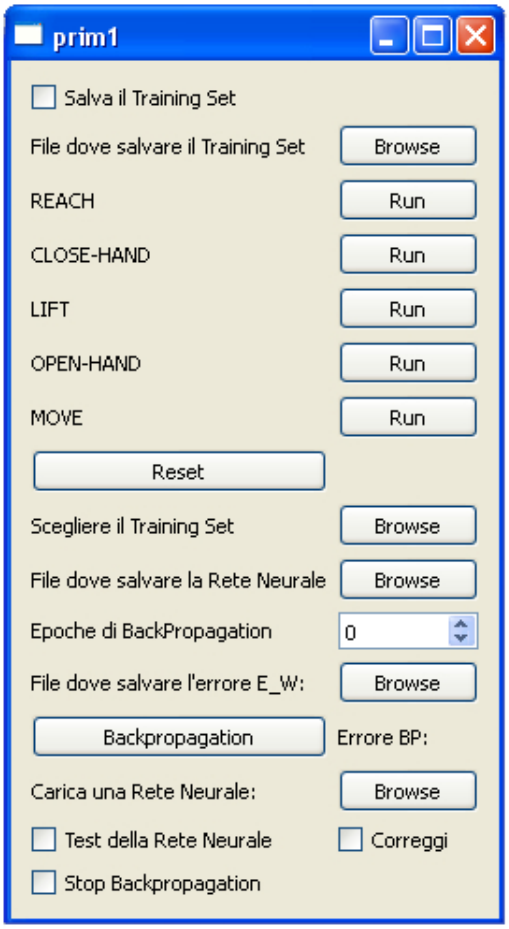

Figura 3.7: Interfaccia grafica

in ingresso per ogni epoca di backpropagation.

Procedendo verso il basso, il bottone di *Reset* consente di portare il braccio del robot in una configurazione iniziale scelta opportunamente per evitare l'urto dello stesso con il tavolino nel compimento dei movimenti legati alle primitive motorie e di creare sul tavolino un oggetto scelto tra cubo, sfera e cilindro. Nella costruzione del training set tale bottone viene utilizzato per riportare il braccio del robot nella configurazione iniziale alla fine di ogni trial e per ricreare di volta in volta un oggetto sul tavolino.

A seguire, sono invece raggruppati i comandi relativi all'apprendimento del controllore sia nel caso si voglia fare l'apprendimento di una nuova rete neurale a partire da un dato training set sia nel caso si voglia fare apprendimento incrementale di una rete precedentemente appresa. In quest'ultimo caso è necessario generare in modo opportuno un nuovo training set, caricare la rete neurale già esistente che si vuole modificare e spuntare la casella identificata dall'etichetta *Correggi*, prima di avviarne l'apprendimento. In particolare, per generare il nuovo training set si lancia il test della rete appresa precedentemente in modo che il robot sia in grado di muoversi autonomamente e si salvano i valori dei sensori in ingresso e delle risposte della rete sul file. In questa fase, quando necessario, l'utente può disabilitare il test della rete spuntando la casella *Correggi* e forzare l'attivazione delle primitive corrette tramite gli appositi bottoni. In questo modo si ottiene un training set a partire dal test della rete precedentemente appresa, modificato solamente là dove l'utente ritiene sia necessario.

Il numero di volte che si vuole reiterare l'algoritmo di backpropagation deve essere specificato nell'apposito spazio e può essere scelto nell'intervallo $[0, 100000]$. Se tuttavia l'errore quadratico medio totale *Errore BP* scende sotto una certa soglia desiderata, è possibile interrompere l'apprendimento tramite l'apposita casella *Stop Backpropagation* anche prima che siano state completate tutte le epoche di backpropagation.

Infine, spuntando la casella *Test della rete neurale* e caricando la rete neurale di interesse è possibile effettuare il test della rete scelta, ovvero osservare il comportamento autonomo del robot dettato dal controllore neurale addestrato.

Capitolo 4

Simulazioni e Risultati

In questo Capitolo vengono presentati i risultati relativi alle simulazioni effettuate per entrambi i controllori neurali oggetto di sperimentazione. In una prima serie di simulazioni, riportate nella sezione 4.1, è stata utilizzata l'Architettura 1 con la codifica localistica delle primitive motorie, mentre nella seconda parte del Capitolo, nella sezione 4.2, vengono riportati i risultati relativi alle simulazioni effettuate con l'Architettura 2 con codifica distribuita delle primitive. Nella parte finale del Capitolo viene infine effettuato un confronto tra i risultati ottenuti a partire dalle due architetture, mettendo in evidenza pregi e difetti di ciascuna di esse.

Come compito di base si è deciso di addestrare il robot affinché apprenda a gestire le transizioni della seguente sequenza di primitive motorie: raggiungere l'oggetto (**REACH**), chiudere le dita (**CLOSE-HAND**), spostare il braccio in corrispondenza del cestino (**LIFT**) e aprire le dita (**OPEN-HAND**). Si noti che in queste simulazioni non è stata sfruttata la presenza della primitiva **MOVE**. Nella Figura 4.1 si può vedere una sequenza di immagini che descrivono il compito scelto, ottenuto attivando le 4 primitive in sequenza.

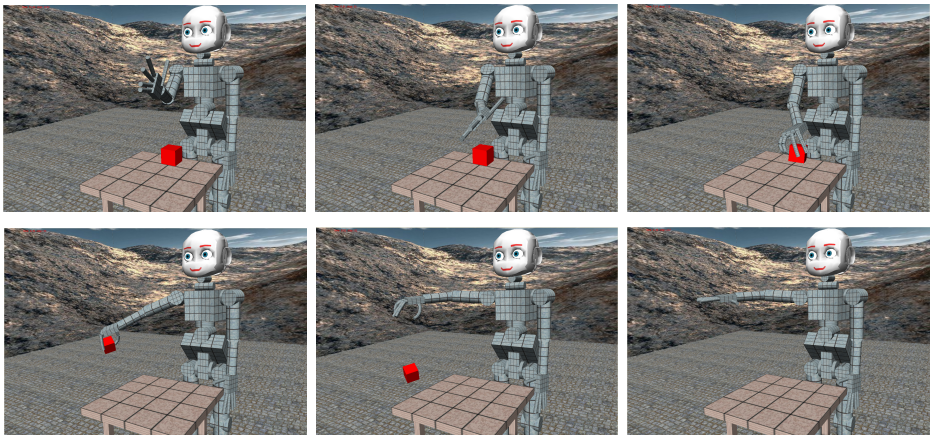

Figura 4.1: Da sinistra verso destra: sequenza di immagini catturate dal simulatore durante l'esecuzione del compito scelto.

4.1 Simulazioni Architettura 1

Di seguito vengono riportati i risultati relativi all'apprendimento del controllore neurale in Figura 3.5 a sinistra, effettuato tramite l'algoritmo di backpropagation dell'errore. In primo luogo vengono analizzati i risultati derivanti dall'apprendimento del controllore suddetto nel caso base, senza alcuna modifica né sul processo d'apprendimento né sulla struttura dell'architettura. Poiché però tale analisi dimostra che il controllore addestrato non riesce a gestire le transizioni tra le primitive, a seguire si illustrano delle varianti in cui il processo di apprendimento e/o l'architettura del controllore sono state modificate al fine di migliorare il risultato. In particolare sono state indagate le seguenti possibilità:

- Apprendimento incrementale sul modello base, grazie al quale l'utente può intervenire nuovamente durante la fase di osservazione del comportamento autonomo del robot, al fine di correggerlo se necessario;
- Modifica del calcolo dell'errore tra le risposte desiderate e le risposte fornite dalla rete ad ogni ciclo di backpropagation, durante ciascuna epoca;

- Esecuzione parallela delle primitive, al fine di favorire la gestione delle transizioni;
- Aggiunta di connessioni ricorrenti sui neuroni interni nella struttura dell'Architettura 1, al fine di inserire una sorta di "memoria" nella rete neurale;
- Apprendimento anticipatorio in cui il training set viene opportunamente rielaborato al fine di anticipare l'elicitazione delle primitive.

Per poter derivare dei risultati generali dalle simulazioni effettuate si è deciso di addestrare 10 reti neurali per ognuno dei casi sopra esposti. Per motivi di chiarezza nelle sezioni successive, per ogni insieme di simulazioni, vengono riportati i grafici relativi all'addestramento di una sola delle 10 reti neurali, presa come esempio tipico di tutte le simulazioni della corrispondente sezione.

Per effettuare un confronto coerente tra le diverse simulazioni è stato utilizzato lo stesso training set iniziale in tutti i casi appena enunciati e ove possibile, la stessa rete di partenza inizializzata in modo casuale. Sempre per motivi di coerenza, in tutte le simulazioni realizzate l'apprendimento è stato interrotto a seguito del raggiungimento di valori di errore E_W (v. equazione (1.9)) pari a 0.008. Il valore del learning rate η è stato scelto pari a 0.05, in quanto si è visto tramite numerose simulazioni che valori maggiori possono influenzare negativamente l'esito dell'apprendimento con backpropagation, ovvero l'andamento della funzione d'errore E_W.

Infine, in tutte le simulazioni di questa sezione è stata indagata la possibilità di normalizzare i pesi dei neuroni interni e d'uscita secondo le modalità riportate in (1.13), ma per ogni valore di β testato non si sono riaggiunti risultati positivi. Nello specifico l'errore quadratico medio complessivo E_W si assesta intorno a valori non sufficientemente bassi, variabili a seconda del β scelto, senza riuscire a scendere ulteriormente. La motivazione di ciò è ragionevolmente dovuta al fatto che il processo di apprendimento del controllore neurale con codifica localistica sfrutta proprio le regioni

di saturazione della funzione d'attivazione logistica per rendere il comportamento dei neuroni simile a quello di interruttori.

4.1.1 Modello di base

Le prime simulazioni che sono state effettuate hanno riguardato l'apprendimento tramite backpropagation del modello base del controllore neurale sotto esame.

In primo luogo viene riportato nel grafico di Figura 4.2 l'andamento dell'errore quadratico medio complessivo nel processo di addestramento della rete ad ogni epoca. Come ci si aspetta E_W tende a decrescere nel tempo, concordemente con l'obiettivo

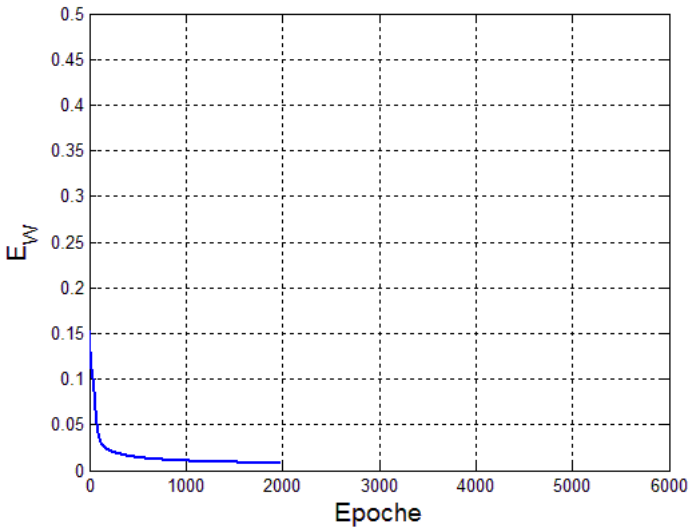

Figura 4.2: Errore quadratico medio complessivo E_W nel processo di addestramento della rete ad ogni epoca nell'esempio tipico.

della backpropagation di minimizzare tale errore. Come si evince dalla Figura bastano circa 2000 epoche di backpropagation per ottenere $E_W = 0.008$.

Per capire meglio la distribuzione dell'errore nel processo di apprendimento del training set, è stato salvato anche il valore dell'errore quadratico medio che l'algoritmo di backpropagation compie ad ogni ciclo, durante l'ultima epoca. L'andamento di tale

valore è riportato nel grafico in Figura 4.3. Osservando il grafico suddetto, si nota come

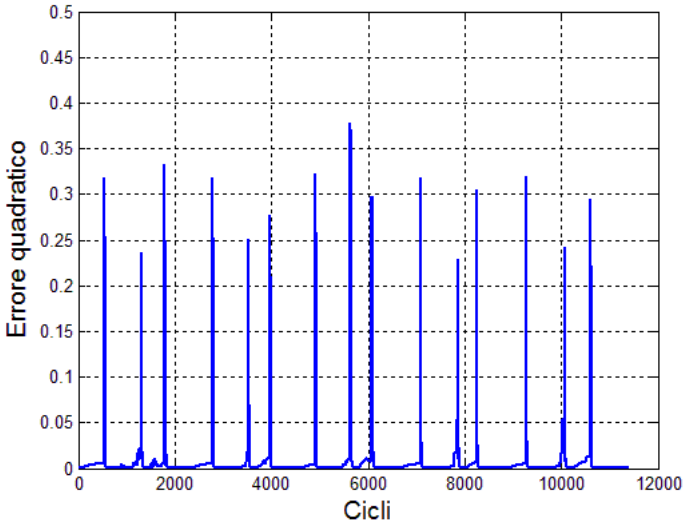

Figura 4.3: Errore quadratico medio ad ogni ciclo, nell'ultima epoca nell'esempio tipico.

l'errore si concentri principalmente in corrispondenza delle transizioni di primitiva, mentre sia praticamente nullo nelle fasi intermedie. La presenza di picchi di errore è dovuta al fatto che in corrispondenza delle transizioni l'algoritmo di apprendimento si ritrova a dover associare a stimoli sensoriali molto simili, se non addirittura identici, risposte differenti.

Infine, per maggiore chiarezza sono state riportate in Figura 4.4 le uscite che fornisce la rete durante l'ultima epoca di backpropagation, in corrispondenza di ogni ciclo. Come è stato già detto, tali uscite rappresentano gli output di categorizzazione, variabili con continuità nell'intervallo $[0, 1]$ e sono in numero pari al numero di primitive. Per semplificare la lettura di questo grafico, nella legenda sono state riportate direttamente le primitive corrispondenti a ciascun neurone di uscita. Si noti che il fatto che il grafico appaia frastagliato non è indicativo in quanto quello che conta è che l'output di categorizzazione relativo alla primitiva corretta sia maggiormente attivato rispetto agli altri.

Osservando la Figura 4.4 sembra che la rete addestrata tenda ad attivare maggiormente la primitiva corretta ad ogni ciclo. In realtà, se si va a studiare il dettaglio

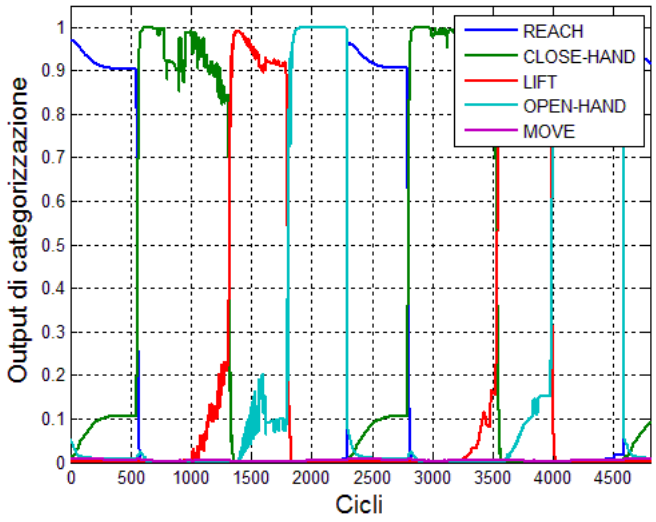

Figura 4.4: Output di categorizzazione in fase di apprendimento nell'esempio tipico.

delle primitive attivate in corrispondenza ad ogni ciclo durante l'ultima epoca di backpropagation, si vede che la rete addestrata non riesce ad attivare maggiormente la primitiva corretta in corrispondenza ai cicli immediatamente successivi alle transizioni. Infatti, come si evince dalla Figura 4.5, in cui le linee nere verticali identificano i cicli in corrispondenza dei quali avvengono le transizioni nel training set e le linee colorate identificano le diverse primitive attivate ad ogni ciclo secondo quanto stabilito dalla legenda, la rete addestrata esibisce un comportamento leggermente ritardato. In altre parole, la rete addestrata tende ad effettuare le transizioni tra le primitive qualche ciclo dopo l'effettiva transizione presente nel training set. Nello specifico, per quanto concerne il primo trial dell'esempio tipico, la rete tende ad attivare la primitiva successiva di una transizione mediamente 20 cicli dopo che è avvenuta l'effettiva transizione nel training set. Quanto detto motiva la presenza dei picchi

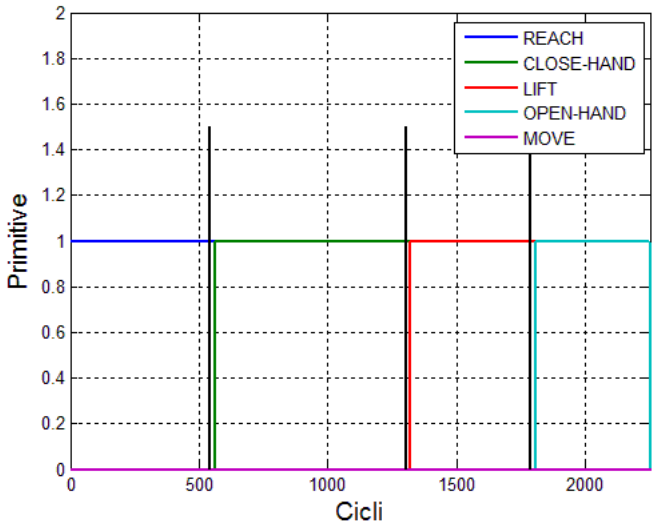

Figura 4.5: Dettaglio sul primo trial dell'esempio tipico: le linee nere verticali identificano i cicli in corrispondenza dei quali avvengono le transizioni tra le primitive nel traning set; le linee colorate identificano l'attivazione delle diverse primitive. In particolare, la primitiva risulta attiva se la linea colorata corrispondente secondo quanto espresso dalla legenda è a 1, viceversa è a 0.

di errore alle transizioni nel grafico in Figura 4.3. Il fatto che la rete non riesca ad attivare la primitiva successiva al momento giusto nella fase di processamento della sequenza di stati sensoriali esperiti durante le dimostrazioni, è una delle ragioni principali che provoca l'incapacità del sistema di passare alla primitiva successiva una volta che il robot viene situato nell'ambiente e valutato per la capacità di produrre il comportamento corretto in modo autonomo. Infatti, come si evince dalla Figura 4.6, in cui vengono riportati gli output di categorizzazione in uscita dal controllore neurale in fase di verifica, il robot non riesce a gestire autonomamente le transizioni tra le diverse primitive, bloccandosi alla fine dell'esecuzione della primitiva **REACH**. Tuttavia, se si forza il robot ad eseguire la seconda primitiva della sequenza anche solo per un passo di simulazione, la rete addestrata sarà poi in grado di continuare ad attivare

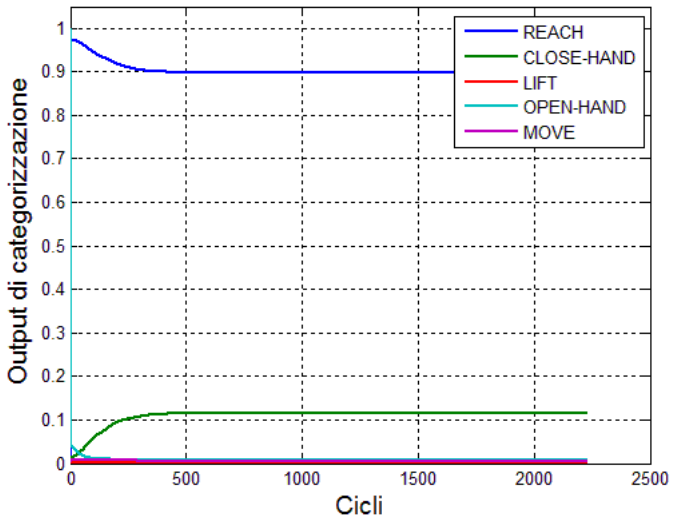

Figura 4.6: Output di categorizzazione in fase di test nell'esempio tipico.

autonomamente la primitiva corretta, bloccandosi alla transizione successiva. Ciò può essere dovuto alle seguenti due ragioni: da una parte gli stimoli sensoriali percepiti, che dipendono dal robot nell'ambiente e dalle azioni da esso eseguite, possono essere diversi da quelli esperiti durante la fase di dimostrazione e il controllore non riesce a generalizzare; dall'altra, in corrispondenza delle transizioni il sistema può entrare in una fase di stallo a causa del fatto che l'esecuzione della primitiva corrente produce variazioni pressocché nulle sullo stato sensoriale. Per spiegare meglio quest'ultimo tipo di problema si può far riferimento all'esempio presentato nella Tabella 4.1, in cui con s_i, $i = 1, .., 4$ sono indicati gli stati sensoriali prima e dopo l'esecuzione della generica primitiva, e con p_i, $i = 1, .., 5$ è indicata la primitiva che viene attivata. Mentre nel caso riportato nella colonna di sinistra gli stati sensoriali prodotti in seguito all'esecuzione della prima primitiva risultano differenti da quelli di partenza, consentendo ad un certo punto l'attivazione della primitiva successiva, nella colonna di destra è riportato un esempio in cui ciò non accade. In particolare, l'esecuzione della prima primitiva in

Situazione ideale	Situazione di stallo
$s_0 \rightarrow p_1 \rightarrow s_1$	$s_0 \rightarrow p_1 \rightarrow s_1$
$s_1 \rightarrow p_1 \rightarrow s_2$	$s_1 \rightarrow p_1 \rightarrow s_2$
$s_2 \rightarrow p_1 \rightarrow s_3$	$s_2 \rightarrow p_1 \rightarrow s_2$
$s_3 \rightarrow p_2 \rightarrow s_4$	$s_2 \rightarrow p_1 \rightarrow s_2$
..	..

Tabella 4.1: Esempio esplicativo.

questo secondo caso porta a stati sensoriali identici agli stati di partenza provocando di conseguenza un'evitabile situazione di stallo.

L'influenza di tale tipo di problema è d'altra parte confermata dal fatto che, se durante la fase di verifica in cui il robot è autonomo si forza l'esecuzione della primitiva successiva durante le fasi di stallo anche solo per una o due unità di tempo, forzando di conseguenza una variazione dello stato sensoriale, il robot si dimostra in grado di eseguire l'intera sequenza di comportamenti in modo corretto. Questo problema, come si è visto nella Figura 4.4, non si verifica durante il processo di apprendimento dal momento che in quel caso le transizioni alle primitive successive sono imposte direttamente dal dimostratore.

4.1.1.1 Apprendimento incrementale

Al fine di ottenere una rete addestrata funzionante, si è pensato di aggiungere un'ulteriore fase di interazione con l'utente, ovvero di implementare l'apprendimento incrementale. In questo modo è possibile riapprendere la rete già appresa a partire dal training set ottenuto dalla fase di test della stessa opportunamente corretto. Tuttavia le prove effettuate in questo contesto non hanno risolto il suddetto problema della transizione temporale tra le diverse primitive.

4.1.1.2 Modifica del calcolo dell'errore

In una terza serie di simulazioni si è deciso di utilizzare come teaching input i valori 0.1 e 0.9 anziché 0 e 1 in modo da evitare che i pesi del controllore neurale tendessero a crescere troppo durante il processo di apprendimento. Nello specifico piuttosto che modificare il training set generato, è stata modificata la funzione che calcola l'errore tra i teaching input e le uscite correnti della rete in modo da porre a zero tale errore quando è minore di 0.1: ciò è equivalente ad avere una risposta della rete che in linea di principio può variare tra 0.1 e 0.9. In tal modo la backpropagation evita di concentrarsi nel mandare gli errori a zero, considerandoli zero anche quando in realtà sono molto piccoli. Tale accorgimento può essere realizzato nel caso in esame in quanto si considerano uscite binarie per le quali non interessa avere errori piccoli molto precisi. Tuttavia, tale modifica non ha prodotto miglioramenti rispetto al problema della gestione delle transizioni.

4.1.1.3 Esecuzione parallela delle primitive

In un'altra serie di simulazioni si è deciso di consentire al robot di attivare tutte le primitive in parallelo, pesando l'effetto di ciascuna sulla base dello stato di attivazione dell'output di categorizzazione corrispondente. In particolare, ciò è stato realizzato applicando ai giunti la somma algebrica delle velocità calcolate da ciascuna primitiva, pesate per lo stato di attivazione della primitiva corrispondente. Come si evince dalla Figura 4.7, che comunque rappresenta in qualche modo l'attivazione delle primitive durante la fase di verifica, il robot riesce a gestire correttamente le transizioni tra le diverse primitive. Tale risultato si spiega con il fatto che, come detto precedentemente, nelle fasi di transizione il controllore riceve stimoli sensoriali simili sia immediatamente prima che dopo la transizione stessa. Tali stimoli tendono ad attivare maggiormente entrambe le primitive corrispondenti alle transizioni spingendo il robot ad anticipare

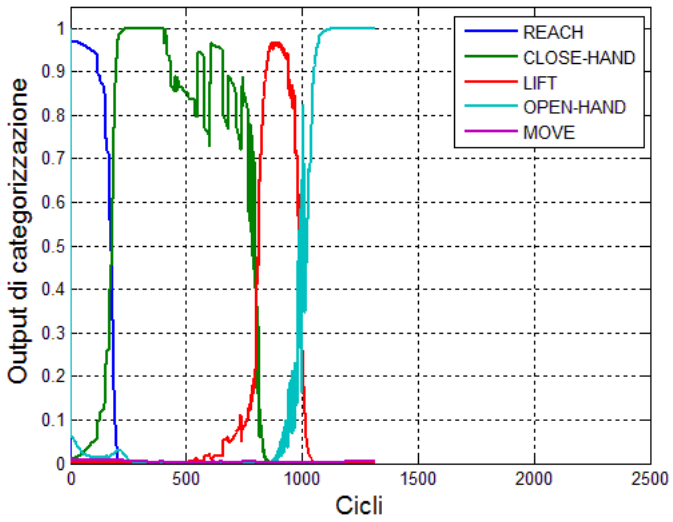

Figura 4.7: Output di categorizzazione in fase di test nell'esempio tipico.

spontaneamente, sia pure in parte, l'attivazione della primitiva successiva. Questo effetto di anticipazione, sia pure parziale, della primitiva successiva influenza gli stimoli sensoriali esperiti successivamente dal robot in modo tale da favorire in modo crescente l'attivazione della primitiva successiva stessa e la progressiva disattivazione della primitiva precedente. In questo modo il robot evita le situazioni di stallo, riuscendo a gestire correttamente le transizioni tra le primitive.

Occorre tuttavia notare che in alcuni casi il robot, pur riuscendo a gestire correttamente le transizioni tra le varie primitive, non è in grado di completare il compito proposto. Questo accade sistematicamente in tutti i casi in cui sul tavolino è presente un oggetto di forma sferica, in quanto il robot tende a chiudere le dita troppo presto, non riuscendo quindi ad afferrare l'oggetto. Viceversa in tutti gli altri casi in cui sul tavolino sia presente un oggetto di forma cilindrica o cubica, il robot riesce ad afferrare l'oggetto anche anticipando la chiusura delle dita. Ciò è dovuto alla forma degli oggetti che in questo secondo caso consentono una presa più facile anche se il

palmo della mano non è posizionato correttamente.

4.1.2 Architettura 1 con Connessioni Ricorrenti

In un'ulteriore serie di simulazioni sono state utilizzate delle reti neurali con connessioni ricorrenti in modo da permettere al robot di tenere traccia degli stati senso-motori precedenti e quindi del passare del tempo. Tali simulazioni sono state effettuate al fine di verificare se tali reti fossero in grado di sfruttare il passare del tempo o il permanere in una situazione di stallo per favorire le transizioni alle primitive successive.

A tal proposito l'Architettura 1 è stata modificata opportunamente inserendo delle connessioni ricorrenti sui neuroni interni come illustrato in Figura 4.8. Per poter adoperare l'algoritmo di backpropagation anche in questo caso, è stata adottata l'architettura di Elman in Figura 1.4, in cui tra gli ingressi vengono inserite delle unità di memoria aggiuntive che mantengono una copia dei valori dei neuroni interni al passo precedente.

Procedendo con l'apprendimento delle 10 reti neurali con ricorrenze, si è visto che

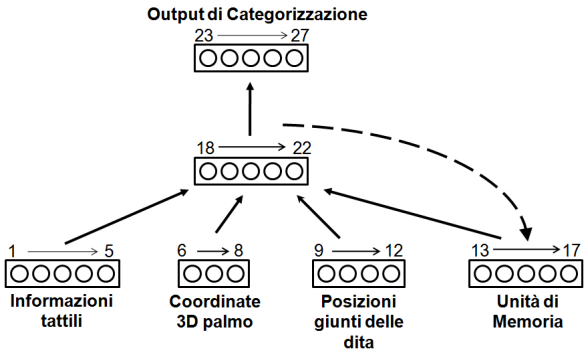

Figura 4.8: Architettura 1 con ricorrenza sulle unità interne.

sono necessarie mediamente 700 epoche di backpropagation per raggiungere valori di errore $E_W = 0.08$ e che solo il 20% delle reti neurali addestrate riesce a completare il

compito qualunque sia l'oggetto presente sul tavolo. Infatti, come si può osservare dall'istogramma in Figura 4.9, la percentuale delle reti che completano il compito correttamente cambia a seconda che l'oggetto sul tavolo sia un cubo, un cilindro o una sfera, nonostante tutti e tre siano stati utilizzati durante la fase di dimostrazione. La motivazione di ciò sta presumibilmente nel fatto che il training set utilizzato per

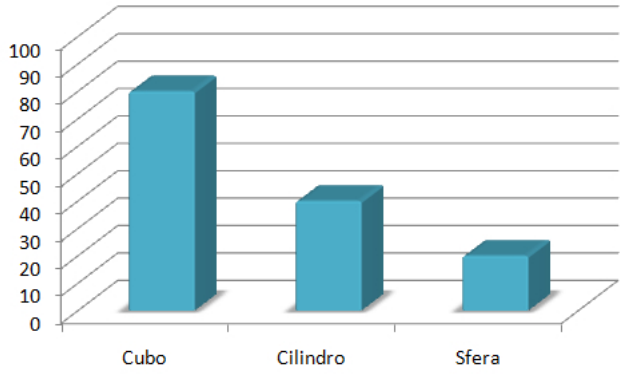

Figura 4.9: Percentuale di reti addestrate funzionanti a seconda dell'oggetto presente sul tavolino.

queste simulazioni contiene 3 trial in cui l'oggetto sul tavolino è un cubo, ed un solo trial in cui l'oggetto è una sfera o un cilindro.

4.1.3 Apprendimento Anticipatorio

In un'altra serie di simulazioni, al fine di facilitare la transizione alla primitiva successiva si è deciso di rielaborare il training set chiedendo al robot di anticipare il passaggio alla primitiva successiva, in modo da spingerlo ad effettuare le transizioni richieste. A tal proposito si è deciso di implementare due diverse funzioni di rielaborazione del training set: con anticipazione a gradino e con anticipazione a rampa. Per capire meglio il comportamento di tali funzioni di rielaborazione si può far riferimento alla Figura 4.10. Nell'esempio illustrato in Figura, sono riportati in nero i valori dei teaching input

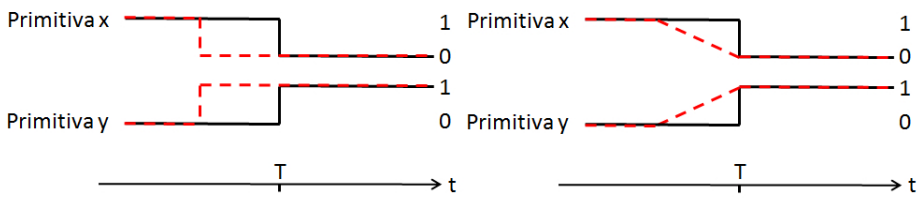

Figura 4.10: Esemplificazione di come è stato rielaborato il training set. L'immagine a sinistra mostra un esempio di anticipazione a gradino, l'immagine a destra un esempio di anticipazione a rampa.

binari corrispondenti alle due primitive x e y rispetto al tempo per entrambi i tipi di rielaborazione: la primitiva x è attiva per $t < T$, mentre la primitiva y è attiva per $t > T$ e viceversa. In rosso invece viene riportato l'andamento dei teaching input anticipati per le due primitive x e y in corrispondenza della transizione. Nel caso di rielaborazione con anticipazione a rampa (a destra in Figura 4.10) le uscite desiderate, originariamente binarie, vengono modificate in modo da variare linearmente in corrispondenza delle transizioni, mentre nel caso di rielaborazione con anticipazione a gradino (a sinistra in Figura 4.10) le uscite desiderate vengono semplicemente anticipate mantenendo carattere binario. Per quanto detto la rielaborazione a rampa dovrebbe risultare quindi più semplice da apprendere visto che gli stimoli sensoriali immediatamente precedenti e successivi ad una transizione tendono ad essere simili e tendono dunque ad attivare gli output di categorizzazione corrispondenti in modo simile.

Per stabilire il numero di pattern del training set interessati dall'anticipazione in corrispondenza di ogni transizione, si è deciso di utilizzare dei valori che codificano la percentuale di completamento delle primitive, evitando di stabilire a priori un numero fisso di pattern da anticipare. Tali valori, come già detto sono compresi tra 0 e 1: prossimi a 0 se la primitiva corrispondente è appena stata lanciata, viceversa prossimi a 1 se la primitiva ha quasi completato il suo compito. In particolare nelle simulazioni

successive vengono processate tante righe del training set quante sono quelle in cui lo stato di completamento della primitiva corrente è maggiore dello stato stesso in corrispondenza della transizione abbassato di un quarantesimo del suo valore. Tale soglia è stata scelta empiricamente, al fine di ottenere reti addestrate funzionanti per entrambi i tipi di funzioni di rielaborazione del training set. In realtà si è visto che la

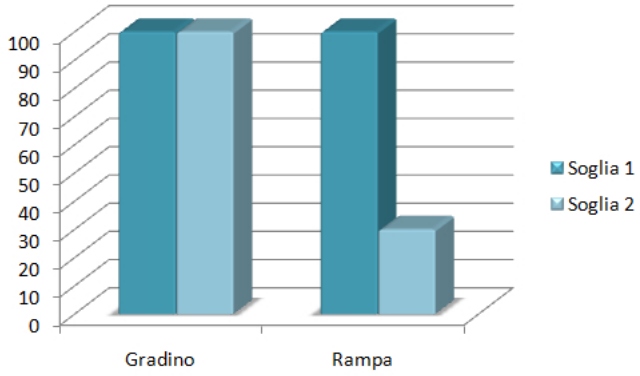

Figura 4.11: Percentuale di reti addestrate funzionanti per tutti i tipi di oggetti sul tavolino e per i due tipi di processamento del training set, utilizzando due differenti soglie: Soglia 1 è la soglia adottata nelle simulazioni, Soglia 2 corrisponde ad un numero di pattern processati inferiore.

rielaborazione con anticipazione a gradino porta a risultati positivi nel 100% dei casi anche processando un numero inferiore di pattern del training set in corrispondenza delle transizioni, mentre la rielaborazione a rampa anticipata risulta più sensibile a variazioni di soglia (v. Figura 4.11). In questo contesto si indica con risultato positivo la rete funzionante, ovvero la rete in grado di portare a termine il compito, gestendo correttamente le transizioni tra le primitive.

4.1.3.1 Addestramento con rielaborazione a gradino anticipato

Dopo aver rielaborato il training set tramite la funzione di anticipazione a gradino, si è quindi proceduto con l'addestramento della rete neurale con Architettura 1, ottenendo

un controllore in grado di gestire correttamente le transizioni tra le varie primitive. Nelle Figure 4.12 e 4.13 vengono riportati i grafici relativi all'errore legato al processo di apprendimento tramite backpropagation in questo secondo caso. Come si evince

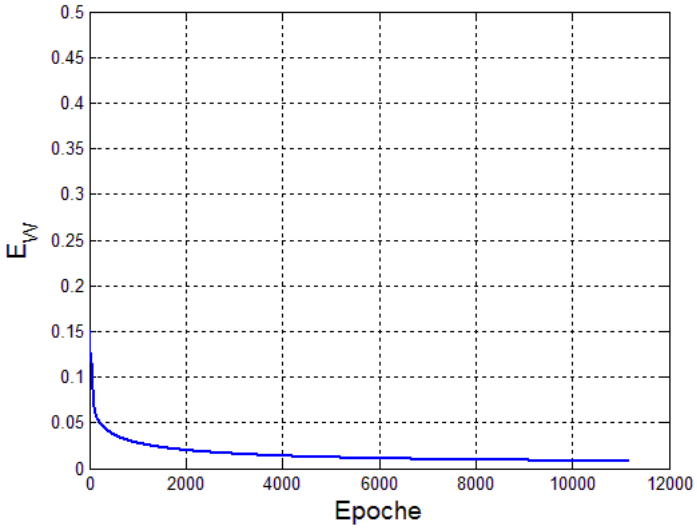

Figura 4.12: Errore quadratico medio complessivo E_W nel processo di addestramento della rete ad ogni epoca nell'esempio tipico.

dal grafico in Figura 4.12, l'algoritmo di backpropagation impiega molte più epoche, circa 11000, per raggiungere un errore quadratico medio complessivo $E_W = 0.008$, come nel caso precedente (v. Figura 4.2). Il fatto che occorrano molte più epoche di backpropagation per ottenere un errore E_W comparabile con quello del caso precedente è dovuto alle modifiche apportate nella fase di rielaborazione del training set, che rendono lo stesso più difficilmente apprendibile per l'algoritmo di backpropagation. Inoltre, in corrispondenza dell'ultima epoca, i picchi legati all'errore quadratico medio ad ogni ciclo risultano di entità minore rispetto a quelli in Figura 4.3. Anche in questo caso risulta che in linea di principio la rete è in grado di attivare maggiormente la primitiva corretta in corrispondenza di ogni training pattern in ingresso, come si evince dalla Figura 4.14 in cui vengono riportati i valori degli output di categorizzazione in

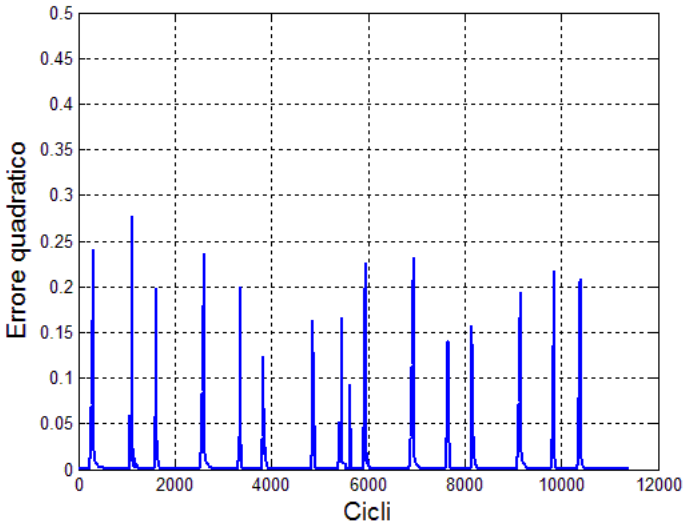

Figura 4.13: Errore quadratico medio ad ogni ciclo, nell'ultima epoca nell'esempio tipico.

corrispondenza ad ogni ciclo durante l'ultima epoca di backpropagation. A seguire

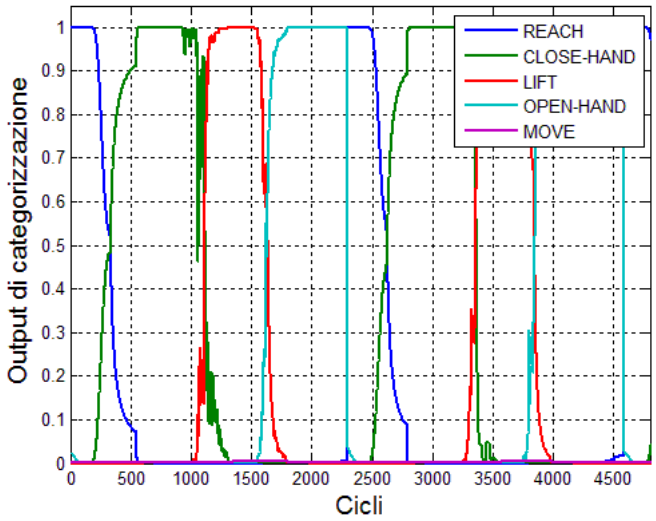

Figura 4.14: Output di categorizzazione in fase di apprendimento nell'esempio tipico.

viene riportato anche l'andamento degli output di categorizzazione in fase di test della

rete neurale (v. Figura 4.15), dal quale si vede che il controllore riesce a gestire correttamente il sequenziamento delle quattro primitive.

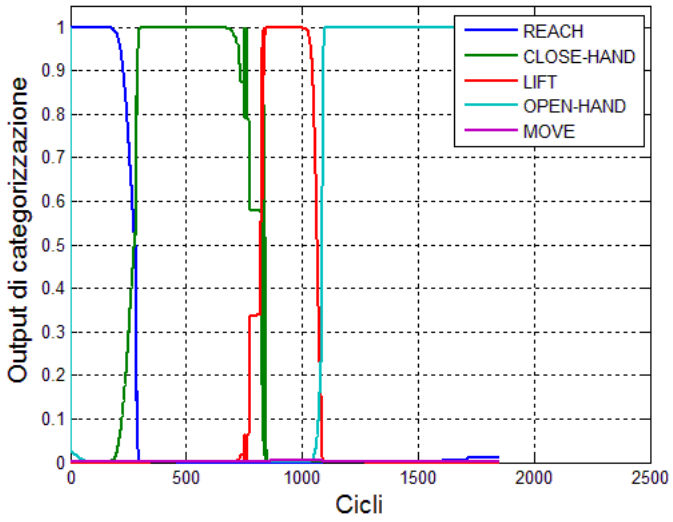

Figura 4.15: Output di categorizzazione in fase di test nell'esempio tipico.

4.1.3.2 Addestramento con rielaborazione a rampa anticipata

Di seguito vengono invece riportati i risultati relativi all'addestramento della rete neurale in esame dopo che il training set sia stato rielaborato con anticipazione a rampa. Dal grafico relativo all'errore commesso dall'algoritmo di backpropagation in Figura 4.16 si evince che l'algoritmo impiega un numero di epoche intermedio rispetto al caso del modello base e a quello dell'addestramento del controllore con rielaborazione a gradino anticipato del training set, per raggiungere un errore quadratico medio complessivo $E_W = 0.008$: circa 4500 epoche. Inoltre dal grafico in Figura 4.17 si può notare come l'errore quadratico medio commesso ad ogni ciclo durante l'ultima epoca di backpropagation sia più basso, nonostante permangano i picchi relativi alle transizioni tra le primitive. Questi grafici confermano quindi l'ipotesi che il processamento a

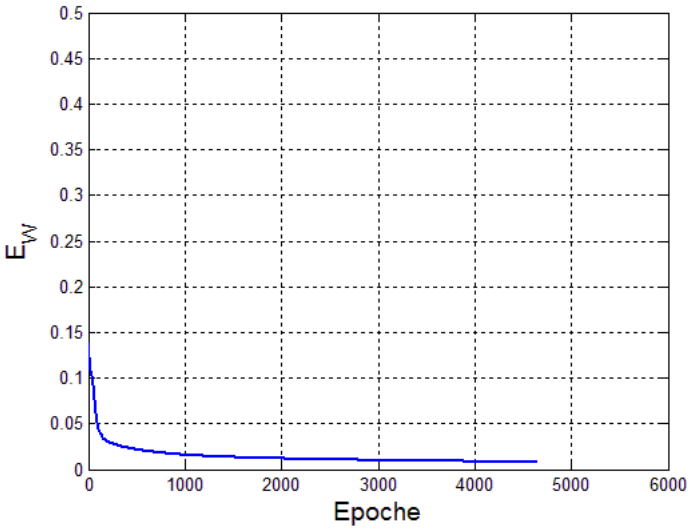

Figura 4.16: Errore quadratico medio complessivo E_W nel processo di addestramento della rete ad ogni epoca nell'esempio tipico.

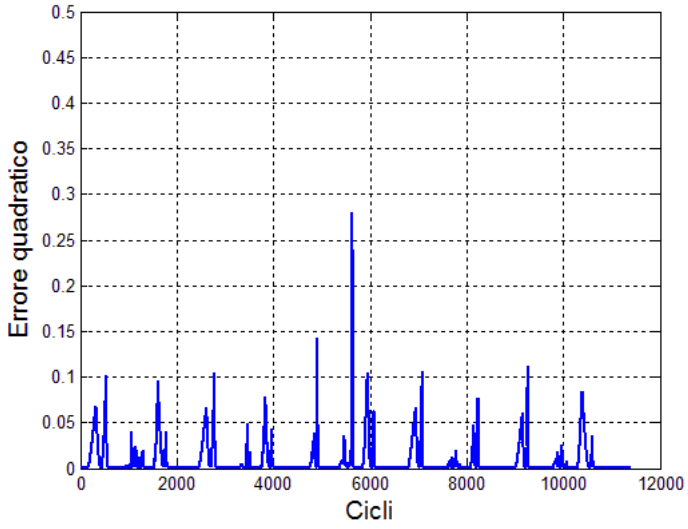

Figura 4.17: Errore quadratico medio ad ogni ciclo, nell'ultima epoca nell'esempio tipico.

rampa anticipata renda il training set più facilmente apprendibile tramite l'algoritmo di backpropagation rispetto a quello a gradino anticipato.

Dalla Figura 4.18 si vede come la rete apprenda ad attivare maggiormente la primitiva corretta praticamente ad ogni ciclo. Infine, dal grafico relativo agli output di

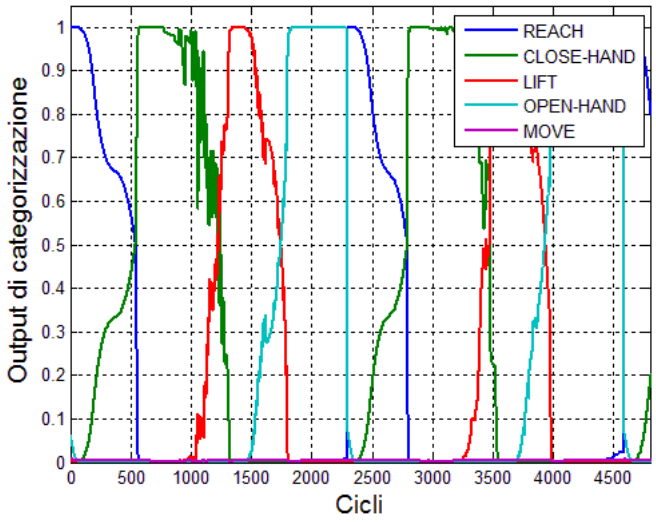

Figura 4.18: Output di categorizzazione in fase di apprendimento nell'esempio tipico.

categorizzazione della rete appresa in fase di test in Figura 4.19, si evince che la rete si comporta correttamente anche in fase di test.

Da un'analisi effettuata su tutte e 10 le reti addestrate in questa serie di simulazioni si può affermare che la rete appresa con rielaborazione del training set a rampa anticipata, pur necessitando di un numero minore di epoche di apprendimento rispetto a quella con rielaborazione a gradino anticipato, impiega più cicli per completare il compito desiderato.

4.1.4 Considerazioni e confronto

Nel grafico di Figura 4.20 vengono riportate le percentuali di reti addestrate in grado di gestire correttamente le transizioni tra le diverse primitive nei quattro casi di maggiore interesse precedentemente descritti. Tali percentuali, come già detto, sono state ricavate

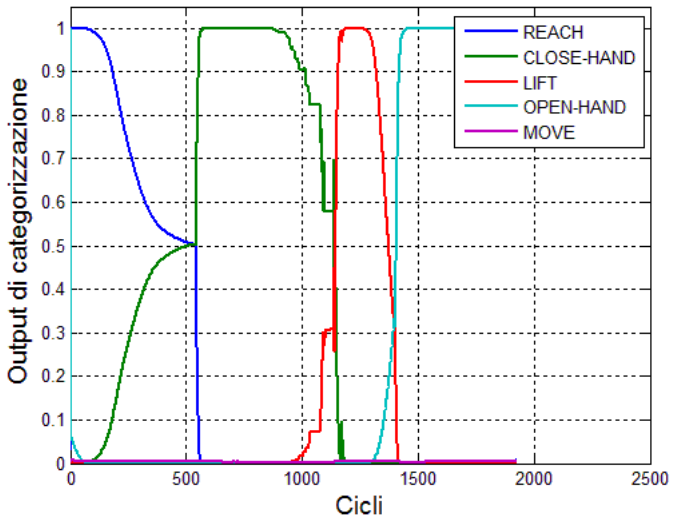

Figura 4.19: Output di categorizzazione in fase di test nell'esempio tipico.

sulla base dell'addestramento di 10 reti neurali per ogni serie di simulazioni e sono state calcolate tenendo in considerazione il numero di trial dedicati a ciascun oggetto nella costruzione del training set. In particolare, poiché il training set utilizzato per tutte le simulazioni di questa sezione è costituito da 3 trial in cui l'oggetto sul tavolino è un cubo, 1 in cui l'oggetto è un cilindro e 1 in cui l'oggetto è una sfera, si è deciso di dare peso triplo alle reti in grado di gestire correttamente le transizioni con il cubo rispetto a quelle che riescono a gestirle con gli altri oggetti. Nello specifico:

- le reti in grado di gestire correttamente le transizioni solamente nel caso in cui l'oggetto sul tavolino sia un cilindro o una sfera pesano il 2%;
- le reti in grado di gestire correttamente le transizioni sia con il cilindro che con la sfera sul tavolino pesano il 4%;
- le reti in grado di gestire correttamente le transizioni solamente nel caso in cui l'oggetto sul tavolino sia un cubo pesano il 6%;

- le reti che oltre a gestire correttamente le transizioni nel caso in cui l'oggetto sul tavolino sia un cubo, riescono a gestirle anche con un cilindro o una sfera sul tavolino pesano l'8%;
- le reti in grado di gestire correttamente le transizioni indistintamente per ciascuno dei tre oggetti, cubo, sfera o cilindro, presenti sul tavolino pesano il 10%.

Per questo motivo, ad esempio, la colonnina relativa all'esecuzione parallela delle primitive indica una percentuale dell'80% in quanto quando l'oggetto sul tavolino è di forma sferica il compito sistematicamente non viene portato a termine.

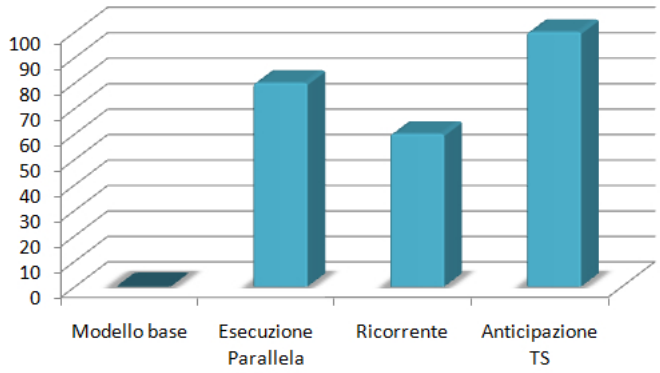

Figura 4.20: Percentuale di reti addestrate funzionanti tenendo conto dei tre diversi oggetti sul tavolino nei quattro casi di maggiore interesse: apprendimento del modello di base, esecuzione parallela delle primitive, inserimento di ricorrenze nell'architettura del controllore neurale e rielaborazione del training set con anticipazione.

Il risultato fondamentale che si evince dall'analisi svolta è che, poiché in corrispondenza delle transizioni gli stimoli sensoriali associati alle primitive sono praticamente identici, il modello di base non produce risultati positivi. Dalla possibilità di eseguire parallelamente le primitive si è visto che il robot spontaneamente tende ad adottare un meccanismo di anticipazione, attivando la primitiva successiva di una transizione in modo crescente prima dell'effettiva transizione nel training set, e

continuando nel contempo a mantenere attivata anche la primitiva precedente in modo decrescente.

Procedendo su questa strada, si è implementato un meccanismo di anticipazione apposito rielaborando opportunamente il training set in modo che la rete impari ad attivare la primitiva successiva quando ancora gli ingressi sensoriali della primitiva precedente variano in maniera percepibile. Tale anticipazione facilita il processo di apprendimento nel senso che consente di gestire più facilmente e correttamente le transizioni ed evita che il robot incorra in situazioni di stallo in corrispondenza delle transizioni tra le varie primitive.

Per completezza, sono state infine indagate le capacità di apprendimento di reti neurali ricorrenti, che come si evince dal grafico hanno portato a risultati positivi circa nel 60% dei casi.

4.2 Simulazioni Architettura 2

In questa sezione vengono invece riportati i risultati relativi all'apprendimento del controllore neurale in Figura 3.5 a destra, effettuato sempre tramite l'algoritmo di backpropagation dell'errore. Come nel caso precendente, inizialmente sono stati analizzati i risultati derivanti dall'apprendimento del controllore suddetto nel caso del modello di base, senza alcuna modifica né sul processo d'apprendimento né sulla struttura dell'architettura. A differenza della codifica localistica, l'approccio con codifica distribuita produce risultati positivi già in questo caso, in quanto alcune delle reti addestrate riescono a gestire correttamente la temporizzazione delle primitive. Tuttavia, per indagare possibili miglioramenti e per effettuare dei confronti con il caso precedente, si è pensato di procedere con l'apprendimento incrementale sul modello di base, di modificare la modalità di attivazione delle primitive in modo da consentire l'esecuzione parallela delle stesse, di modificare la struttura del controllore

neurale tramite connessioni ricorrenti sui neuroni interni, e infine di analizzare i risultati derivanti dall'implementazione di un apprendimento di tipo anticipatorio con rielaborazione del training set.

Anche per queste simulazioni, per poter effettuare un confronto coerente è stato utilizzato lo stesso training set iniziale in tutti i casi appena enunciati e ove possibile, la stessa rete di partenza inizializzata in modo casuale. Per poter effettuare un confronto anche con i risultati ottenuti nelle simulazioni dell'Architettura 1 il training set è costituito anche in questo caso da 3 trial in cui l'oggetto sul tavolino è di forma cubica, 1 trial con un oggetto di forma sferica ed 1 trial in cui l'oggetto sul tavolino è di forma cilindrica.

Poiché in questo caso, per come è implementato l'algoritmo di apprendimento, il valore dell'errore quadratico medio complessivo E_W risulta poco indicativo, si è scelto di interrompere l'apprendimento a seguito del raggiungimento di 40000 epoche di backpropagation. Il valore del learning rate η è stato mantenuto pari a 0.05.

Infine, anche in tutte le simulazioni di questa sezione è stata indagata la possibilità di normalizzare i pesi dei neuroni interni e d'uscita secondo le modalità riportate in (1.13), per alcuni valori di β. In questo caso la normalizzazione risulta fondamentale in quanto si sta implementando una codifica distribuita in cui piccole variazioni del net input in ingresso al neurone devono essere distinte in uscita. In particolare, il raggiungimento di zone di funzionamento saturato della logistica risulta controproducente in quanto può causare problemi nella dislocazione dei rettangoli, ovvero può aumentare la tendenza dei rettangoli a collassare lungo i bordi dello spazio $[0, 1] \times [0, 1]$.

In questa sezione, per chiarezza di esposizione, anziché procedere come nella sezione precedente esponendo dettagliatamente le varie modifiche apportate al sistema e i relativi miglioramenti, viene presentato prima un confronto tra i vari approcci studiati e poi un esempio di rete addestrata in grado di gestire correttamente le temporizzazioni tra le diverse primitive.

4.2.1 Considerazioni e confronto

Anche in questo caso, allo stesso modo descritto per l'Architettura 1 nel paragrafo 4.1.4, è possibile calcolare la percentuale di reti addestrate funzionanti ottenuta tenendo in considerazione la forma degli oggetti sul tavolino, a partire dalle 10 reti neurali apprese nei quattro casi di maggiore interesse: apprendimento del modello di base, esecuzione parallela delle primitive, architettura con connessioni ricorrenti e apprendimento con training set rielaborato in modo da anticipare l'elicitazione delle primitive. Come si evince dall'istogramma in Figura 4.21, per ciascuna di queste categorie sono stati provati i casi senza normalizzazione dei pesi delle connessioni e con normalizzazione con diversi valori di β. Si noti che i risultati relativi all'apprendimento incrementale sulle reti apprese nel caso del modello di base non sono stati riportati in quanto anche in questo caso non hanno prodotto miglioramenti rispetto al problema della gestione delle transizioni tra le primitive.

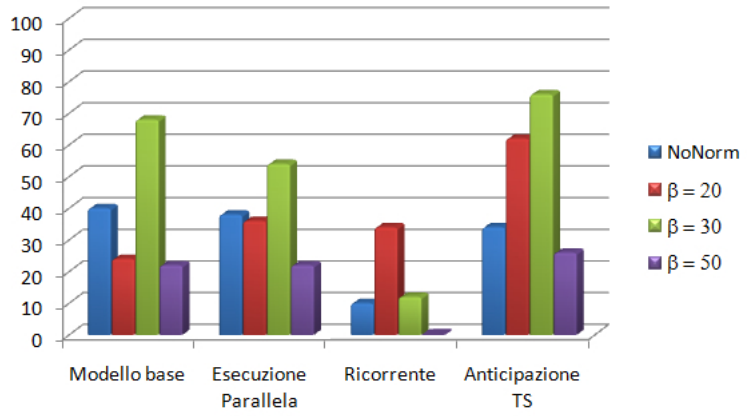

Figura 4.21: Percentuale di reti addestrate funzionanti tenendo conto della forma dell'oggetto sul tavolino nei quattro casi enunciati e per diversi valori di β.

Il risultato fondamentale che si evince dalla Figura è che, a differenza del controllore con codifica localistica delle primitive motorie, in questo caso l'apprendimento del modello di base produce già risultati positivi, in quanto alcune delle reti neurali

addestrate sono in grado di guidare il robot ad eseguire correttamente il compito proposto. In particolar modo si può dedurre che il valore di β migliore è $\beta = 30$, con il quale nella maggior parte dei casi si ottengono le percentuali maggiori di reti addestrate funzionanti.

Per quanto concerne il caso di esecuzione parallela delle primitive, questo si differenzia da quello nel paragrafo 4.1.1.3 in quanto non vengono attivate parallelamente tutte le primitive opportunamente pesate ma solamente quelle relative ai rettangoli di un'intersezione se l'uscita della rete cade nell'intersezione stessa. Nello specifico, le primitive relative all'intersezione vengono attivate con ugual peso: se l'intersezione è generata da due rettangoli, quindi due primitive, la velocità di esecuzione di ciascuna viene dimezzata, se l'intersezione è costituita da tre rettangoli, viene divisa per 3, e così procedendo. Come si evince dalla Figura, l'anticipazione motoria non ha apportato miglioramenti degni di nota rispetto al caso dell'apprendimento del modello di base.

Per quanto riguarda l'architettura con connessioni ricorrenti, questa si è rivelata fallimentare rispetto agli altri casi praticamente per tutti i valori di β provati.

Infine, l'apprendimento anticipatorio con elaborazione del training set ha invece aumentato le percentuali di successo, seppur di poco, nella maggior parte dei casi. In queste simulazioni il training set è rielaborato in modo tale da associare alla primitiva successiva gli ultimi punti relativi alla primitiva corrente in corrispondenza delle transizioni. La soglia in base alla quale decidere quanti punti anticipare è stata scelta anche in questo caso empiricamente ed è minore di quella utilizzata nelle corrispondenti simulazioni dell'Architettura 1, in quanto per l'Architettura 2 il modello di base riesce già a risolvere il problema in alcuni casi. Nello specifico, vengono processate tante righe del training set quante sono quelle in cui lo stato di completamento della primitiva corrente è maggiore dello stato stesso in corrispondenza della transizione abbassato di un sessantesimo del suo valore.

Per effettuare un'analisi più dettagliata, in Figura 4.22 vengono riportate le

percentuali di funzionamento delle reti neurali addestrate nei quattro casi di maggiore interesse con $\beta = 30$, valore ritenuto migliore, e per i diversi oggetti sul tavolo: cubo, cilindro e sfera. In questo contesto sono state ritenute funzionanti tutte le reti che

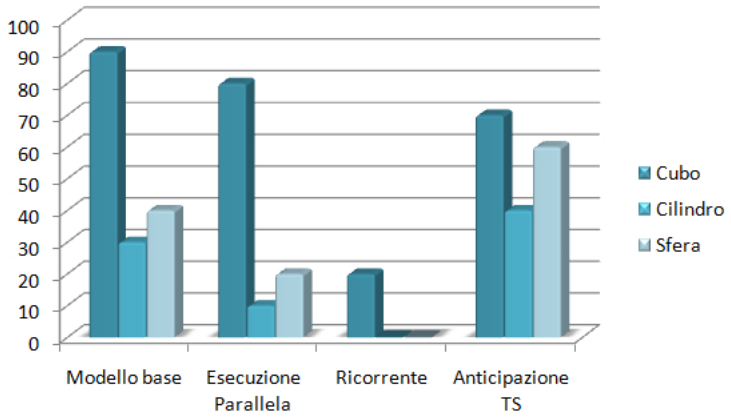

Figura 4.22: Percentuale di reti addestrate funzionanti nei quattro casi di interesse con $\beta = 30$ e i diversi oggetti sul tavolino: cubo, cilindro e sfera.

riescono a portare a termine il compito con l'oggetto di interesse, seppur dopo tempi elevati. Come si evince dalla Figura, sebbene il modello di base riesca a fornire le percentuali di funzionamento maggiori nel caso in cui l'oggetto sul tavolino sia di forma cubica, le reti neurali addestrate con rielaborazione del training set hanno una maggiore capacità di gestire le transizioni tra le primitive nel caso in cui l'oggetto sul tavolino sia una sfera o un cilindro. Per questo motivo la percentuale complessiva riportata in Figura 4.21 relativa all'anticipazione nel training set risulta maggiore rispetto a quella del modello di base. Sembra quindi che anche in queste simulazioni l'anticipazione possa comunque aiutare a gestire le transizioni tra le primitive, specialmente nei casi minormente trattati nella fase di dimostrazione. Tuttavia, osservando la percentuale relativa alle reti che riescono a gestire le transizioni con il cubo sul tavolino nel modello base, ci si aspetta che aumentando il numero di trial nel training set per il cilindro e la sfera aumentino le percentuali di funzionamento relative nel modello base in Figura

4.22 e di conseguenza anche quelle complessive per il modello base in Figura 4.21.

4.2.2 Analisi del comportamento autonomo del robot con un controllore addestrato con modello base

In questo paragrafo si illustra come l'output di categorizzazione di una rete neurale addestrata con il modello di base vari durante un trial tipico in cui il robot si muove autonomamente. In particolare si considera un esempio di rete neurale addestrata in grado di gestire correttamente le transizioni tra le primitive con un cubo sul tavolino, con un fattore di normalizzazione dei pesi delle connessioni $\beta = 30$. A tal proposito, nella Figura 4.23 i rettangoli rosso, fucsia, blu e verde rappresentano le aree di categorizzazione associate alle primitive motorie **REACH**, **CLOSE-HAND**, **LIFT** e **OPEN-HAND** rispettivamente, mentre la linea indica l'andamento dei due output di categorizzazione nel tempo sulla porzione di spazio cartesiano $[0, 1] \times [0, 1]$. Il colore della linea in Figura indica il tipo di primitiva attivata sulla base della posizione corrente dei due output di categorizzazione rispetto alle aree di categorizzazione e ai loro centri di massa (gli asterischi in Figura). Osservando la Figura, in questo caso così come in tutte le altre simulazioni che danno luogo a reti neurali funzionanti, si nota che lo stato dei due output di categorizzazione varia in modo continuo durante il trial e tende ad assumere valori significativamente diversi durante la fase iniziale, centrale e finale dell'esecuzione di una primitiva. Inoltre, è possibile notare che la disposizione dei rettangoli tende a riorganizzarsi durante il processo di apprendimento in modo tale da avere accostati o parzialmente sovrapposti i rettangoli relativi a primitive successive. E' proprio grazie a questa tendenza a riorganizzare i rettangoli che anche il semplice apprendimento del modello di base può avere successo. Infatti, tipicamente alla fine di ogni primitiva l'output di categorizzazione si ritrova al confine del rettangolo relativo alla primitiva corrente, o comunque in una zona di sovrapposizione, e bastano perciò piccole variazioni degli stimoli sensoriali in ingresso per spostare l'output di

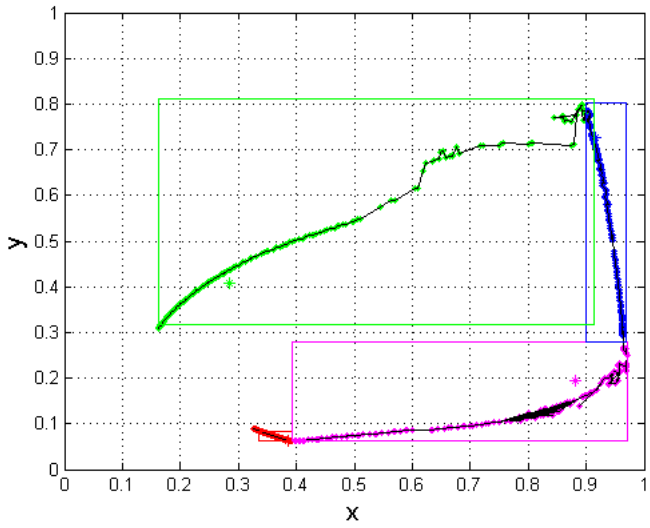

Figura 4.23: Andamento dell'output di categorizzazione in fase di test per un esempio tipico.

categorizzazione e fare in modo che venga attivata la primitiva successiva, diminuendo così i casi in cui si vengono a creare situazioni di stallo.

4.3 Test di generalizzazione e confronto

Per completezza di esposizione, in questa sezione vengono sinteticamente indagate le capacità di generalizzazione delle reti neurali che hanno la percentuale complessiva di successo maggiore per entrambe le architetture oggetto di sperimentazione: le reti con anticipazione a gradino e a rampa nel training set per quanto concerne l'Architettura 1 e le reti con $\beta = 30$ e anticipazione nel training set per l'Architettura 2. In questo contesto, anche riuscire a riprodurre il comportamento corretto autonomamente nelle stesse condizioni in cui è stata effettuata la dimostrazione implica una capacità di geneneralizzare in quanto gli stati sensoriali esperiti in fase di test tipicamente non sono identici a quelli esperiti durante la fase di dimostrazione, per esempio a causa del fatto che piccole differenze durante le fasi di transizione possono produrre

differenze significative negli stati sensoriali esperiti successivamente. Tuttavia, in questa sezione per generalizzazione si intende la capacità di esibire lo stesso tipo di comportamento in condizioni significativamente diverse da quelle presentate durante la fase di dimostrazione.

Poiché le analisi effettuate hanno dimostrato che le parti più critiche del compito proposto sono le transizioni tra le varie primitive, si è pensato di verificare le capacità di generalizzazione delle reti funzionanti di ciascuna categoria suddetta proprio in corrispondenza delle transizioni. A tal fine, considerando le informazioni sensoriali rilevanti per i controllori neurali implementati, si è pensato di effettuare le seguenti quattro tipologie di variazioni:

1. modificare la configurazione di partenza del braccio del robot in modo che il palmo della mano si discosti dalla posizione originale in un range di circa $10cm$ lungo i tre assi, per verificare se il controllore riesce comunque ad attivare la prima primitiva della sequenza, ovvero **REACH**. In questo contesto sono state scelte 3 posizioni di partenza alternative;

2. modificare la posizione dell'oggetto sul tavolino e di conseguenza la posizione desiderata della primitiva **REACH** per indagare le capacità di generalizzazione alla transizione tra la primitiva **REACH** e la primitiva **CLOSE-HAND**. In questo caso sono state scelte solamente 2 posizioni alternative a causa di difficoltà tecniche legate al fatto che risulta piuttosto complicato posizionare il palmo della mano parallelamente al piano del tavolo spostando manualmente i giunti del braccio ed evitando di muovere il busto del robot;

3. modificare le dimensioni dell'oggetto sul tavolino per analizzare le capacità di generalizzazione alla transizione tra le primitive **CLOSE-HAND** e **LIFT**. A tal proposito sono stati utilizzati oggetti le cui dimensioni sono state aumentate del 20% e analogamente diminuite del 20%. Per semplicità in queste fase viene

considerato il caso di oggetto maggiormente presentato in fase di dimostrazione, ovvero il cubo sul tavolino;

4. modificare la posizione desiderata della primitiva **LIFT** in modo che il palmo della mano vari la sua posizione in un range di circa $10cm$ lungo i tre assi rispetto alla posizione originale, per verificare se il controllore riesce comunque ad attivare l'ultima primitiva della sequenza, ovvero **OPEN-HAND**. Anche in questo caso sono state testate 3 posizioni finali alternative.

A tale proposito nella Figura 4.24 vengono riportate le percentuali mediate di reti addestrate funzionanti in grado di generalizzare per ciascuna delle quattro tipologie di modifiche suddette, per le tre categorie di reti addestrate di interesse. Con il termine di percentuale mediata in questo contesto si intende la percentuale che si ottiene facendo la media delle percentuali di reti addestrate funzionanti in grado di gestire correttamente le transizioni a seguito delle diverse variazioni effettuate per ogni tipologia. Come si evince dalla Figura l'analisi delle capacità di generalizzazione

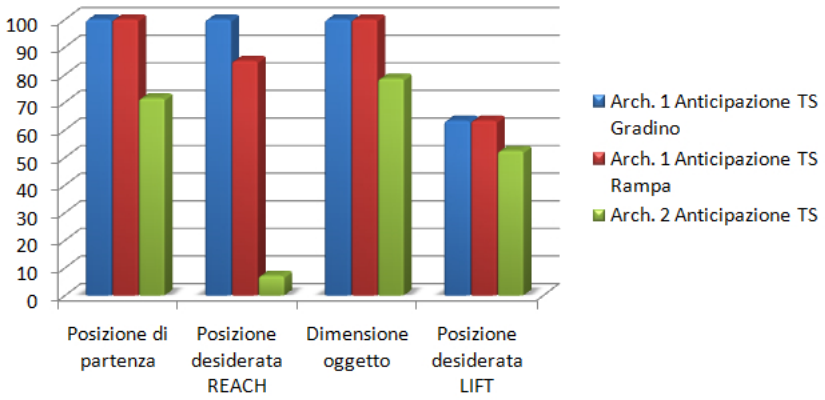

Figura 4.24: Percentuale mediata delle reti in grado di generalizzare per le quattro tipologie di modifiche effettuate e per le 3 categorie di reti analizzate: Architettura 1 con anticipazione a gradino nel training set, Architettura 1 con anticipazione a rampa nel training set e Architettura 2 con anticipazione nel training set.

condotta sulle 10 reti neurali addestrate con Architettura 1 e anticipazione a gradino nel training set, ha prodotto risultati molto soddisfacenti: tali reti hanno difficoltà a generalizzare solamente in alcuni casi per la transizione tra la primitiva **LIFT** e **OPEN-HAND**. Analogamente, anche le 10 reti neurali addestrate con Architettura 1 e anticipazione a rampa nel training set hanno prodotto buoni risultati, seppur di poco inferiori a quelli con anticipazione a gradino. Per quanto concerne invece l'analisi delle capacità di generalizzazione delle sole reti addestrate funzionanti con Architettura 2 e anticipazione nel training set, come si può osservare dalla Figura queste si sono rivelate possedere capacità di generalizzazione inferiori rispetto ai due casi precedenti. Tale risultato rispecchia d'altronde il fatto che le reti addestrate con Architettura 1 e anticipazione nel training set riescono a gestire correttamente le transizioni tra le primitive indipendentemente dalla forma dell'oggetto sul tavolino, anche se al cilindro e alla sfera è dedicato solamente 1 trial nel training set, mentre le reti neurali con Architettura 2 e anticipazione nel training set risultano più sensibili alle variazioni e quindi meno in grado di generalizzare anche per gli esempi scarsamente trattati in fase di dimostrazione.

Si noti infine che naturalmente è lecito aspettarsi un aumento delle capacità di generalizzazione all'aumentare della variabilità delle condizioni esperite durante la fase di dimostrazione.

4.4 Considerazioni finali

In questo Capitolo sono stati presentati i risultati ottenuti dalle simulazioni relative all'apprendimento dei due controllori neurali in Figura 3.5: a sinistra l'Architettura 1 con codifica localistica delle primitive motorie, a destra l'Architettura 2 con codifica distribuita.

In primo luogo, occorre notare che l'Architettura 1, seppur funzionante nel 100% dei

casi grazie all'anticipazione, è poco scalabile in quanto presenta lo svantaggio di aver bisogno di un numero di neuroni di uscita pari al numero di primitive con cui si vuole lavorare. L'aumento del numero di primitive provocherebbe un aumento corrispondente del numero di neuroni di uscita e di conseguenza del numero di neuroni interni, causando quindi un allargamento dell'architettura. E' proprio per poter superare questa limitazione che si è pensato all'implementazione di un'architettura alternativa con codifica distribuita delle primitive.

Dopo aver presentato i risultati relativi a tutte le simulazioni effettuate per i due controllori oggetto di sperimentazione, è possibile fare alcune osservazioni. Un risultato importante è che, mentre nel caso dell'Architettura 1 l'apprendimento del modello di base non ha successo, nel caso dell'Architettura 2 lo stesso ha dato luogo a risultati parzialmente soddisfacenti. In particolare si è visto che la codifica localistica necessita in tutti i casi di una forma di anticipazione, sia che essa nasca come effetto spontaneo tramite l'esecuzione parallela delle primitive, sia che essa sia imposta tramite un meccanismo di rielaborazione del training set realizzato appositamente. Negli approcci con codifica distribuita invece, considerando il caso in cui sul tavolino sia presente l'oggetto maggiormente utilizzato nella fase di dimostrazione, ovvero il cubo, è la disposizione delle aree di categorizzazione che facilita le transizioni tra le diverse primitive senza bisogno di alcun meccanismo di anticipazione. Tuttavia, anche in questo caso l'anticipazione risulta vantaggiosa quando il robot si trova ad affrontare situazioni poco familiari, come ad esempio oggetti cilindrici o sferici, esperiti una sola volta durante il processo di dimostrazione e addestramento.

Uno degli svantaggi dell'apprendimento del modello base del controllore con codifica distribuita è che anche con il cubo sul tavolino, in alcuni casi prima che il sistema riesca ad effettuare la transizione tra la primitiva corrente e la successiva sono necessari molti cicli di simulazione, in quanto gli stimoli sensoriali alle transizioni rimangono comunque molto simili.

Un'ulteriore differenza che è stata riscontrata nei due approcci è legata all'analisi condotta per indagare gli effetti della normalizzazione dei pesi delle connessioni tra i neuroni. Nel caso dell'Architettura 1, da quanto si è visto, non ci sono problemi legati alle regioni di saturazione delle funzioni d'attivazione logistiche, anzi i neuroni sono spinti a comportarsi come interruttori sfruttando proprio queste zone. Nel caso dell'Architettura 2 invece, il problema è molto delicato, in quanto la disposizione e la forma dei rettangoli dipende molto dalle zone della funzione logistica in cui l'algoritmo di backpropagation porta i neuroni a lavorare.

Inoltre, un grande svantaggio dell'Architettura 2 è che l'apprendimento in questo tipo di approccio è fortemente dipendente dalla configurazione iniziale, ovvero dalla rete inizializzata in modo casuale utilizzata per individuare i rettangoli iniziali. Tale dipendenza è uno dei motivi principali che limita le percentuali di funzionamento delle reti addestrate, mostrate in Figura 4.21 e 4.22.

Infine, a seguito dell'analisi delle capacità di generalizzazione delle reti neurali addestrate effettuata nella sezione precedente, si può affermare che il controllore neurale con codifica distribuita delle primitive motorie presenta minori capacità di generalizzazione rispetto al controllore neurale con codifica localistica.

Capitolo 5

Esperimento sull'iCub

Per convalidare i risultati ottenuti in simulazione si è deciso di effettuare gli esperimenti direttamente sull'iCub disponibile presso il laboratorio LARAL dell'ISTC. Poiché l'iCub in questione attualmente non dispone di sensori di tatto, è stato necessario testare delle versioni semplificate delle reti neurali, prive cioè del gruppo di neuroni di ingresso che codificano le informazioni tattili.

Come tipologia di esperimenti si è deciso di testare sul robot reale reti addestrate in simulazione tramite training set ricavati dal simulatore stesso, in grado di gestire correttamente tutte le transizioni tra le primitive. Dalle simulazioni relative alle varianti senza informazioni tattili delle due architetture in Figura 3.5 si è visto che nel caso di codifica distribuita le informazioni relative ai contatti della mano con l'oggetto sono fondamentali per ottenere reti neurali funzionanti. In altre parole, l'assenza dei sensori di tatto non ha consentito di effettuare esperimenti per il caso di controllori neurali con codifica distribuita delle primitive motorie, in quanto neanche in simulazione è stato possibile ricavare reti in grado di gestire correttamente tutte le transizioni tra le primitive. Nel caso del controllore con codifica localistica invece, nonostante la mancanza delle informazioni tattili, le reti neurali addestrate in simulazione risultano in grado di gestire correttamente le transizioni tra le diverse primitive.

Di seguito vengono perciò riportati i risultati relativi al test delle reti con Architettura 1, modificata come illustrato in Figura 5.1, nel caso in cui il training set venga rielaborato con anticipazione a rampa o a gradino. In particolare sono state addestrate 3 reti neurali con l'Architettura di Figura 5.1 e rielaborazione del training set con anticipazione a gradino e 3 con anticipazione a rampa.

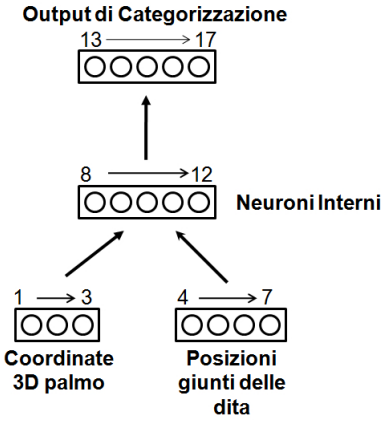

Figura 5.1: Architettura del controllore neurale utilizzato nell'esperimento sull'iCub. I blocchi evidenziati rappresentano i gruppi di neuroni che hanno le stesse caratteristiche; le frecce indicano le connessioni tra i blocchi: ciascun neurone del blocco di partenza è connesso con tutti i neuroni del blocco di arrivo.

Poiché il software di controllo è stato scritto in modo da poter funzionare utilizzando le interfacce YARP, sia per quanto concerne l'acquisizione delle informazioni sensoriali sia per la gestione dei comandi per gli attuatori, non è stata necessaria alcuna modifica per poter passare dalle simulazioni agli esperimenti sull'iCub reale. Nello specifico l'utente può ancora interagire con il robot tramite la stessa interfaccia grafica descritta nella sezione 3.7.

Gli esperimenti relativi alla validazione delle reti neurali addestrate in simulazione sull'iCub reale hanno avuto successo: il robot riesce a gestire correttamente tutte le transizioni tra le primitive e a completare il compito desiderato, nonostante la povertà di

informazioni sensoriali in ingresso al controllore. Il robot riesce quindi a generalizzare le proprie capacità rispetto alle inevitabili differenze tra le esperienze senso-motorie simulate esperite durante la fase di addestramento e quelle reali esperite durante la fase di test in hardware.

Occorre tuttavia notare che, poiché l'iCub disponibile presso il laboratorio non è aggiornato alla versione finale, mancano sensori in grado di leggere in modo preciso la posizione di ciascun giunto delle dita. Quest'ultima è infatti ottenuta da alcuni sensori accoppiati ai motori che controllano ciascun dito, posizionati a distanza rispetto alla mano e perciò poco accurati. Le imprecisioni nei valori restituiti dai sensori propriocettivi delle dita della mano influenzano principalmente l'esecuzione della primitiva **CLOSE-HAND** che raggiunge una posizione differente dalla posizione desiderata, complicando di conseguenza la presa dell'oggetto. Ad esempio il pollice, che in linea di principio durante l'esecuzione dovrebbe opporsi in corrispondenza dell'indice e dell'anulare, tende ad opporsi in corrispondenza delle ultime due dita, dando luogo quindi ad una presa malsicura. Nella Figura 5.2 è possibile osservare una serie di scatti dell'iCub mentre agisce autonomamente nell'ambiente riproducendo correttamente il compito precedentemente dimostrato, grazie all'utilizzo di un controllore neurale con l'architettura di Figura 5.1, addestrato con anticipazione a gradino nel training set.

Per completezza sono state infine indagate le capacità di generalizzazione delle suddette 3 reti addestrate con anticipazione a gradino e delle 3 con anticipazione a rampa testate sull'iCub reale. In particolare sono state verificate solo tre delle quattro tipologie di modifiche presentate nella sezione 4.3, ovvero: modifica della posizione di partenza, modifica della posizione desiderata della primitiva **REACH** e modifica della posizione desiderata della primitiva **LIFT**. In questi esperimenti le dimensioni dell'oggetto non sono state modificate in quanto le reti non dispongono di neuroni che codificano le informazioni di tatto e le letture delle posizioni di giunto delle dita sono piuttosto imprecise. Le percentuali di reti in grado di generalizzare sono riportate

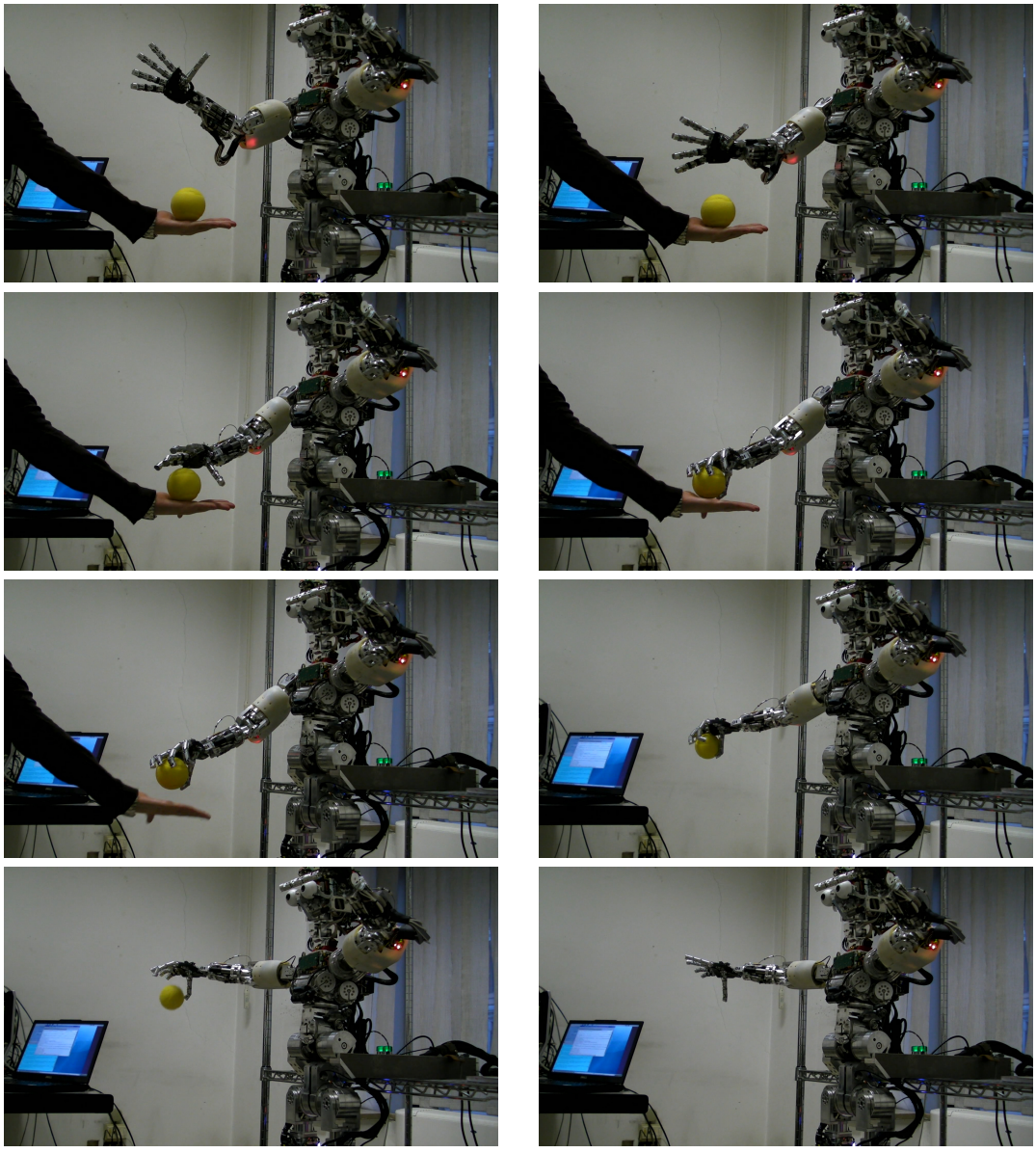

Figura 5.2: Scatti dell'iCub mentre sta riproducendo autonomamente il compito dimostrato grazie ad un controllore neurale con l'architettura di Figura 5.1, addestrato con un training set rielaborato con anticipazione a gradino.

in Figura 5.3, mediate su tutte le variazioni effettuate per ogni tipologia di modifica. Come si evince dalla Figura, i test di generalizzazione condotti sull'iCub conducono a

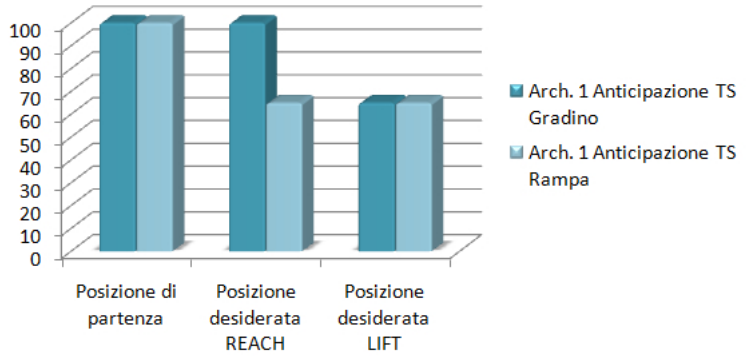

Figura 5.3: Percentuale mediata delle reti in grado di generalizzare per 3 delle quattro tipologie di modifiche effettuate, per le reti con l'Architettura di Figura 5.1 con rielaborazione del training set a gradino e a rampa.

percentuali di reti addestrate in grado di generalizzare simili a quelle presentate nella sezione 4.3 in Figura 4.24. Il fatto che le reti per lo più non riescano a generalizzare sulla modifica della posizione della primitiva **LIFT** è legato alla povertà di informazioni sensoriali in ingresso al controllore che rende le informazioni relative alle coordinate del palmo della mano piuttosto influenti. In altre parole, specialmente quando la posizione desiderata della primitiva **LIFT** viene modificata in modo tale che risulti spazialmente più vicina a quella di partenza, i controllori neurali si ritrovano in situazioni ambigue in cui non sono in grado di capire correttamente quale primitiva attivare tra le primitive **LIFT** e **REACH**. In questi casi il sistema entra in condizioni di stallo in cui vengono attivate alternativamente entrambe le primitive.

Capitolo 6

Conclusioni e sviluppi futuri

In questa tesi è stato presentato un sistema in grado di consentire ad un robot umanoide di acquisire, attraverso un processo di apprendimento per dimostrazione, la capacità di esibire il comportamento appropriato al contesto corrente e la capacità di esibire comportamenti complessi, combinando una serie di comportamenti elementari in sequenza o in parallelo.

In particolare, all'inizio del processo di addestramento il robot è dotato di una serie di primitive motorie, ognuna delle quali corrispondente ad un certo comportamento elementare. Durante la fase di dimostrazione il robot viene controllato dallo sperimentatore attraverso l'esecuzione di comandi "linguistici", costituiti da parole quali "reach" o "lift", che elicitano l'esecuzione delle primitive motorie corrispondenti. Tale processo di dimostrazione viene utilizzato per generare un training set che consiste nella sequenza degli stati sensoriali esperiti dal robot e delle primitive motorie eseguite a seguito dei comandi linguistici prodotti dallo sperimentatore. Il training set così generato viene poi utilizzato per addestrare una rete neurale artificiale attraverso l'algoritmo di backpropagation ad associare gli stati sensoriali esperiti in fase di dimostrazione alle primitive motorie corrispondenti. Alla fine del processo di addestramento infine, il robot dotato del sistema di controllo addestrato viene valutato

per la capacità di esibire autonomamente i comportamenti dimostrati in interazione con l'ambiente senza ricevere i comandi linguistici, nonché per la capacità di generalizzare i comportamenti appresi in condizioni diverse. Per queste caratteristiche il modello presenta degli aspetti originali rispetto ai lavori precedenti, con particolare riferimento a quelli di Zhang e Weng in [1] e Dominay et al. in [2], che si propongono un obiettivo simile ma che si limitano a prendere in considerazione la capacità di elicitare una serie di comportamenti elementari in sequenza, indipendentemente dal contesto senso-motorio.

L'analisi dei risultati ottenuti in questa tesi mostra come, in alcuni dei setup sperimentali utilizzati, il robot riesca ad acquisire le capacità richieste e a generalizzare in condizioni diverse. Inoltre, l'analisi del comportamento esibito dall'iCub reale sulla base del sistema di controllo con controllore neurale addestrato in simulazione dimostra che il modello è in grado di generalizzare anche rispetto alle inevitabili differenze riscontrabili tra l'ambiente simulato e quelle reale.

In generale, l'analisi dei risultati ottenuti mostra come la difficoltà principale di tale sistema di apprendimento risieda nella gestione delle transizioni tra le primitive. Ciò è principalmente dovuto a due ragioni: da una parte gli stati sensoriali esperiti a cavallo delle transizioni sono molto simili o addirittura indistinguibili tra loro, dall'altra le informazioni utilizzate dallo sperimentatore per effettuare le transizioni in fase di dimostrazione sono tipicamente diverse da quelle utilizzate dal robot per via del diverso punto di vista. Per tali ragioni il sistema tende a ritrovarsi in situazioni di stallo in cui il robot non riesce ad effettuare le transizioni tra le diverse primitive motorie.

L'analisi comparativa di una serie di simulazioni effettuate variando l'architettura della rete neurale, la codifica delle primitive motorie, e le caratteristiche del training set, indica che tali difficoltà possono essere affrontate in due modi: spingendo il robot ad anticipare la transizione alla primitiva successiva, oppure utilizzando una codifica distribuita e auto-organizzata delle categorie, del tipo di quella utilizzata da Tuci et al. in [27] nell'ambito degli algoritmi evolutivi.

Il ruolo dell'anticipazione può essere spiegato con il fatto che il tipo di stimoli sensoriali esperiti dal robot dipende dal tipo di azioni eseguite dal robot stesso e di conseguenza l'esecuzione anticipata della primitiva successiva favorisce esperienze di stimoli sensoriali fortemente associati a tale primitiva e facilita quindi la transizione. In particolare, le simulazioni effettuate mostrano come tale processo di anticipazione possa essere realizzato sia consentendo al robot di attivare più primitive in parallelo sia rielaborando il training set in modo tale da spingere il robot ad anticipare l'attivazione della primitiva successiva in corrispondenza delle transizioni.

Il ruolo della codifica auto-organizzata può essere invece spiegato considerando che la possibilità di auto-organizzare le aree che corrispondono alle diverse primitive motorie consente di porre le zone di transizione al confine di aree adiacenti. Il ruolo della codifica distribuita, in cui una primitiva motoria è rappresentata da un area in uno spazio multidimensionale piuttosto che da un punto, può essere spiegato con il fatto che questa permette di rappresentare fasi diverse dell'esecuzione della primitiva con zone diverse, favorendo dunque le transizioni tra le fasi al confine tra le diverse primitive.

Poiché il modello proposto si presenta quindi come un approccio originale nell'ambito dei sistemi di acquisizione di capacità complesse a partire da capacità elementari nel contesto dell'apprendimento per dimostrazione, molte sono le direzioni in cui si può procedere per l'analisi e lo sviluppo di sistemi più efficienti. Da una parte si può agire sull'implementazione delle primitive motorie, al fine di renderle più sofisticate ed eventualmente parametriche, in modo che la loro combinazione possa dar luogo a compiti più complessi, sia discreti che periodici. In questo contesto si colloca ad esempio l'introduzione di un modulo di visione che consenta al robot di acquisire le informazioni sulle posizioni o sulle caratteristiche degli oggetti e su eventuali ostacoli presenti nell'ambiente. Dall'altra parte è possibile procedere cercando di ottimizzare il processo di apprendimento, prevedendo ad esempio una fase di apprendimento incrementale successiva più sofisticata in cui vengono utilizzate tecniche di apprendimento per

rinforzo per spingere il processo di apprendimento nelle direzioni desiderate.

Un'estensione interessante di quanto proposto potrebbe prevedere dei moduli di processamento che stabiliscano i pesi in base ai quali attivare le diverse primitive negli esperimenti con esecuzione parallela secondo una qualche euristica. Inoltre, per quanto concerne l'approccio con codifica distribuita delle primitive motorie, potrebbe essere interessante indagare delle soluzioni migliorative che posseggano maggiori capacità di generalizzazione e nel contempo siano meno dipendenti dalle condizioni iniziali.

Infine, per convalidare quanto proposto in questa tesi si può pensare di inserire un modulo di processamento del linguaggio naturale in modo da riconoscere i comandi linguistici dettati dall'utente nella fase di dimostrazione.

Quelli sopra elencati sono solo alcuni dei miglioramenti più immediati che è possibile immaginare per aumentare l'efficienza del sistema e per consentire una maggiore potenzialità di utilizzo, permettendo l'esecuzione di comportamenti più complessi e l'impiego di interfacce più user-friendly.

Bibliografia

[1] Y. Zhang, J. Weng, "Task transfer by a Developmental Robot", *IEEE Transactions on evolutionary computation*, vol. 11, no. 2, Aprile 2007.

[2] P.F. Dominey, A. Mallet, E. Yoshida, "Real-Time spoken-language programming for cooperative interaction with a humanoid apprentice", *International Journal of Humanoid Robotics*, vol. 6, no. 2, pp. 147-171, 2009.

[3] D. Floreano, C. Mattiussi, "Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies", Cambridge, MA: MIT Press, 2008.

[4] R.S. Sutton, A.G. Barto, "Reinforcement learning: An introduction", The MIT Press, Cambridge, MA, London, England, 1998.

[5] B.D. Argall, S. Chernova, M. Veloso, B. Browning, "A survey of robot learning from demonstration", *Robotics and Autonomous Systems (2009)*, doi: 10.1016/j.robot.2008.10.024.

[6] A. Billard, S. Callinon, R. Dillmann, S. Schaal, "Robot programming by demonstration", *Philosophical Transactions: Biological Sciences*, vol. 358, no.1431, pp. 537-547, Marzo 2003.

[7] S. Schaal, A. Jjspeert, A. Billard, "Computational Approaches to Motor Learning by Imitation", in: B. Siciliano, O. Khatib (Eds.), *Handbook of Robotics*, Springer, New York, NY, USA, 2008 (Capitolo 59).

[8] S. Schaal, "Is imitation learning the route to humanoid robots?", *Trends in Cognitive Sciences*, vol. 3, no. 6, pp. 233-242, 1999.

[9] C. Breazeal, B. Scassellati, "Robots that imitate humans", *Trends in Cognitive Sciences*, vol. 6, no. 11, pp. 481-487, 2002.

[10] P. Kormushev, S. Calinon, D.G. Caldwell, "Robot Motor Skill Coordination with EM-based Reinforcement Learning", in: *Proceedings of the IEEE/RSJ Intl Conf. on Intelligent Robots and Systems (IROS'10)*, 2010.

[11] S. Muench, J. Kreuziger, M. Kaiser, R. Dillman, "Robot Programming by Demonstration (RPD) - Using Machine Learning and User Interaction Methods for the Development of Easy and Comfortable Raobot Programming Systems", in: *Proceedings of the 24th International Symposium on Industrial Robots (ISIR'94)*, pp. 685-693, 1994.

[12] A. Alissandrakis, C. L. Nehaniv, K. Dautenhahn, "Correspondence Mapping Induced State and Action Metrics for Robotic Imitation", *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, vol. 37, no. 2, Aprile 2007.

[13] S. Vijayakumar, S. Schaal, "Locally Weighted Projection Regression: An O(n) Algorithm for Incremental Real Time Learning in High Dimensional Space", in: *Proceedings of Seventeenth International COnference on Machine Learning (ICML2000)*, pp. 1079-1086, 2000.

[14] A.J. Ijspeert, J. Nakanishi, and S. Schaal, "Learning control policies for movement imitation and movement recognition", *Neural Information Processing System (NIPS)*, vol. 15, pp. 1547-1554, 2003.

[15] M. J. Mataric, "Sensory-Motor Primitives as a Basis for Imitation: Linking Perception to Action and Biology to Robotics", in: *Imitation in Animals and Artifacts*, The MIT Press, 2000.

[16] P.K. Pook, D.H. Ballard, "Recognizing teleoperated manipulations", in: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA'93)*, 1993.

[17] P.E. Rybski, K. Yoon, J. Stolarz, M. Veloso, "Interactive robot task training through dialog and demonstration", in: *Proceedings of the 2nd ACM/IEEE International Conference on Robotics and Automation (ICRA'07*, 2007.

[18] S. Lauria, G. Bugmann, T. Kyriacou, E. Klein, "Mobile robot programming using natural language", *Robotics and Autonomous Systems*, vol.38, pp. 171-181, 2002.

[19] P. J. Werbos, "Backpropagation Through Time: What It Does and How to Do It", in: *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1550-1560, Ottobre 1990.

[20] M. Ito, K. Noda, Y. Hoshino, J. Tani, "Dynamic and interactive generation of object handling behaviors by a small humanoid robot using a dynamic neural network model", *Neural Networks*, vol. 19, no. 3, pp. 323-337, 2006.

[21] S. Schaal, "Dynamic Movement Primitives - A Framework for Motor Control in Humans and Humanoid Robotics", in: *The International Symposium on Adaptive Motion of Animals and Machines*, 2003.

[22] J. Yang, Y. Xu, C.S. Chen, "Hidden Markov model approach to skill learning and its application to telerobotics", *IEEE Transactions on Robotics and Autonomation*, vol. 10, pp. 621-631, Ottobre 1994.

[23] J. Aleotti, S. Caselli, "Robust trajectory learning and approximation for robot programming by demonstration", *Robotics and Autonomous Systems*, vol. 54, pp. 409-413, 2006.

[24] J. Weng, "Developmental Robotics: Theory and Experiments", *International Journal of Humanoid Robotics*, vol. 1, no. 2, 2004.

[25] G. Sandini, G. Metta, and D. Vernon, "The iCub Cognitive Humanoid Robot: An Open-System Research Platform for Enactive Cognition", in M. Lungarella et al. Eds. 50 *Years of AI*, Festschrift, LNAI 4850, Springer-Verlag, Heidelberg, pp. 359-370, 2007.

[26] G. Metta, P. Fitzpatrick, L. Natale, "YARP: Yet Another Robot Platform", *International Journal of Advanced Robotics Systems, special issue on Software Development and Integration in Robotics*, vol. 3, no. 1, Marzo 2006.

[27] E. Tuci, G. Massera, S. Nolfi, "Active categorical perception of object shapes in a simulated anthropomorphic robotic arm", *IEEE Transactions on evolutionary computation journal*, vol. 14, no. 6, 2010, In Press.