UNIVERSITÀ DEGLI STUDI DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

XXIII Ciclo

STUDIO E SVILUPPO DI UN SISTEMA NEUROCOMPUTAZIONALE PER L'ANALISI E LA PREDIZIONE DI SEQUENZE TEMPORALI MULTIDIMENSIONALI

Coordinatore:

Chiar.mo Prof. Marco Locatelli

Tutor:

Chiar.mo Prof. Stefano Cagnoni

Dottorando: *Federico Sassi*

Gennaio 2012

Borsa di studio finanziata dal Fondo Sostegno Giovani,
ambito di Indagine 11 (ICT e componentistica elettronica)

Alla donna della mia vita

Sommario

Elenco delle figure

Elenco delle tabelle

Introduzione

Una delle macchine più misteriose, versatili e non ancora del tutto conosciute è la nostra mente. Il nostro cervello è in grado di apprendere e riconoscere tutti i segnali che raccogliamo durante la nostra vita tramite i nostri sensi; per esempio è in grado di riconoscere una canzone, tra le mille, fin dalle prime note per poi essere in grado di proseguire "mentalmente" la sua melodia, oppure è in grado di correlare percezioni uditive, visive e olfattive per richiamare dalla memoria un luogo conosciuto.

Nondimeno il nostro cervello è ancora in grado di "stupirsi" quando accadono degli imprevisti, ovvero quando non si verificano le condizioni attese dalla nostra esperienza. Siamo seduti in una stanza con le spalle rivolte verso una porta e, sentendo un campanellino, ci giriamo ed osserviamo un gattino che si sta avvicinando. La nostra mente imparerà la correlazione tra il suono percepito e la figura osservata ed ogni volta che sentirà il suono richiamerà dalla memoria l'immagine di tale gattino.

Questo metodo di "addestramento" ci permette di scoprire quotidianamente nuove relazioni e di formare nuove associazioni mnemoniche a partire dalla nostra esperienza.

I moderni sistemi di computazione artificiale sono ancora lontani dal raggiungere la sofisticatezza del nostro cervello e, nonostante l'incremento costante delle prestazioni dei moderni elaboratori, nessun algoritmo può vantare prestazioni simili. Al contrario, esistono sensori artificiali sempre più sofisticati e, in molti casi, più efficienti di quelli umani. In particolare viene raccolta una quantità sempre maggiore di dati che descrivono il nostro ambiente, il nostro corpo o processi industriali ma per

poterne trarre vantaggio è necessario trasformare questi dati in "informazione". Tale processo, quando vi sono dati eterogenei provenienti da sorgenti differenti e correlati temporalmente, può essere estremamente complicato per un sistema artificiale, ma non per un essere umano.

Alcune delle principali sfide dei nostri tempi riguardano l'aumento della qualità della vita e la salvaguardia dell'ambiente.

Migliorare la qualità della vita, soprattutto quando si osserva un aumento della longevità media, comporta la necessità di introdurre una supervisione negli ambienti in cui abittiamo per evitare eventi traumatici oppure per *comprendere* le azioni compiute e rispondere adeguatamente ottimizzando comfort, sicurezza e risparmio energetico. In particolare è necessario introdurre sensori in grado di percepire il movimento del corpo, come accelerometri o telecamere, e predisporre un sistema computazionale in grado di estrarre l'informazione necessaria all'individuazione di *pattern* ricorrenti da memorizzare o predire per soddisfare ed anticipare i bisogni delle persone.

Ognuno di noi sta prendendo coscienza di quanto sia delicato e prezioso l'equilibrio del nostro ambiente. Questo può essere salvaguardato ad esempio mediante un monitoraggio costante delle sue condizioni. Per l'acquisizione di tali informazioni è necessario l'impiego di reti di sensori per la raccolta di parametri ambientali (temperatura, umidità, concentrazione di inquinanti, ...) su larga scala. Queste infrastrutture di raccolta dati devono essere in grado di operare anche in luoghi difficilmente accessibili e non strutturati perciò è necessario creare dei meccanismi di comunicazione altamente efficienti che permettano di instradare l'informazione raccolta con il minor dispendio energetico possibile. Infine sarà possibile analizzare l'insieme dei dati raccolti per individuare, e dunque predire, le possibili cause scatenanti di eventi naturali o artificiali.

In entrambi i casi è necessario analizzare un insieme di dati eterogeneo che descrivono l'evoluzione temporale del fenomeno di interesse. Lo stato dell'arte della ricerca non ha ancora prodotto una soluzione olistica a questi problemi perciò questa tesi si pone l'obiettivo di studiare, sviluppare e valutare in campo applicativo un

nuovo paradigma computazionale, ispirato al funzionamento del nostro cervello, che mira a replicare quelle caratteristiche di apprendimento, riconoscimento e predizione così apparentemente "semplici" ma davvero complesse da ricreare artificialmente.

Nel primo capitolo saranno introdotte ed analizzate le Hierarchical Temporal Memories (HTM), una prima implementazione di una più vasta teoria, il Memory Prediction Framework, volta a modellare il comportamento della neocorteccia. Le HTM combinano ed estendono le reti Bayesiane, algoritmi di clustering spazio-temporale all'interno di una gerarchia di nodi simile alle reti neurali, ma dal funzionamento intrinsecamente differente in quanto i singoli nodi contengono delle memorie di sequenze temporali che sono combinate tra loro gerarchicamente.

Nel capitolo successivo sarà presentato un caso applicativo delle Hierarchical Temporal Memories nell'area di ricerca denominata *Ambient Intelligence* ed in particolare riguardo al riconoscimento di movimenti, azioni ed attività umane. I sensori utilizzati per raccogliere tali informazioni spaziano da sensori indossabili (accelerometri triassiali wireless) a sensori ambientali (telecamere). Tali sensori, dopo una pre-elaborazione, forniscono un insieme di segnali monodimensionali dai quali si cercherà di individuare pattern ricorrenti, *invarianti*, tra le differenti persone ed azioni.

Nel terzo capitolo sarà introdotto un nuovo protocollo di comunicazione a bassissimo consumo per *Wireless Sensor Networks* orientate alla raccolta di dati distribuiti nell'ambiente. Tali reti di sensori possono raccogliere e trasmettere un insieme di segnali relativamente semplici per mantenere un costo e un consumo energetico molto bassi. Verrà inoltre presentato uno studio sulla classificazione di dati ambientali raccolti mediante una rete di sensori tramite le Hierarchical Temporal Memories.

Infine nell'ultimo capitolo verrà presentata una nuova architettura computazionale basata sull'implementazione delle Hierarchical Temporal Memories che le estendende per avvicinarle ulteriormente al Memory Prediction Framework descritto nel primo capitolo. In particolare tali modifiche introducono: vari algoritmi di apprendimento e di inferenza in grado di rappresentare in maniera più efficiente sequenze temporali lunghe; una suddivisione delle memorie tra "istintive" ed "apprese" ed in-

fine un meccanismo di funzionamento che permette ad una rete di eseguire predizioni di lunghezza arbitraria.

Capitolo 1

Hierarchical Temporal Memories

Prediction by analogy -creativity -
is so pervasive we normally don't notice it.

– Jeff Hawkins

1.1 Un nuovo paradigma computazionale: Memory Prediction Framework

Il Memory Prediction Framework è una teoria di funzionamento del cervello formulata da Hawkins nel libro *On Intelligence* [1] nel 2004. Tale teoria si fonda su basi della fisiologia del cervello per individuare il ruolo delle varie parti, tra cui la neocorteccia, l'ippocampo ed il talamo, nei processi cognitivi. In particolare Hawkins sostiene che il processo cognitivo si basi principalmente sul riconoscimento ed apprendimento di pattern provenienti dai nostri sensi e come questo processo porti a generare continuamente delle predizioni sulle probabili situazioni future che i nostri sensi devono attendersi. Tale meccanismo di richiamo dalla memoria permette di individuare eventi inattesi, predizioni errate, dai quali è possibile apprendere nuovi eventi o individuare eventi già conosciuti e perciò rafforzarne la loro rappresentazione.

Mountcastle ha descritto [2] come le varie parti della neocorteccia si occupino di differenti canali sensoriali o a differenti livelli, siano topologicamente molto simili; perciò è molto probabile che l'intera neocorteccia implementi un unico algoritmo comune eseguito ai differenti livelli di una gerarchia di processi. In accordo con tale teoria, ogni nodo di una rete HTM esegue all'incirca lo stesso algoritmo, indipendentemente dalla posizione del nodo o dai dati che la rete deve analizzare.

L'idea principale del Memory Prediction Framework (MPF) è quella che i segnali in ingresso siano associati e rimappati tramite una gerarchia di riconoscimento (percorso *feedforward*) e generino predizioni (o aspettative) codificate come potenziali (percorso *feedback*). All'interno dei nodi della gerarchia i segnali di *feedforward* e di *feedback* concorrono a generare le predizioni dei futuri segnali, tra cui anche quelli di ingresso. Lo stato corrente di un nodo è denominato belief del nodo. In figura 1.1 è possibile trovare una rappresentazione di una rete generica.

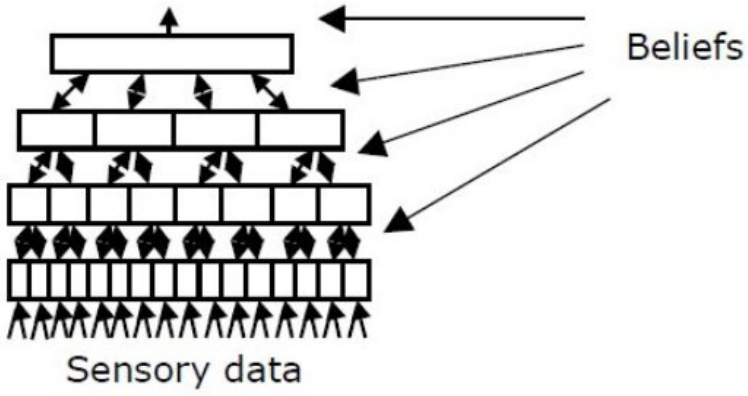


Figura 1.1: Una generica rete secondo il modello MPF

Ogni nodo della gerarchia contiene una memoria differente e quando una sequenza di ingressi viene abbinata ad una sequenza memorizzata il nodo propaga il nome di questa sequenza, riducendo così inutili dettagli e permettendo ai nodi superiori di imparare sequenze di ordine maggiore, ovvero sequenze-di-sequenze. L'obiettivo di questo processo è quello di generare degli invarianti ai livelli più alti, ovvero me-

morie indipendenti da piccole variazioni ma che rappresentano concetti più ampi. Nel caso di una non corrispondenza tra *feedforward* e *feedback* il nodo deve propagare maggiori informazioni ai nodi superiori in modo che essi possano risolvere l'incongruenza.

Consideriamo come esempio un problema di visione artificiale dove i dati di addestramento sono composti da figure geometriche e la rete può avere la struttura rappresentata in figura 1.8. I nodi ai livelli inferiori della gerarchia memorizzeranno linee ed angoli in movimento, i nodi leggermente superiori combineranno tali memorie per creare quadrati, rettangoli, cerchi di differenti dimensioni; infine i nodi ai livelli più alti astraggono tali rappresentazioni per determinarne la presenza o il comportamento generale (movimento). È possibile notare come, salendo lungo la gerarchia, i livelli differiscano per

- estensione del campo recettivo: ovvero i nodi ai livelli inferiori potrebbero "vedere" solamente una parte dei dati in ingresso mentre i nodi ai livelli più alti, ricevendo i dati da tutti i nodi sottostanti, osservano tutti i dati
- stabilità temporale: le rappresentazioni ai bassi livelli cambiano costantemente mentre la rappresentazione ad alto livello è più stabile
- astrazione: come precedentemente introdotto il processo di selezione degli invarianti lungo la gerarchia porta alla creazione di rappresentazioni astratte slegate dal contesto specifico.

Il *Memory Prediction Framework* è una teoria recente che contiene ancora vari punti non del tutto chiariti, in particolare non definisce molto dei dettagli di funzionamento di varie parti del cervello e manca di dettagli specifici su una possibile trattazione matematica. In particolare, quest'ultimo problema è lasciato alle varie implementazioni derivate da tale teoria.

1.1.1 Un esempio: visione artificiale

Per comprendere la prospettiva del *Memory Prediction Framework*, consideriamo un esempio legato al mondo della visione. Una persona è in grado di riconoscere un

certo soggetto in un'immagine nonostante cambi di dimensione, ubicazione, condizioni di illuminazione o deformazioni. Questo compito, che una persona è in grado di svolgere in modo istantaneo e naturale, diventa estremamente complesso se vogliamo creare un algoritmo al calcolatore che replichi tale comportamento. Molte tecniche sono state proposte in questo senso, ma ancora nessuna si è dimostrata in grado di generalizzare a tal punto da poter riconoscere correttamente oggetti anche in scene mai viste in modo pienamente soddisfacente. La causa di questo fallimento va probabilmente ricercata nel fatto che a lungo gli scienziati hanno ignorato il ruolo del tempo legato alla visione. Poiché una persona è in grado di riconoscere un oggetto anche a partire da una singola immagine, il contributo del tempo è stato considerato spesso superfluo. Tuttavia i mammiferi utilizzano le informazioni raccolte da successioni di più immagini che variano nel tempo per astrarre caratteristiche di carattere generale dell'oggetto che permettono poi loro il riconoscimento di uno stesso elemento anche a partire da immagini statiche. I problemi legati al riconoscimento di modelli noti all'interno di immagini o sequenze video vengono solitamente trattati con algoritmi di apprendimento supervisionato, in cui si addestra il sistema fornendo ad esso esempi degli oggetti che si vogliono riconoscere associatati ad una etichetta. Tuttavia, se si considera con maggiore attenzione il processo di apprendimento dei mammiferi, si può concludere che il problema della visione ha in realtà una natura non supervisionata.

Consideriamo ad esempio un cane che si sta avvicinando alla sua ciotola del cibo. Ad ogni passo compiuto, sulla retina del cane si forma una diversa immagine della medesima ciotola. Possiamo pensare all'immagine che si crea sulla retina come ad una immagine composta da pixel. La distanza tra due pixel omologhi in due immagini successive può essere anche notevole. Tuttavia, il cane è consapevole che quella che ha di fronte a sè e sempre la stessa ciotola nonostante non abbia una etichetta ciotola. A causa del moto relativo tra il cane e la ciotola, sulla retina del cane si formano continuamente nuove immagini rappresentanti lo stesso oggetto e data la consequenzialità temporale egli comprende che si tratta di rappresentazioni dello stesso oggetto. Ne consegue che il tempo sta agendo da supervisore implicito[3].

Facendo un ulteriore passo avanti possiamo pensare al problema di identifica-

zione di un oggetto come al problema di individuare una rappresentazione invariante dell'oggetto, ovvero di identificare le caratteristiche costanti (invarianti) tra gli oggetti appartenenti alla stessa categoria. Tornando all'esempio precedente, una volta che il cane ha imparato il concetto di ciotola sarà in grado di riconoscere le maggior parte delle altre ciotole. Inoltre man mano che osserverà altre ciotole, queste concorreranno ad accrescere la sua rappresentazione di ciotola.

Consideriamo ora un ulteriore esempio, che sfrutta la natura gerarchica del modello. Se osserviamo un animale appartenente ad una razza sconosciuta e riconosciamo una bocca e dei, denti possiamo predire che quell'animale mangia con la bocca e che potrebbe essere una minaccia. La nostra neocorteccia ha memorizzato i denti e la bocca come un invariante proprio di molti tipi di animali. Allo stesso modo, una volta che i livelli più bassi nella gerarchia avranno imparato i blocchi invarianti, i livelli superiori della gerarchia vedranno un nuovo oggetto come una ricombinazione degli invarianti del livello precedente. Tale comportamento permette un grande risparmio di memoria e una semplificazione del processo di addestramento. Per questo motivo le memorie all'interno di una gerarchia sono composta da *sequenze di sequenze*.

1.2 Una introduzione alle Hierarchical Temporal Memories

Le Hierarchical Temporal Memories (HTM) sono una specifica implementazione di una sottoparte del Memory Prediction Framework. Tale implementazione combina ed estende le reti Bayesiane con algoritmi di clustering spazio-temporale e utilizzando un approccio ispirato alle catene di Markov per memorizzare le sequenze degli eventi. Le Hierarchical Temporal Memories sono state sviluppate da Numenta inc.[1]

Una Hierarchical Temporal Memory è strutturata come una rete gerarchica di nodi dove i dati sensoriali entrano al livello più basso mentre le uscite della rete sono fornite dai nodi al livello più alto. Tali uscite rappresentano le possibili cause derivate dagli ingressi.

Ogni nodo all'interno di una rete Hierarchical Temporal Memory rileva pattern spazio-temporali ricorrenti (denominati *invarianti*) a partire dai suoi ingressi e li rag-

[1] http://www.numenta.com

gruppa come cause (o *gruppi*). Una rete HTM è addestrata in maniera non supervisionata, almeno secondo la concezione classica, ma il tempo effettua una supervisione implicita: se due eventi (ingressi) occorrono spesso consecutivamente, allora è molto probabile che entrambi appartengano alla stessa causa e quindi il nodo debba fornire la stessa uscita. Come in altri approcci connessionisti, i nodi in fondo alla gerarchia ricevono gli ingressi dai sensori e le loro uscite diventano gli ingressi dei nodi posti sopra di essi. Perciò i nodi ai livelli più bassi possono ricevere in ingresso solamente un insieme limitato dei dati sensoriali mentre i dati visibili ainodi ai livelli più alti hanno un' estensione maggiore rispetto ai dati in ingresso, poiché sono già stati processati dagli altri nodi. Questo porta a considerare come i nodi di basso livello possano trovare delle cause appartenenti ad una scala spaziale (ingressi al sistema) limitata mentre i nodi ad un livello più alto sono in grado di trovare cause ad una scala più ampia. In altre parole, ogni nodo ha una rappresentazione interna delle possibili cause (eventi spazio-temporali) e le sue uscite sono un insieme i cui elementi rappresentano le probabilità che le cause corrispondenti siano attive. Per esempio se la memoria di un nodo è composta da 5 cause (gruppi) e l'uscita di un nodo è [0.6, 0.2, 0.1, 0.9, 0.15] significa che ad un certo tempo la quarta causa è l'evento che più probabilmente sta avvenendo, ma anche le altre hanno una certa probabilità. Ogni componente del vettore di uscita del nodo rappresenta la probabilità di attivazione di tale gruppo indipendentemente dalle altre componenti. Tale codifica permette di rappresentare l'incertezza di un nodo, semplificando gli ingressi senza cancellare informazioni necessarie ai nodi superiori. Un esempio di tale gerarchia e di tali segnali può essere osservato in figura 1.1. In accordo a tale modello, l'uscita del nodo a livello più alto della rete può essere considerato la distribuzione delle probabilità che l'ingresso corrente corrisponda alle cause (*gruppi*) memorizzate in tale nodo.

In figura 1.1 è rappresentata una rete HTM generica dove i nodi sensore sono al livello più basso e i segnali salgono da essi fino al nodo al livello più alto attraverso la rete.

Per misurare le prestazioni di una rete è necessario aggiungere un classificatore supervisionato (come una Support Vector Machine o un K-Nearest Neighbors classifier) che analizzi le uscite del nodo al livello più alto. Quando utilizzate in questa

configurazione è possibile considerare la HTM come un filtro spazio-temporale applicato ai dati. Queste caratteristiche rendono le HTM una tecnologia molto potente ma le cui prestazioni dipendono dalla qualità dei dati utilizzate per addestrarle, come d'altro canto, succede a praticamente tutte le tecniche di *machine learning*.

La propagazione dei segnali all'interno delle Hierarchical Temporal Memories avviene in maniera similare alla *Bayesian Belief Propagation* ma le HTM aggiungono un *clustering* spazio-temporale e una struttura gerarchica, simile ai modelli connessionisti.

L'addestramento di una rete HTM procede sequenzialmente livello per livello, dai livelli più bassi al più alto. Una volta che i nodi al livello più basso sono stati addestrati, essi passeranno in modo inferenza e i nodi al secondo livello entreranno in modo addestramento utilizzando i risultati dell'inferenza dei nodi sottostanti. Queste operazioni sono ripetute finché tutti i livelli della rete non sono addestrati.

Si supponga che i nodi al primo livello siano stati addestrati. L'uscita di questi nodi corrisponde alle probabilità di attivazione dei pattern trovati nei dati di addestramento, diventando quindi l'ingresso per i nodi presenti al secondo livello. Questi nodi ora sono in fase di addestramento ed analizzeranno dati in ingresso per estrarre dei nuovi gruppi, o sequenze temporali. Questi gruppi sono una composizione dei gruppi temporali del livello sottostante e formano perciò una sequenza di sequenze, che rappresenta un evento ad una scala temporale più ampia. In questa maniera, salendo lungo la gerarchia della rete HTM, gli ingressi sono analizzati ad una scala temporale sempre maggiore, e l'uscita del nodo in cima alla rete riguarda tutti gli ingressi del sistema.

Un nodo HTM generico comprende due moduli: SpatialPooler e TemporalPooler. Ogni modulo deve essere addestrato prima di essere in grado di fornire un output. In taluni casi specifici è possibile utilizzare solamente una parte del nodo. Lo SpatialPooler quantizza un input multidimensionale per ridurre rumore e dimensionalità dei dati. Lo SpatialPooler estrae un insieme limitato di centri di quantizzazione (detti *coincidenze*) dai dati durante la fase di addestramento. Terminata tale fase il nostro può entrare nella fase di inferenza e descrive ogni pattern in ingresso come una distribuzione di probabilità associata ad ogni centro di quantizzazione. Il TemporaPooler

riceve in ingresso le uscite di uno (o più) SpatialPooler e raggruppa insieme le coincidenze che con maggiore frequenza si susseguono durante la fase di addestramento. Questo nodo crea una *Temporal Adjecency Matrix* (detta TAM) che contiene le transizioni tra le coincidenze ricevute in ingresso. Da tale matrice posso essere definiti gli stati: uno stato è definito dalla coincidenza corrente e dalla transizione rispetto a quella precedente. Il numero degli stati non è noto a priori ma dipende dal dataset utilizzato per la fase di addestramento, dai parametri dell'algoritmo e dai dati in ingresso al nodo. Tale valore è quindi dipendente dall'uscita dei nodi sottostanti; in particolare se le coincidenze in ingresso sono condivise da più sequenze temporali sarà necessario attuare un processo di replica degli stati (*splitting*) come verrà descritto nella sezione 4.2.3.

Al termine dell'osservazione dei dati di addestramento la TAM contiene tutte le informazioni di transizione tra gli stati e viene eseguito un algoritmo di raggruppamento per individuare i *gruppi*. I gruppi sono formati da catene di stati consecutivi; tali gruppi rappresentano perciò le "cause" nei dati di ingresso. Al termine di tale fase il numero di stati e di gruppi è fissato. Durante la fase di inferenza il TemporalPooler mappa ogni coincidenza in ingresso al corrispondente stato (utilizzando il proprio stato precedente) ed assegna una probabilità di attivazione ad ogni possibile gruppo memorizzato partendo dalla storia passata e dallo stato corrente.

Maggiori riferimenti a questa implementazione possono essere trovati nel lavoro di George and Hawkings [4], inoltre nella sezione 4.2.3 saranno presentate in maggiore dettaglio alcune parti rilevanti tra cui l'algoritmo di creazione e raggruppamento degli stati. Le Hierarchical Temporal Memories sono state utilizzate con successo in vari problemi di classificazione di immagini [5, 6], riconoscimento di gesti [7] e classificazione di attività umane[8].

È possibile notare come la corrente implementazione delle Hierarchical Temporal Memories non copra tutti gli aspetti descritti dal Memory Prediction Framework. In particolare non sono stati implementati quei meccanismi di *feedback* all'interno della rete, di temporizzazione, di auto-apprendimento.

1.3 Confronto con altri modelli

Il No Free Lunch Theorem asserisce che nessun algoritmo di addestramento ha un vantaggio fondamentale rispetto a qualsiasi altro algoritmo se consideriamo le sue prestazioni su tutti i possibili problemi [9].

Quello che importa è l'insieme delle assunzioni su cui un algoritmo si basa per la creazione del suo modello. L'assunzione compiuta dalle Hierarchical Temporal Memories è che i dati del mondo possano avere una rappresentazione gerarchica sia nello spazio che nel tempo, perciò esse ricreano nella loro struttura di memoria tale rappresentazione.

Bayesian Network

Le reti Bayesiane [10, 11] sono una classe di modelli probabilistici che sono utilizzati per analizzare relazioni statistiche tra un insieme di variabili. Tale modello non specifica che cosa debbano rappresentare tali variabili, perciò viene definito come un modello probabilistico general purpose. Le reti Bayesiane sono strutturate come un grafo aciclico direzionato su cui si fanno alcune assunzioni di indipendeza delle distribuzioni di probabilità. Altri modelli, come gli Hierarchical Hidden Markov Models possono essere riformulati come reti Bayesiane.

Hierarchical Hidden Markov Models

Gli Hierarchical Hidden Markov Models o HHMM sono abbastanza simili alle Hierarchical Temporal Memories poichè modellano il tempo in una struttura gerarchica; tuttavia mancano della componente spaziale.

Support Vector Machine

Le Support Vector Machine [12] sono un modello discriminativo altamente efficiente nel trovare i confini di separazione tra differenti esempi di categorie in uno spazio di elevate dimensioni. Non vi sono assunzioni di gerarchia o organizzazione temporale. Non essendo un modello generativo non possono essere utilizzate per generare delle

predizioni in avanti nel tempo. Le Support Vector Machine possono essere utilizzate in congiunzione alle HTM per ottenere un classificatore supervisionato di efficienza maggiore [13].

Multi Layer Perceptron

Le reti neurali [14], nella loro più classica forma, sono modelli di apprendimento supervisionato addestrati tramite un algoritmo denominato backpropagation. Nella loro forma più classica non sono un modello generativo e anche se alcune implementazioni sfruttano l'informazione temporale essa non assume un ruolo centrale come nelle HTM. La struttura a livelli di una rete neurale differisce fondamentalmente dalla struttura gerarchica di una HTM, infine i nodi di una rete neurale possono essere assimilati a soglie di attivazione e non contengono vere e proprie memorie di sequenze come i nodi HTM.

HMAX

Il modello HMAX [15] è un modello biologicamente realistico che ha molte similarità con le HTM. Tale modello utilizza una struttura gerarchica simile e le *feature* che il modello HMAX utilizza nei vari livelli sono simili alle coincidenze e ai gruppi temporali delle HTM. Le differenze principali sono nel meccanismo di apprendimento che, nel modello HMAX, richiede un maggior intervento manuale, nella fase di inferenza che non è probabilistica e dalla caratteristica non generativa del modello, ovvero non in grado di generare delle predizioni a partire da uno stato del sistema.

1.4 Analisi dell'implementazione di riferimento

È possibile descrivere matematicamente il processo utilizzato dalle Hierarchical Temporal Memories utilizzando un modello generativo (mostrato in Fig.1.2) e descrivendo le operazioni di comunicazione tra i nodi secondo la *Belief Propagation*.

Possiamo definire come "nodo genitore" e "nodo figlio" come due nodi posti su due livelli differenti, il primo posto al livello direttamente superiore rispetto a quello

del secondo; inoltre l'uscita di feedforward del "nodo figlio" corrisponde a tutto, o a parte, dell'ingresso del "nodo genitore".

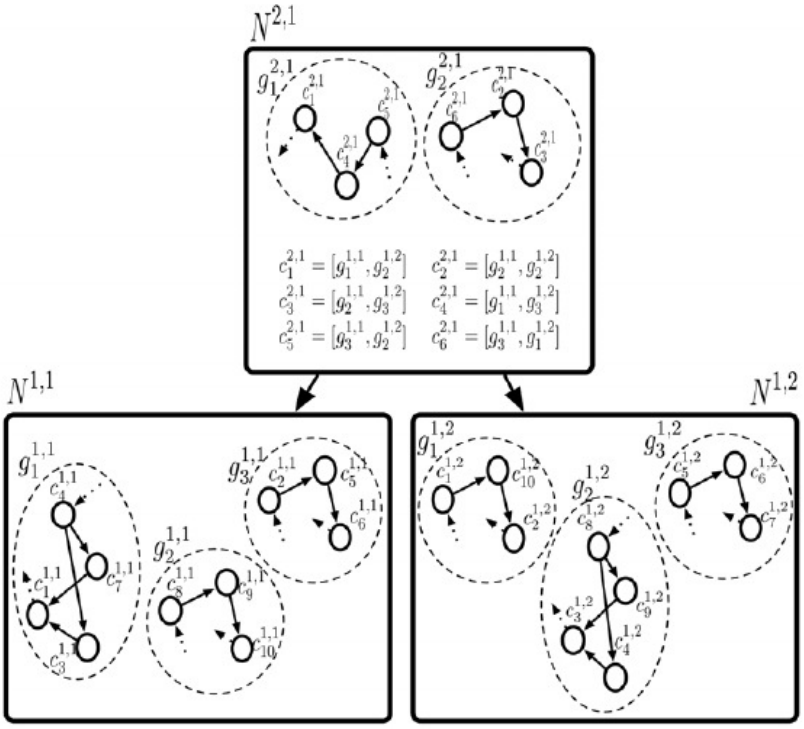


Figura 1.2: La figura mostra un modello generativo per una HTM a 2 livelli con 3 nodi. Ogni nodo nella gerarchia contiene un insieme di modelli di coincidenza etichettati con $c's$ ed un insieme di catene di Markov etichettate con $g's$ definite sull'insieme dei modelli di coincidenza. Un modello di coincidenza in un nodo rappresenta una co-attivazione di una particolare catena di Markov dei suoi nodi figli. Il modello generativo HTM è una gerarchia spazio-temporale in cui i livelli più alti rimangono stabili per più tempo e possono generare attivazioni più veloci a basso livello.

Ogni nodo nella gerarchia contiene un insieme di *coincidence patterns* (modelli

di coincidenza) $c_1, c_2, \cdots, c_{|C|}$, ed un insieme di catene di Markov $g_1, g_2, \cdots, g_{|G|}$, in cui ogni catena di Markov è definita su un sottoinsieme dell'insieme dei modelli di coincidenza in quel nodo. Un modello di coincidenza in un nodo rappresenta una co-attivazione di catene di Markov dei suoi nodi figli. Un modello di coincidenza che è generato campionando una catena di Markov in un nodo di livello più alto, attiva le relative catene di Markov nei nodi di livello inferiore. Per un particolare modello di coincidenza e per una particolare catena di Markov che è "attiva" al nodo di livello superiore, vengono generate delle sequenze di modelli di coincidenza campionando le catene di Markov attivate nei nodi figli.

Il processo di apprendimento di un modello HTM per dati spazio-temporali coincide con il processo di apprendimento dei modelli di coincidenza e delle catene di Markov in ogni nodo ad ogni livello della gerarchia. Sebbene algoritmi di vari livelli di complessità possano essere usati per apprendere gli stati di un nodo HTM, il processo base può essere schematizzato in due operazioni:

1. memorizzazione dei modelli di coincidenza;
2. apprendimento di una serie di catene di Markov sullo spazio dei modelli di coincidenza.

Nel caso di un modello generativo semplificato, un nodo HTM ricorda tutti i modelli di coincidenza che vengono generati dal modello generativo. Nei casi del mondo reale, dove non è possibile conteggiare tutte le coincidenze incontrate durante l'apprendimento, si è scoperto [4] che è sufficiente memorizzare un numero fisso di una selezione casuale dei modelli di coincidenza per poter descrivere adeguatamente tutto il set di dati. Ogni catena di Markov in un nodo rappresenta un insieme di modelli di coincidenze che si verificano in sequenza nel tempo. Questo vincolo di prossimità temporale è analogo al principio di lentezza temporale utilizzato nell'apprendimento di features invarianti. L'apprendimento di un insieme di catene di Markov è semplificato notevolmente per mezzo del vincolo della lentezza. Un modo semplice per imparare l'insieme di catene di Markov per i casi del mondo reale è quello di imparare una grande matrice di transizione che viene poi partizionata con un algoritmo di partizionamento grafico.

Come già introdotto nella sezione 1.4.2, subito dopo la fase di apprendimento, segue quella di inferenza. Un nodo che ha terminato il suo processo di apprendimento possiede un insieme di modelli di coincidenza ed un insieme di catene di Markov. La Fig.1.3 [A] mostra un nodo che ha 5 modelli di coincidenza e 2 catene di Markov.

Il meccanismo di inferenza in una HTM è basato sulla propagazione di nuove *evidenze* da qualsiasi punto della rete verso l'alto e verso il basso. La presentazione di una nuova immagine al primo livello di una rete di visione HTM è un esempio di una nuova evidenza. Una nuova immagine può portare ad un differente *belief* nel top level della rete. In generale, le HTM inferiscono su input che variano nel tempo. L'inferenza su un input statico rappresenta un caso particolare di questo calcolo.

Le HTM usano la belief propagation bayesiana per l'inferenza e dato che un nodo HTM fa un'astrazione sia dello spazio che del tempo, è necessario derivare nuove equazioni per la belief propagation nei nodi HTM, come mostrato in [4].

1.4.1 Belief propagation nei nodi HTM

In generale i messaggi che giungono in un nodo HTM dai suoi figli rappresentano il grado di certezza sulle catene di Markov del figlio. Il nodo converte questi messaggi al suo grado di certezza su ogni sua catena di Markov basandosi sulla storia dei messaggi ricevuti. Questo poi viene passato al nodo al livello superiore. Ciò che il nodo riceve dai suoi genitori tramite il link di feedback è il grado di certezza dei genitori su queste catene di Markov. Nella didascalia in Fig.1.3 viene descritto il significato dei termini "feed-forward" e "feedback". Le catene di Markov sono poi *risolte* passo-passo per trovare la distribuzione di probabilità sui modelli di coincidenza. Da ciò vengono calcolati i gradi di certezza del nodo sulle catene di Markov dei suoi nodi figli. Questi messaggi di feedback sono poi inviati ai nodi figli.

In accordo con quanto formulato da D.George in [4] si riportano in tabella 1.1 le equazioni che regolano il funzionamento del nodo.

Verrà ora descritta in dettaglio la notazione ed il significato di queste equazioni usando il nodo HTM standard mostrato in Fig.1.3.

In queste equazioni i modelli di coincidenza sono indicati con $c_i's$ e le catene di Markov con $g_i's$. Il nodo HTM mostrato in Fig.1.3[A] contiene 5 modelli di coin-

cidenza e 2 catene di Markov (gruppi). La matrice di transizione della probabilità della catena di Markov g_r è indicata con $P((c_i(t)|c_j(t-1), g_r)$. Questo termine compare nelle equazioni 1.3 e 1.6. Ogni modello di coincidenza nel nodo rappresenta una co-occorrenza dei gruppi temporali dei suoi figli. Specificazioni del modello di coincidenza sono usate nei calcoli descritti nelle equazioni 1.1 e 1.8.

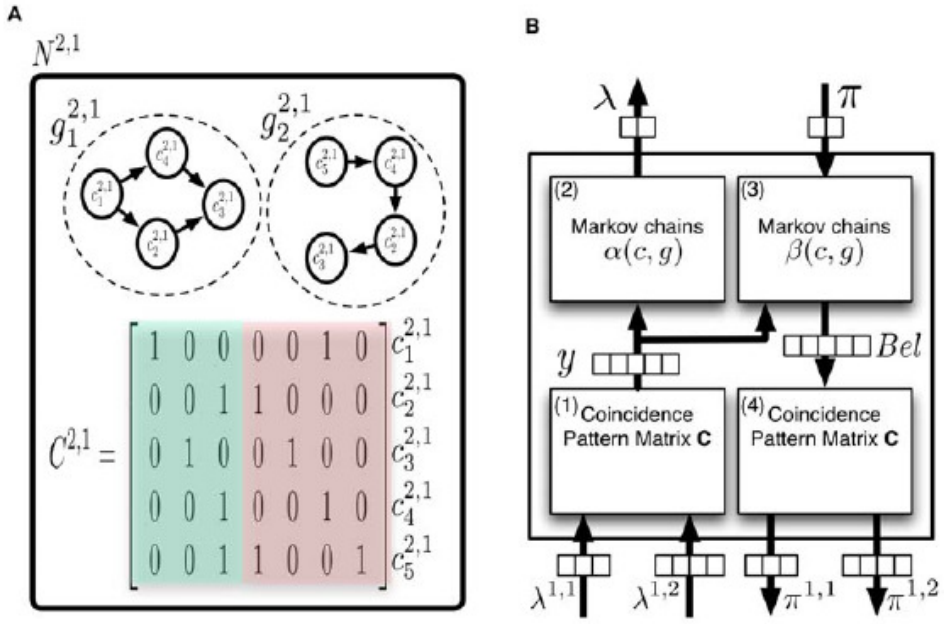


Figura 1.3: [A] Struttura del nodo HTM standard con cinque modelli di coincidenza e due catene di Markov. Si assume che questo sia il primo nodo al livello 2 di una rete ed è etichettato come $N^{2,1}$. Ogni modello di coincidenza rappresenta una co-occorenza delle catene di Markov del figlio. Questo nodo ha due figli. Il figlio 1 ha tre catene di Markov e il figlio 2 ha quattro catene di Markov, quindi ci sono sette elementi in ogni modello di coincidenza. Le porzioni del modello di coincidenza che provengono dal primo e secondo figlio sono mostrate in differenti scale di colore. I rettangoli dentro il nodo sono unità che elaborano le equazioni nelle righe che corrispondono al numero mostrato in ogni rettangolo. Viene utilizzato il termine "feed-forward" o "bottom-up" per indicare messaggi ricevuti dai figli e messaggi inviati ai genitori mentre viene utilizzato il termine "feedback" o "top-down" per indicare messaggi ricevuti dai genitori e messaggi inviati ai figli. Il nodo in figura ha due messaggi "bottom-up" in ingresso che provengono dai due figli e ha due messaggi "top-down" in uscita che sono i messaggi inviati a questi figli. Le frecce rappresentano vettori di ingresso, uscita e valori intermedi di calcolo. Il numero intermedio di componenti di ogni vettore è rappresentato usando un array posto su queste frecce.

1)Calcolo della probabilità sui modelli di coincidenza.	$$y_i = P(-e_t \mid c_i(t)) \propto \prod_{j=1}^{M} \lambda_t^{m_j}(r_i^{m_j}) \quad (1.1)$$ dove il modello di coincidenza c_i è la co-occorrenza della $r_i^{m_1}$-iesima catena di Markov dal figlio 1, $r_i^{m_2}$-iesima catena di Markov dal figlio 2, $\cdots$, e $r_i^{m_M}$-iesima catena di Markov dal figlio M.
2)Calcolo della probabilità feed-forward sulle catene di Markov usando la programmazione dinamica.	$$\lambda_t(g_r) = P(-e_0^t \mid g_r(t)) \propto \sum_{c_i(t) \in C^k} \alpha_t(c_i, g_r) \quad (1.2)$$ $$\alpha_t(c_i, g_r) = P(-e_t \mid c_i(t)) \sum_{c_j(t-1) \in C^k} P(c_i(t) \mid c_j(t-1), g_r)\alpha_{t-1}(c_j, g_r) \quad (1.3)$$ $$\alpha_0(c_j, g_r) = P(-e_0 \mid c_i(t=0))P(c_i(t=0) \mid g_r) \quad (1.4)$$
3)Calcolo della distribuzione belief sui modelli di coincidenza.	$$Bel_t(c_i) \propto \sum_{g_r \in G^k} \beta_t(c_i, g_r) \quad (1.5)$$ $$\beta_t(c_i, g_r) = P(-e_t \mid c_i(t)) \sum_{c_j(t-1) \in C^k} P(c_i(t) \mid c_j(t-1), g_r)\beta_{t-1}(c_j, g_r) \quad (1.6)$$ $$\beta_0(c_i, g_r) = P(-e_0 \mid c_i(t=0))P(c_i \mid g_r, +e_0)\pi_0(g_r) \quad (1.7)$$
4)Calcolo dei messaggi da inviare ai nodi figli.	$$\pi^{m_i}(g_r) \propto \sum_i I(c_i)Bel(c_i) \quad (1.8)$$ where $$I(c_i) = \begin{cases} 1 & \textit{se } g_r^{m_i} \textit{ e' una componente di } c_i \\ 0 & \textit{altrimenti} \end{cases} \quad (1.9)$$

Tabella 1.1: Equazioni di belief propagation per un nodo HTM

Ogni nodo riceve messaggi in ingresso feed-forward dai suoi figli e manda messaggi feed-forward ai suoi genitori. I messaggi in input feed-forward sono denotati con $\lambda^{childnodeindex}$. Il messaggio di output feed-forward del nodo è indicato con λ. Allo stesso modo il nodo riceve messaggi di feedback dai suoi genitori e manda messaggi di feedback ai suoi nodi figli. Il messaggio di feedback in ingresso al nodo è denotato con π. I messaggi di uscita che il nodo manda ai nodi figli sono indicati con $\pi^{childnodeindex}$. Le equazioni mostrate in tabella 1.1 descrivono come i messaggi in uscita siano derivati da quelli in ingresso. Dal punto di vista del nodo, i messaggi feed-forward trasportano informazione riguardo alll'*evidenza*, proveniente dal basso, che ad ogni tempo t è denotata con $-e_t$. Allo stesso modo l'evidenza proveniente dal genitore è denotata con $+e_t$.

L'equazione 1.1 descrive come il nodo calcola la sua probabilità dei modelli di coincidenza, usando i messaggi dei figli. La probabilità bottom-up del modello di coincidenza c_i al tempo t è rappresentata da $y_i(t)=P(-e_t|c_i(t))$. La probabilità di ogni modello di coincidenza è calcolata come il prodotto delle componenti del messaggio che corrispondono a quel modello di coincidenza.

Nell'equazione 1.2, la probabilità bottom-up della catena di Markov g_r al tempo t è indicata con $P(-e_0^t|gr(t))$, dove il termine $-e_0^t$ rappresenta la sequenza di evidenze bottom-up dal tempo 0 al tempo t. Ciò mostra che la probabilità delle catene di Markov dipende dalla sequenza degli ingressi ricevuti dal nodo. Le variabili α e β definite nelle equazioni 1.3 e 1.6 sono variabili di stato che vengono aggiornate in modo ricorsivo ad ogni istante di tempo. Nelle equazioni 1.3 e 1.6 gli stati sono aggiornati ad ogni istante passando lo stato del precedente "time-step" attraverso le matrici di transizione di Markov e combinandole con l'evidenza bottom-up/top-down.

L'implementazione delle equazioni di belief propagation per le HTM è caratterizzata dai seguenti passi:

1. Calcolo della probabilità sui modelli di coincidenza: gli ingressi bottom-up al nodo HTM rappresentano i messaggi di uscita feed-forward dai suoi figli. Questi messaggi di uscita trasportano informazione sul grado di certezza delle catene di Markov nei nodi figli. Ogni messaggio è un vettore di lunghezza uguale al numero di catene di Markov nel figlio corrispondente. La probabilità di

coincidenze viene derivata da questi messaggi in ingresso secondo l'equazione 1.1. Questa operazione viene eseguita dal rettangolo marcato (1) in Fig.1.3[B]. La Fig.1.4 mostra un'implementazione neuronale astratta per il nodo standard HTM. In tale figura ogni nodo corrisponde ad un modello di coincidenza memorizzato. Il modello corrispondente alla co-occorrenza è memorizzato nelle connessioni che questi neuroni fanno con i messaggi dai nodi in ingresso del figlio. Per esempio, il neurone corrispondente al modello di coincidenza c_1 ha connessioni con la prima posizione del messaggio dal primo figlio e la terza posizione del messaggio dal secondo figlio. Queste connessioni corrispondono alla prima riga della matrice $C^{2,1}$ del modello di coincidenza in Fig.1.3[A]. L'uscita denotata con y in Fig.1.3[B], è un vettore di 5 componenti, una per ogni modello di coincidenza. Questo vettore rappresenta la probabilità dei modelli di coincidenza basata sui messaggi ricevuti dai nodi figli.

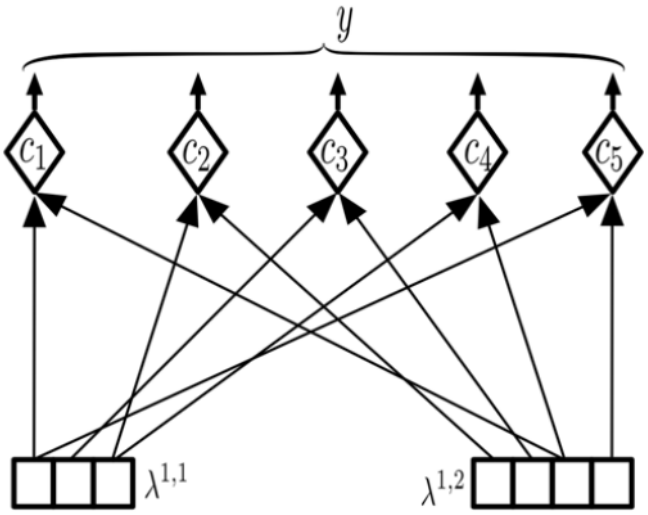


Figura 1.4: Circuito per il calcolo della probabilità bottom-up sui modelli di coincidenza.

2. Calcolo della probabilità feed-forward sulle catene di Markov usando la pro-

grammazione dinamica: il prossimo passo nel calcolo dei messaggi feed-forward, corrispondente al rettangolo marcato (2) in Fig.1.3[B], è il calcolo del grado di certezza del nodo HTM in ciascuna delle sue catene di Markov. La quantità che deve essere calcolata è $P(-e_0, -e_1, \cdots, -e_t|g_i)$ per ogni catena di Markov g_i, in cui $-e_0, -e_1, \cdots, -e_t$ rappresentano la distribuzione delle evidenze bottom-up ricevute dal tempo 0 al tempo t. La probabilità delle catene di Markov dipende dalla sequenza di messaggi che il nodo ha ricevuto dai suoi figli. Un calcolo esatto di questa quantità non è fattibile poichè ciò richiede l'enumerazione delle probabilità di un numero esponenzialmente crescente di percorsi. Per calcolare $P(-e_0^t|g_i)$ in modo efficiente, tutte le evidenze passate devono essere inserite in una variabile di stato che può essere aggiornata in modo ricorsivo ad ogni istante. Ciò è fatto utilizzando la *programmazione dinamica* (equazione 1.3).

L'equazione 1.3 può avere un'implementazione neuronale molto efficiente come mostrato in Fig.1.5.

I neuroni cerchiati (neuroni "cerchio") in questo circuito implementano la sequenza di memoria delle catene di Markov nel nodo HTM. Le connessioni tra i neuroni cerchio implementano le probabilità di transizione della catena di Markov.

Tutti i neuroni cerchio collocala in una colonna hanno lo stesso ingresso bottom-up: il modello di coincidenze rappresentato con i rombi. Ogni colonna, considerando solo l'ingresso bottom-up, può essere pensata come rappresentativa di un modello di coincidenza. Questi neuroni "cerchi" hanno anche connessioni "laterali" che provengono da altri neuroni"cerchio" nella stessa catena di Markov. Le connessioni laterali specificano il significato di un neurone in una sequenza. Un neurone "cerchio" che è etichettato con $c_i g_j$ rappresenta il modello di coincidenza c_i nel contesto della catena di Markov g_j. Lo stesso modello di coincidenza può appartenere a differenti catene di Markov e può quindi essere attivo in differenti contesti temporali. Ogni neurone "cerchio" in questo circuito compie lo stesso calcolo. Ogni neurone calcola la sua uscita

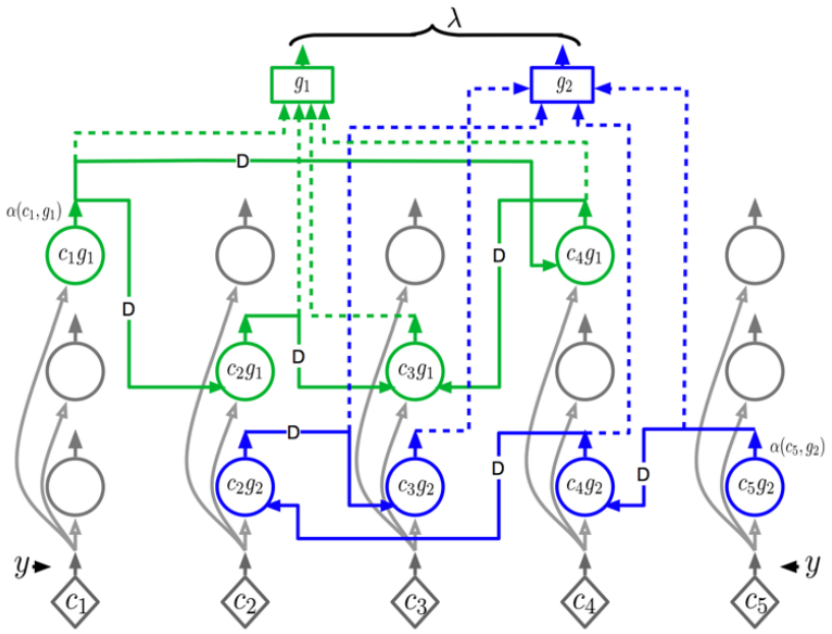


Figura 1.5: Circuito per il calcolo della probabilità delle catene di Markov basato su una sequenza di ingressi.

moltiplicando l'ingresso bottom-up con la somma pesata dei suoi ingressi laterali. L'uscita di un neurone cerchio è denotata con α *(numero di coincidenza, numero della catena di Markov)*. Perciò l'uscita di ogni neurone "cerchio" $c_i g_r$ è calcolata come:

$$\alpha_t(c_i, g_r) = y(i) \sum_j w(i,j) * \alpha_{t-1}(c_j, g_r) \tag{1.10}$$

Quindi l'uscita di un neurone cerchio ad ogni istante di tempo è la somma pesata delle uscite dei neuroni nella stessa catena di Markov all'istante di tempo precedente, moltiplicata per l'attivazione bottom-up attuale. L'equazione 1.10

corrisponde all'equazione 1.3 se si sostituisce $w(i, j)$ con $P(c_i c_j, g_r)$. Quindi i circuiti mostrati in Fig.1.5 implementano l'equazione 1.3.

Si consideri adesso il terzo tipo di neuroni, i neuroni rettangolari (neuroni "rettangolo"), in Fig.1.5. Il neurone rettangolo marcato g_1 riceve gli ingressi dalle uscite di tutti i neuroni cerchio nella catena di Markov g_1. Ad ogni istante di tempo, l'uscita del neurone rettangolo è calcolata come somma (o massimo) degli ingressi a quel neurone. L'operazione dei neuroni rettangolo corrisponde all'integrazione delle attivazioni di tutti i neuroni cerchio della stessa catena di Markov. Le uscite concatenate dei neuroni rettangolo sono il messaggio λ che questo nodo invia ai suoi genitori. Questo è un vettore di due componenti, corrispondente alle due catene di Markov nel nodo standard in Fig.1.3.

3. Calcolo della distribuzione di belief sui modelli di coincidenza: un nodo HTM calcola il suo grado di belief in un modello di coincidenza combinando le evidenze bottom-up, top-down e temporali secondo le equazioni presenti nel terzo blocco della tabella 1.1. L'ingresso top-down del nodo è un vettore di lunghezza uguale al numero di catene di Markov del nodo. L'uscita di questo calcolo è il vettore belief sui modelli di coincidenza.

   Il calcolo del belief, descritto nell'equazione 1.5, ha quasi la stessa forma di quello presente nell'equazione 1.3. La variabile di stato β ha la stessa forma della variabile di stato α . L'unica differenza tra queste due equazioni è la moltiplicazione per un fattore $P(g_k|+e_0)$ nelle equazioni del calcolo del belief. Per questo motivo si può concludere dicendo che l'implementazione neuronale della parte di programmazione dinamica dell'equazione del calcolo del belief è molto simile a quella della programmazione dinamica in avanti della variabile α. Questa implementazione è mostrata in Fig.1.6 in cui i neuroni con i cerchi pieni corrispondono ai neuroni cerchio nel calcolo in avanti. I neuroni con il cerchio pieno adesso hanno anche un ingresso moltiplicativo top-down che corrisponde a $P(g_k|+e_0)$. I neuroni pentagonali (neuroni "pentagono") sono i neuroni belief e le operazioni che essi svolgono sono differenti da quelle dei neuroni rettangolo in Fig.1.5. I neuroni rettangolo mettono insieme diffe-

renti modelli di coincidenza nella stessa catena di Markov; i neuroni pentagono mettono insieme lo stesso modello di coincidenza in differenti catene di Markov.

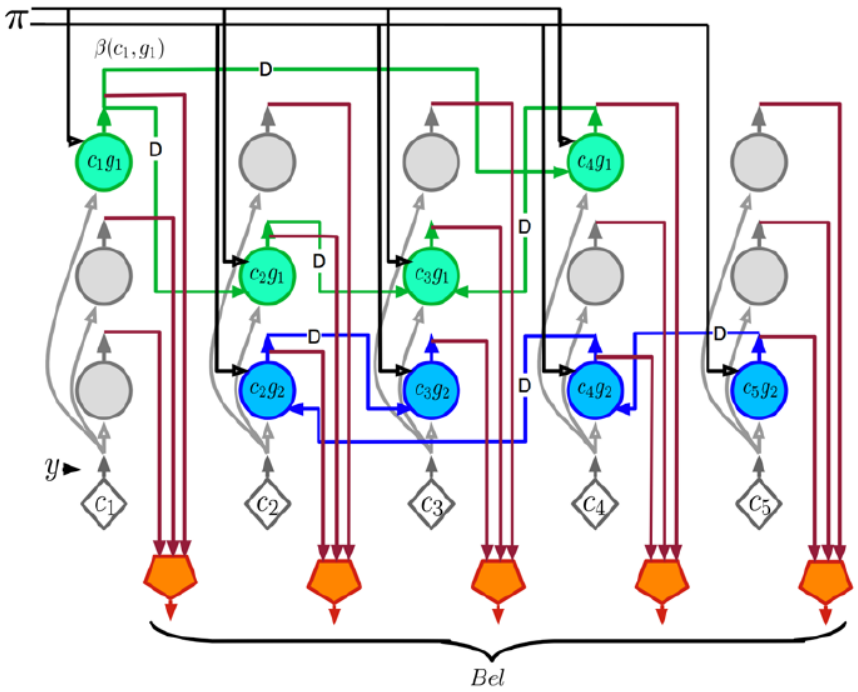


Figura 1.6: Circuito per il calcolo della distribuzione del belief sui modelli di coincidenza integrando la sequenza degli ingressi bottom-up con gli ingressi top-down.

4. Calcolo dei messaggi da inviare ai nodi figli: quest'ultimo step è descritto dall'equazione 1.8. L'ingresso per questa operazione è il vettore belief. Le uscite sono i messaggi π che sono inviati ai nodi figli. Un messaggio è mandato ad ogni nodo figlio e descrive il grado di certezza che questo nodo ha riguardo alle catene di Markov dei nodi figli. La Fig.1.7 mostra come questa equazione può essere implementata usando i neuroni. Il belief in ingresso è alimentato da

neuroni "esagonali" che calcolano i messaggi per i nodi figli. La Fig.1.7 mostra due insiemi di neuroni esagonali (neuroni "esagono") che corrispondono ai due nodi figli di questo nodo. Ogni neurone esagonale corrisponde ad una catena di Markov del suo nodo figlio. Il nodo figlio sulla sinistra ha 3 catene di Markov e quello di destra ne ha 4. Le uscite di questi neuroni esagono sono i messaggi che sono inviati ai rispettivi figli.

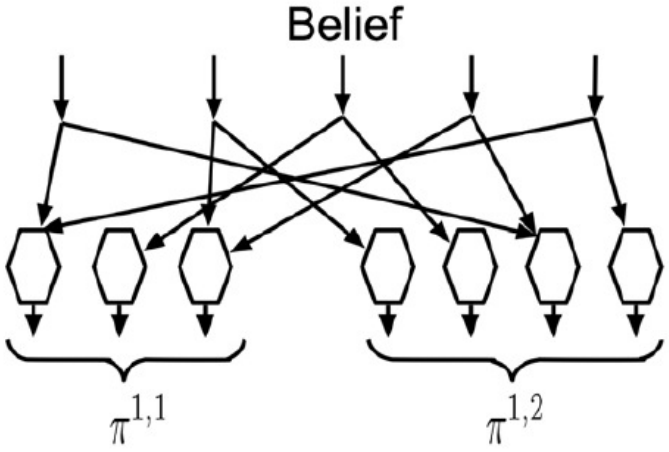


Figura 1.7: Circuito per il calcolo dei messaggi che devono essere inviati ai figli in accordo con l'equazione 1.8.

Le connessioni tra l'ingresso ed i neuroni esagono codificano i componenti dei modelli di coincidenza. L'operazione dei neuroni esagono mostrata in Fig.1.7 può essere considerata come l'inverso delle operazioni compiute dai neuroni diamante descritti in Fig.1.4. I pesi sugli ingressi di entrambi questi tipi di neuroni definiscono i modelli di coincidenza. Nel caso dei neuroni "diamante", essi calcolano la probabilità delle coincidenze partendo dalla distribuzione di probabilità sulle catene di Markov per ogni figlio. I neuroni esagono fanno il contrario: calcolano le distribuzioni di probabilità sulle catene di Markov per ogni figlio partendo dalla distribuzione di probabilità sui modelli di coincidenza.

1.4.2 Analisi del nodo HTM

Come descritto nella sezione 1.2, un nodo, indipendentemente dalla sua collocazione nella rete, è composto da due parti distinte, lo spatial pooler ed il temporal pooler, ognuna delle quali esegue un compito ben preciso, sia in fase di training che in fase di inferenza. Analizziamo in dettaglio il funzionamento di questi due moduli.

Spatial Pooler

Lo spatial pooler possiede due fasi di funzionamento:

- durante la fase di apprendimento quantizza i modelli in ingresso e memorizza i centri di quantizzazione;
- durante la fase di inferenza produce delle uscite in relazione ai centri di quantizzazione memorizzati.

Per spiegare come avviene l'apprendimento dei punti di quantizzazione si consideri il nodo (a) in Fig.1.8 al livello-1 della gerarchia che rappresenta un esempio di visione artificiale.

L'ingresso di questo nodo è una parte dell'immagine in ingresso, di dimensione 4x4 pixel. Il nodo considera questo ingresso come un modello di lunghezza 16 linearizzando la matrice di pixel. Così come la rete è esposta ai fotogrammi da un movimento, così il nodo è esposto ad una serie di modelli. Questi modelli rappresentano gli ingressi per lo Spatial Pooler ed esso apprende una quantizzazione di questi modelli in ingresso. Per ogni modello in ingresso, si controlla se c'è un centro di quantizzazione che si trova entro la distanza euclidea D (un parametro del nodo) da esso. Se esiste non si esegue nessuna azione, altrimenti viene aggiunto un nuovo modello alla lista dei centri di quantizzazione corrispondente all'ingresso stesso.

Con tale metodo lo spatial pooler memorizza un piccolo sottoinsieme dei suoi modelli in ingresso come centri di quantizzazione. Ogni centro di quantizzazione è un vettore di lunghezza uguale a quella dei modelli in ingresso.

Il parametro D nello spatial pooler dovrebbe essere "grande" abbastanza da renderlo immune dalle variazioni nei modelli in ingresso causate dal rumore. Se D è trop-

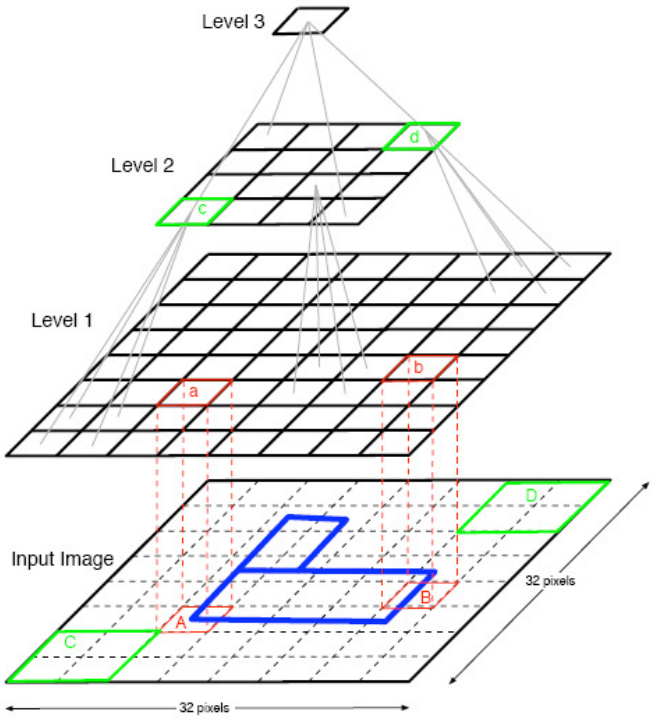


Figura 1.8: Struttura di una HTM per l'apprendimento di rappresentazioni invarianti per immagini binarie. La rete è organizzata in 3 livelli e il segnale in ingresso è gestito al livello base. I nodi sono raffigurati come quadrati. Il top level della rete ha un nodo, quello di mezzo ne ha 16 e quello base 64. L'immagine in input ha dimensione 32x32 pixel ed è divisa in blocchi di 4x4 pixel. Ogni ingresso del nodo del livello base corrisponde ad un blocco di 4x4 pixel. La figura mostra come i blocchi dell'immagine siano mappati ai nodi del livello base. Due nodi al livello 1 sono marcati (a) e (b) ed i quadrati marcati (A) e (B) nell'immagine in input rappresentano i rispettivi campi recettivi di questi nodi. Lo stesso per quanto riguarda quelli marcati (C) e (D).

po piccolo, lo spatial pooler genererà troppi centri di quantizzazione; diversamente, se troppo grande, potrebbe raggruppare modelli con significato diverso.

All'inizio dell'apprendimento, il nodo aggiunge rapidamente nuovi centri di quantizzazione. Tanto più vengono aggiunti centri di quantizzazione, tanto più diminuisce la quantità di spazio che non è mappata ad un centro di quantizzazione esistente. Quindi il numero di nuovi centri di quantizzazione che uno spatial pooler forma in un'unità di tempo decresce.

Una volta che il processo di quantizzazione è terminato, lo spatial pooler può eseguire inferenza e usare i dati. In questa fase esso calcola uno stato di uscita per ogni modello in ingresso.

Si indichi con N_c il numero totale di centri di quantizzazione che sono stati memorizzati all'interno di un nodo e siano $c_1, c_2, \cdots, c_{N_c}$ i singoli centri di quantizzazione. L'output dello spatial pooler è un vettore di lunghezza N_c. L'i-esima posizione in questo vettore corrisponde a c_i che è l'i-esimo centro di quantizzazione memorizzato all'interno dello spatial pooler. In generale quest'output rappresenta una probabilità di distribuzione nello spazio dei centri di quantizzazione. Questa distribuzione può essere considerata indicativa del grado di unione del modello in ingresso con i centri di quantizzazione.

Il primo passo per calcolare il grado di unione tra un ingresso e il centro di quantizzazione è calcolare la loro distanza euclidea. Sia d_i la distanza dell' i-esimo centro di quantizzazione dall'ingresso. Più larga è questa distanza e più piccola è l'unione tra l'ingresso ed il centro di quantizzazione. Se si assume che la probabilità che un modello appartienga ad un centro di quantizzazione sia descritta come una funzione Gaussiana della distanza euclidea, allora la probabilità che il modello in ingresso appartenga al centro di quantizzazione i può essere calcolata come $e^{-d_i^2/\sigma^2}$.

Le uscite dello spatial pooler sono passate poi al temporal pooler ed è in base a questi ingressi che il temporal pooler esegue l'apprendimento ed, eventualmente, l'inferenza.

Temporal Pooler

L'operazione compiuta dal temporal pooler può essere divisa in tre fasi. In una prima fase esso apprende i modelli di transizione temporale tra i centri di quantizzazione dello spatial pooler. Poi, basandosi su queste transizioni temporali, forma

gruppi di centri di quantizzazione in base alle vicinanze temporali. Dopo che sono state completate queste due fasi, il temporal pooler inizia a produrre le uscite in relazione ai gruppi temporali appresi.

Sia $y(t)$ l'ingresso al temporal pooler al tempo t. Come detto in precedenza, questo è un vettore di lunghezza N_c e può essere visto come una distribuzione di probabilità nello spazio dei centri di quantizzazione dello spatial pooler. Sia $c(t)$ l'indice del massimo di $y(t)$, ad esempio, $c(t) = argmax\ y(t)$. In altre parole, $c(t)$ è l'indice del centro di quantizzazione che è più attivo al tempo t, e $c(t-1)$ può essere considerato come l'indice del centro di quantizzazione che era attivo maggiormente al tempo $t-1$. Allo scopo di apprendere i modelli delle transizioni temporali dei centri di quantizzazione, il temporal pooler osserva nel tempo i vari $c(t)$. Esistono vari metodi per apprendere e rappresentare transizioni temporali. Sempre per semplicità, si consideri un modello che impari una matrice di adiacenza temporale del primo ordine tra i centri di quantizzazione.

Apprendere la matrice di adiacenza temporale, significa osservare occorrenze consecutive dei centri di quantizzazione. All'inizio di questo processo di apprendimento, il temporal pooler inizializza una matrice che ha N_c righe ed N_c colonne tutte a zero (N_c è il numero dei centri di quantizzazione che il nodo ha memorizzato). Le righe e le colonne della matrice corrispondono ai centri di quantizzazione che si verificano rispettivamente al tempo $t-1$ e al tempo t. La matrice di adiacenza temporale all'interno del temporal pooler sarà denotata con il simbolo T.

Siano $x(t-1)$ e $x(t)$ gli ingressi al nodo al tempo $t-1$ e al tempo t. Le uscite corrispondenti dello spatial pooler (e di conseguenza gli ingressi del temporal pooler) sono $y(t-1)$ e $y(t)$. Il temporal pooler calcola $c(t)$ e $c(t-1)$ come $c(t) = argmax\ y(t)$ e $c(t-1) = argmax\ y(t-1)$. Al tempo t, il nodo aggiorna la sua matrice di adiacenza temporale incrementando $T(c(t-1), c(t))$.

Durante questo processo, il temporal pooler periodicamente normalizza la sua matrice di adiacenza temporale lungo le righe per ottenere la matrice T_{norm} che è una matrice di transizione del primo ordine. Gli ingressi (i, j) di questa matrice rappresentano la probabilità di osservare c_j al tempo t, che era c_i al tempo $t-1$. Il nodo

può fermare questo processo di apprendimento quando la matrice normalizzata T_{norm} è sufficientemente stabilizzata. [2].

Al termine della creazione della matrice di adiacenza temporale, essa viene partizionata in gruppi. L'obiettivo è partizionare l'insieme dei centri di quantizzazione in sottogruppi *temporalmente coerenti*. Ciò significa che bisogna formare gruppi in modo che tutti i centri di quantizzazione all'interno di uno stesso gruppo si susseguano tra loro e tutti i centri di quantizzazione all'interno di gruppi differenti non si susseguano.

Ci sono differenti modi per formalizzare l'idea della formazione dei gruppi temporali partendo dalla matrice di adiacenza temporale. Un modo è quello di considerare questo come un problema di partizionamento di un grafo dove i centri di quantizzazione sono i vertici ed i valori di adiacenza temporale sono gli archi; in questo caso l'obiettivo è trovare sottografi fortemente connessi all'interno di un grafo. Un altro modo è considerarlo come un problema di clustering in cui la somiglianza tra due centri di quantizzazione viene misurata come adiacenza temporale tra questi due centri di quantizzazione. Differenti modi possono portare alla formazione di differenti gruppi.

La definizione di quello che può essere considerato un "buon" gruppo nell'algoritmo di creazione della matrice di partizione è: $D_1, \cdots, D_{N_g}$, dove D_i sono i sottogruppi disgiunti dei centri di quantizzazione, che è possibile vedere come un possibile partizionamento dei centri di quantizzazione $c_1, \cdots, c_{N_c}$. Successivamente è possibile definire una misura t_i di adiacenza temporale su D_i come:

$$t_i = \frac{1}{n_i^2} \sum_{k \in D_i} \sum_{m \in D_i} T(k,m) \tag{1.11}$$

dove n_i è il numero di elementi della partizione D_i e $T(k,m)$ denota la $(k,m)^{esima}$ voce della matrice di adiacenza temporale T. Nell'equazione 1.11, t_i può essere pensata come la misura della somiglianza media temporale dei centri di quantizzazione

[2]Una matrice quadrata si definisce stabile se ogni autovalore ha una parte reale negativa. La matrice si definisce stabile positivamente se ogni autovalore ha parte reale positiva.

all'interno del cluster D_i. Se i modelli spaziali all'interno di quel cluster sono frequentemente adiacenti nel tempo, i valori corrispondenti nella matrice di adiacenza temporale dovrebbero essere alti. Ciò porterebbe un alto valore di t_i. Basandosi su questa pre-partizione della misura di similarita', è possibile definire J:

$$J = (1/2)\sum_{i=1}^{N_g} n_i t_i \tag{1.12}$$

Idealmente si vorrebbero trovare l'insieme delle partizioni $D^*_{i=1\cdots n}$ che massimizza questa funzione obiettivo. Tuttavia, dato un insieme di centri di quantizzazione, il numero di partizioni possibili all'interno di quell'insieme è grande, e computazionalmente non è possibile risolvere questo problema attraverso una ricerca esaustiva. D'altra parte i metodi di approssimazione Greedy sono computazionalmente possibili, ma possono portare ad avere degli ottimi locali molto lontani dagli ottimi globali. Fortunatamente, l'architettura HTM è complessivamente robusta nei raggruppamenti sub-ottimali, e le soluzioni bubottime in ogni nodo lavorano bene nella rete HTM completa.

Sarà presentato il metodo di ricerca greedy, veloce ed estremamente semplice. L'algoritmo di raggruppamento ripete il seguente processo finché tutti i punti di quantizzano appartengano ad un gruppo:

1. Trovare i punti di quantizzazione, che non fanno ancora parte del gruppo, con le maggiori connessioni: il punto di quantizzazione con le maggiori connessioni è semplicemente il punto di quantizzazione al quale corrisponde la riga con la somma più grande nella matrice di adiacenza temporale.

2. Cercare gli N_{top} punti di quantizzazione maggiormente connessi a questo punto di quantizzazione. Il pooler aggiunge questi punti di quantizzazione al gruppo corrente. Solo i punti di quantizzazione che non fanno ancora parte di un gruppo vengono considerati.

3. Ripetere lo step 2 per ogni punto di quantizzazione appena aggiunto. Il processo termina:

- all'esaurimento dei punti di quantizzazione disponibili;
- al raggiungimento di una dimensione massima del gruppo;
- se non vi sono più punti di quantizzazione connessi.

In questo ultimo caso si può considerare la creazione del gruppo corrente terminata e ripartire dallo step 1 con la formazione di un nuovo gruppo. Tale processo si ripete fino a che tutti i nodi non appartengono ad un gruppo.

Questo processo è illustrato in Fig.1.9, 1.10, 1.11, in cui la matrice di adiacenza temporale è visualizzata come un grafo. Le linee più spesse rappresentano i valori più alti nella matrice. Il ruolo del pooler è di dividere questo grafo al fine di raggruppare i punti di quantizzazione che appartengono alla stessa causa. Si assume che i punti di quantizzazione con collegamenti più forti rappresentino probabilmente la stessa causa rispetto a quelli con collegamenti più deboli o non collegati. In questo esempio, il parametro N_{top} è impostato a 2.

Quando ogni punto di quantizzazione é assegnato ad un gruppo, l'apprendimento è completo. La matrice di adiacenza temporale è mantenuta per uno scopo di debugging, ma non viene piu‘ utilizzata.

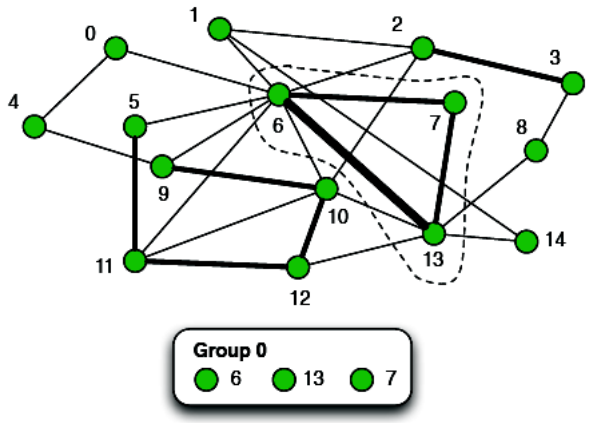


Figura 1.9: Prima iterazione dell'algoritmo di raggruppamento temporale: il gruppo formato usa il punto di quantizzazione maggiormente frequente, 6, come seme. Il pooler trova i due punti di quantizzazione più fortemente connessi con il punto 6, i punti 13 e 7, e li aggiunge al gruppo. Il pooler successivamente esamina i punti più vicini al punto 13, che sono i punti 6 e 7, ma questi fanno già parte del gruppo. Fa la stessa cosa per il gruppo 7, e trova i centri di quantizzazione 6 e 13. Quando non ci sono più punti, il pooler si ferma e il gruppo 0 sarà completo, con tre punti. Questi punti vengono rimossi dal grafo.

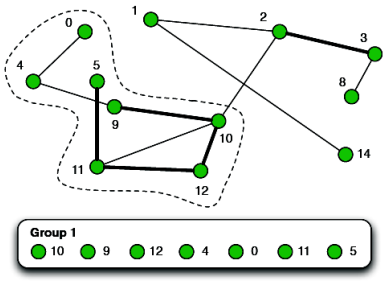


Figura 1.10: Seconda iterazione dell'algoritmo di raggruppamento temporale: il gruppo formato usa il centro di quantizzazione più frequente rimanente, il 10, come seme. I punti vicini al 10 sono i punti 9 e 12, che vengono aggiunti al gruppo. I punti più vicini al punto 9 sono i punti 10 (già nel gruppo) e il 4.I punti più vicini al 4 sono il 9 (già nel gruppo) e lo 0. Il punto 0 ha solo un punto vicino, che è già nel gruppo. Il punto 12 ha un nuovo punto vicino il punto 11, che ha un nuovo vicino il punto 5, che non ha nuovi punti vicini. Il gruppo è completo, e questi punti vengono rimossi dal grafo.

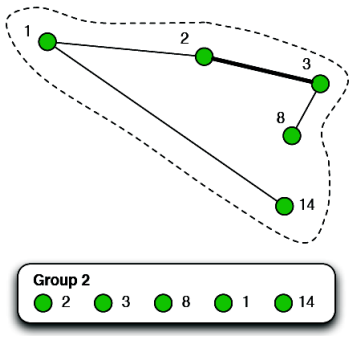


Figura 1.11: Terza ed ultima iterazione dell'algoritmo di raggruppamento temporale. È facile vedere come tutti i punti rimanenti vengono raccolti nel prossimo gruppo, perchè non ci sono più punti con più di due collegamenti.

Una volta che sono stati formati i gruppi temporali, il temporal pooler può iniziare a calcolare le uscite dei nuovi ingressi. Sia N_g il numero di gruppi temporali; allora l'output del temporal pooler è un vettore di dimensione N_g. L'output può essere considerato come la probabilità di distribuzione nello spazio dei gruppi temporali. Considerando il caso senza rumore, i modelli in ingresso al nodo coincidono esattamente con i centri di quantizzazione nello spatial pooler. Lo spatial pooler trova l'indice del centro di quantizzazione attivo in quel momento prendendo il valore massimo dell'ingresso attuale. Successivamente il temporal pooler trova l'indice del gruppo temporale a cui appartiene questo indice di quantizzazione. L'uscita del temporal pooler è costituita da tutti zero eccetto il valore corrispondente all'indice del gruppo temporale a cui appartiene il centro di quantizzazione.

1.5 Corrispondenza biologica delle Hierarchical Temporal Memory

Rimane da spiegare come l'implementazione neuronale astratta vista in precedenza sia organizzata fisicamente in strati e colonne nell'attuale anatomia corticale.

Una zona della corteccia può essere pensata come la codifica di una serie di modelli e di sequenze in relazione a modelli e sequenze di regioni gerarchicamente al di sopra e al di sotto di essa[1]. I modelli corrispondono ai modelli di coincidenza in un nodo HTM e le sequenze corrispondono alle catene di Markov. Un nodo HTM, come descritto in precedenza, codifica un insieme di modelli mutuamente esclusivi e catene di Markov. Una regione della corteccia che ha diversi modelli contemporaneamente attivi sarà implementata utilizzando diversi nodi HTM. La Fig.1.12[D] mostra l'implementazione HTM della gerarchia logica corticale mostrata in Fig.1.12[C].

Questa situazione corrisponde ad uno dei principi fondamentali dell'organizzazione del sistema visivo in cui i neuroni nelle aree visive di livello superiore ricevono gli ingressi da molti neuroni con campi ricettivi più piccoli in aree visive di livello inferiore. In questo mapping molto semplificato, l'area V1 è implementata usando 4 nodi HTM mentre l'area V2 è implementata usando 2 nodi HTM. Tipicamente, il numero di modelli non esclusivi che è necessario mantenere decresce man mano

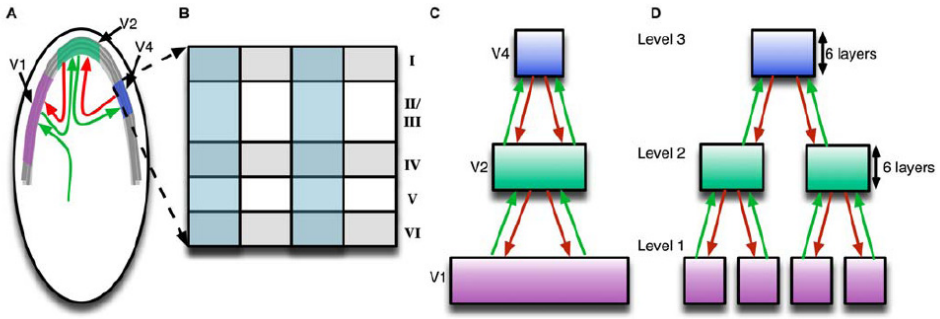


Figura 1.12: Mapping tra la gerarchia della neocorteccia e la gerarchia della HTM.

che si sale nella gerarchia. Per questo motivo, le regioni corticali a livello superiore possono essere modellate usando un ristretto numero di nodi HTM.

I modelli di coincidenza e le catene di Markov in un nodo HTM possono essere rappresentati usando variabili casuali. Una colonna corticale può essere pensata come la codifica di un particolare valore della variabile casuale che rappresenta i modelli di coincidenza nel nodo HTM. Le connessioni di feed-forward e di feedback ad un insieme di colonne corticali trasportano i messaggi di belief propagation. L'informazione osservata in qualche parte della neocorteccia viene propagata nelle altre regioni attraverso questi messaggi e può alterare i valori di probabilità associati con le ipotesi mantenute da altre colonne corticali. Nelle HTM questi messaggi vengono calcolati usando la matematica della belief propagation bayesiana introdotta nel precedente paragrafo.

La Fig.1.13 descrive la funzione, la connettività e l'organizzazione fisica degli strati corticali e delle colonne.

Questa figura corrisponde all'implementazione del circuito corticale laminare e a colonne delle equazioni di belief propagation per il nodo HTM standard in Fig.1.3 come proposto da D.George in [4]. La Fig.1.13 è stata creata organizzando i neuroni dell'implementazione neuronale astratta della belief propagation HTM in colonne e lamine in modo tale che il circuito risultante corrisponda alla maggior parte delle

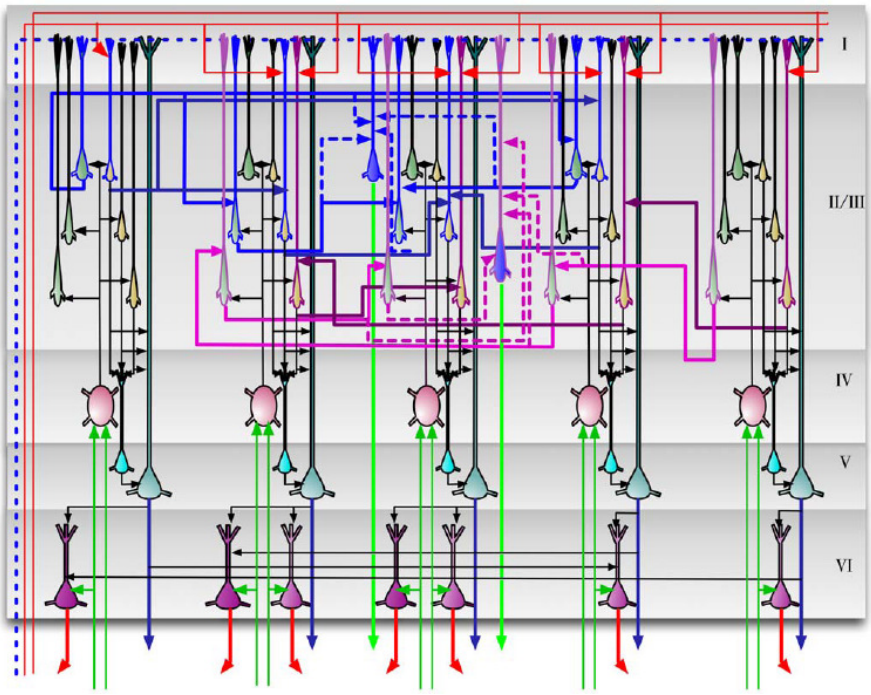


Figura 1.13: Una istanziazione laminare biologica delle equazioni di belief propagation bayesiana usate nei nodi HTM.

strutture che si trovano nella neocorteccia dei mammiferi. I circuiti in Fig.1.13 forniscono una istanziazione paradigmatica dei calcoli bayesiani nei circuiti corticali biologici laminari e a colonne. Si possono trovare diverse variazioni ed eccezioni a questo schema a causa dei gradi di libertà nell'implementazione delle equazioni di belief propagation e a causa dell'incompletezza dei dati anatomici.

Il circuito mostrato in figura 1.13 è organizzato in 5 colonne che corrispondono ai 5 modelli di coincidenza nel nodo HTM standard analizzato in Fig.1.3. I neuroni in ogni colonna rappresentano alcuni aspetti del modello di coincidenza che la colonna rappresenta. Per esempio, i neuroni nello strato II/III rappresentano il modello di coincidenza nel contesto di differenti sequenze, mentre i neuroni nello strato 6 rap-

presentano la partecipazione del modello di coincidenza nel calcolo dei messaggi di feedback. Le strutture a 5 colonne rappresentano un insieme di 5 ipotesi mutuamente esclusive nello stesso spazio dell'ingresso. I 5 modelli di coincidenza potrebbero corrispondere a differenti orientazioni di una linea. Se il campo recettivo è abbastanza piccolo, le differenti orientazioni possono essere considerate mutuamente esclusive, l'attività di una riduce l'attività dell'altra.

Ogni singolo elemento che deve essere rappresentato per un modello di coincidenza viene rappresentato usando un singolo neurone. Per esempio c'è esattamente un neurone che rappresenta il modello coincidenza 1 nel contesto della catena di Markov 1. Ciò significa che non c'è ridondanza in questa rappresentazione corticale idealizzata. Inoltre non cambia nulla nel calcolo o nella rappresentazione se si replica ogni neurone in questo circuito preservando le la connessioni. Una coincidenza che è rappresentata da un singolo neurone nella colonna corticale può essere rappresentata tramite un cluster di neuroni interconnessi lateralmente.

In questo modello molti dei collegamenti all'interno di una colonna verticale di cellule possono essere fissati senza apprendimento. La Fig.1.14[A] mostra una singola colonna idealizzata. I collegamenti all'interno di questa colonna che possono essere fissati a priori sono indicati in nero. Queste connessioni agiscono come backbone per il trasporto dei calcoli di belief propagation. La colonna corticale idealizzata corrisponde a ciò che è noto con il nome di *mini-colonne* o *colonne ontogenetiche* della corteccia. Le mini-colonne sono unità di sviluppo che contengono circa 80- 100 neuroni. Dalla ventiseiesima settimana gestazionale, la necorteccia umana è composta da un gran numero di mini-colonne disposte in array verticali paralleli. Nel cervello reale non si vuole rappresentare qualcosa con una singola cella. Pertanto, si ipotizza che nel cervello reale la colonna computazionale di base sia costituita da molte cellule ridondanti legate insieme con ingressi comuni e connessioni a corto raggio come in Fig.1.14[B].

I neuroni presenti nello strato IV vengono chiamati, per la loro forma a stella, "neuroni stellati" ed implementano il calcolo della probabilità feed-forward sui modelli di coincidenza. Essi sono rivelatori di coincidenza e le loro sinapsi rappresentano modelli di co-occorrenza in corrispondenza degli ingressi. Nella Fig.1.13 i

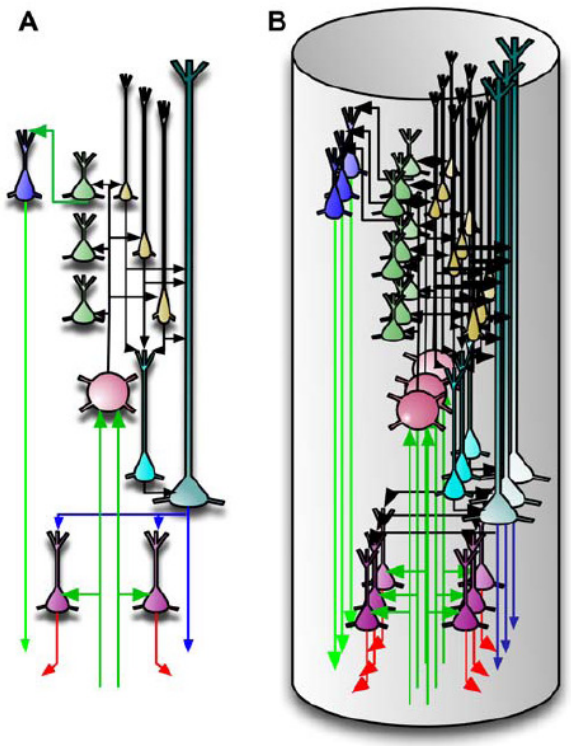


Figura 1.14: Organizzazione colonnare del microcircuito.

neuroni dello strato IV sono mostrati in rosso. Nello strato I invece sono presenti le connessioni di feedback dai livelli più alti della corteccia ed esso viene definito come strato *broadcast* per l'informazione temporale e di feedback. Nello strato II e III sono presenti invece i "neuroni piramidali" che svolgono differenti ruoli:

1. calcolo della sequenza degli stati feed-forward della catena di Markov;

2. proiezione dell'informazione della catena di Markov ad aree corticali di livello più alto;

3. calcolo della sequenza di stati che incorporano l'informazione di feedback.

Nello strato V avviene l'implementazione del calcolo del belief per mezzo di neuroni di colore ciano chiaro come mostrato in Fig.1.13. Essi ricevono gli ingressi dai neuroni di colore giallo presenti negli strati II/III. Infine nello strato VI avviene il calcolo dei messaggi di feedback per i figli in regioni che sono gerarchicamente più in basso. In Fig.1.13 i neuroni che compiono questo lavoro sono rappresentati di color porpora.

Capitolo 2

Riconoscimento delle attività umane

Non è forte colui che non cade mai,
ma colui che cadendo si rialza

– Johann Wolfgang von Goethe

Nel corso degli ultimi anni gli avanzamenti tecnologici, la miniaturizzazione e diffusione di dispositivi e sensori [16] e l'incremento delle prestazioni delle CPU hanno stimolato la ricerca sugli *ambienti intelligenti*, in particolar modo riguardo al monitoraggio delle persone e alle loro interazioni con l'ambiente circostante. Questa applicazione riguarda l'osservazione e la classificazione delle attività compiute dalle persone, con lo scopo di assisterle per individuare delle situazioni potenzialmente pericolose.

Varie tecniche possono essere utilizzate per analizzare le attività umane. Un grande numero di questo genere di sistemi utilizzano esclusivamente telecamere ([17, 18]),mentre altri sistemi utilizzano differenti tipologie di sensori: indossabili (e.g. accelerometri [19, 20, 21] o microfoni [22]) o passivi [23]. Da un punto di vista, sensori ambientali, come le telecamere, hanno il vantaggio di poter monitorare vari soggetti

contemporaneamente senza richiedere che ognuno di essi indossi alcun dispositivo. Nonostante ciò il loro svantaggio principale è la necessità di strutturare l'ambiente ed equipaggiarlo con almeno una telecamera calibrata per stanza. Inoltre, l'analisi di immagini può essere estremamente complesso, dipendente dalle condizioni di illuminazione, dalla disposizione dell'arredamento, dalla disposizione delle telecamere e dall'affollamento dell'ambiente.

Da un altro punto di vista, i sensori indossabili, come gli accelerometri wireless, hanno vari vantaggi rispetto alle telecamere: sono specifici rispetto alla persona che li indossa e i dati misurati sono indipendenti dalle condizioni ambientali. Inoltre i dati forniti sono di solito in un formato di più semplice elaborazione. Anche se richiedono un ambiente meno strutturato (è necessaria la presenza di un nodo ricevitore all'interno dell'area di copertura del segnale di trasmissione), essi impongono che ogni soggetto sottoposto a monitoraggio debba indossare un sensore. Un altro svantaggio è la necessità di rimpiazzare le batterie.

Recentemente si é sviluppato un grande interesse per la progettazione di architetture costituite da differenti tipologie di sensori, al fine di ottenere una maggior robustezza del sistema e superare gli svantaggi delle architetture a singola tipologia di sensori. Per esempio Lester et al. [24] utilizzano un dispositivo di acquisizione dati molto piccolo che include otto differenti tipologie di sensori, Leone et al.[25] cercano di individuare le cadute utilizzando tre sistemi indipendenti, utilizzando una range-camera 3D a tempo-di-volo, un accelerometro ed un microfono. Entrambi i sistemi non cercano però di riconoscere le attività umane utilizzando sia sensori indossabili che telecamere allo stesso tempo.

Un altro possibile campo di ricerca è la creazione di un ambiente intelligente che capisca le azioni e i comportamenti delle persone e le assistea nel completare i loro compiti di routine. Per esempio, il lavoro di Rashidi et. al [26] sfrutta un ambiente altamente sensorizzato, ma senza telecamere, per individuare il comportamento delle persone in maniera estremamente dettagliata.

L'obiettivo a lungo termine di questa linea di ricerca è quello di creare un sistema intelligente che, senza interferire nelle attività quotidiane, sia in grado di individuare gli eventi e le azioni compiute in maniera tempestiva e possa fornire delle predizio-

ni affidabili per anticipare ed evitare eventi traumatici agli utenti del sistema, oltre ad aumentarne il comfort. Il sistema potrebbe anche essere utilizzato per mantenere sotto controllo delle persone anziane all'interno di una struttura di assistenza per individuare eventi potenzialmente pericolosi o per individuarne il grado di indipendenza durante la degenza.

Un aspetto comune ai sistemi successivamente proposti è la natura dei dati analizzati. Tutti i dati, dopo una fase di preprocessing più o meno complessa, saranno sequenze monodimensionali di valori. Ogni evento sarà descritto come un inseme di vettori di dati monodimensionali rappresentanti l'evoluzione temporale dello stesso.

2.1 Classifcazione di movimenti semplici tramite accelerometro

Un primo sistema per il rilevamento e la classificazione dei movimenti impiega un accelerometro wireless per la raccolta dati e una Hierarchical Temporal Memory, in congiunzione con una Support Vector Machine, per classificare i movimenti effettuati nelle categorie di "saltare", "camminare", "stare fermi" e "cadere".

2.1.1 Support Vector Machines

Una Support Vector Machine riceve come ingresso un vettore di dati che è mappato, tramite una *kernel function*, su uno spazio multi-dimensionale (iperspazio). A questo punto, tutti i dati appartenenti alla stessa categoria sono separati dai dati appartenenti ad altre classi tramite iperpiani definiti *support vectors*. Infine è calcolato l'insieme di iperpiani che massimizza la distanza tra quelli che delimitano le classi. Si conclude che ogni SVM è definita dai suoi *support vectors*. Durante la fase di addestramento, la SVM mappa i dati nell'iperspazio e cerca il partizionamento ottimale. Al termine di tale fase, la classificazione di nuovi elementi è data dalla mappatura del dato sull'iperspazio attraverso la *kernel function* per determinare a quale regione appartiene, e

da quella la classe [27]. Negli ultimi anni, le SVM sono state riconosciute come una tecnica di classificazione efficace [28, 29, 30, 31, 32].

2.1.2 Classificatore Ibrido HTM/SVM

L'uscita del nodo al vertice di una rete HTM non è differente dai segnali che circolano all'interno della rete: essi descrivono le probabilità che l'input corrente appartenga ad ogni gruppo (causa) memorizzato nel nodo. Per poter rimappare questi valori su un numero conosciuto di categorie è necessario aggiungere un livello di classificazione supervisionata. Questo passo è necessario poichè l'algoritmo di addestramento delle Hierarchical Temporal Memories è non supervisionato e i gruppi formati non possono sempre essere in relazione diretta con le categorie desiderate. L'uscita di tale nodo è stata fornita come ingresso ad una Support Vector Machine assieme alle etichette di categoria per descrivere la tipologia di evento in corso. Da questo punto di vista è possibile considerare la HTM come un filtro di preprocessing che aggiunge informazione temporale rispetto alla classificazione *stateless* tipica di una Support Vector Machine.

È stato dimostrato che la combinazione di HTM e SVM migliora le prestazioni rispetto ad una semplice SVM, soprattutto in presenza di rumore nei segnali di ingresso [13].

Piattaforma Hardware

La piattaforma di raccolta dati consiste in un modulo wireless a basso costo che incorpora un accelerometro triassiale ad alta risoluzione e un trasmettitore RF IEEE 802.15.4. Tale modulo invia in tempo reale valore di accelerazione misurati ad una stazione base attrezzata con un ricevitore compatibile. Le caratteristiche principali di questo modulo sono riportate in tabella 2.1. I moduli sono stati interamente progettati e sviluppati da Henesis s.r.l. e sono utilizzati per una piattaforma di monitoraggio distribuito all'interno di WISNP[1]. La figura 2.1 mostra il modulo utilizzato per l'acquisizione dei dati accelerometrici.

[1] Wireless Sensor Network Platform: http://wisnp.henesis.eu

Figura 2.1: Modulo wireless con accelerometro posizionato su una banda medicale elastica

Setup applicazione

In questa applicazione è stato applicato un solo accelerometro per persona, fissato fermamente con una banda medicale elastica sulla parte terminale dello sterno della persona sotto monitoraggio. Tale posizione è stata scelta per meglio studiare i movimenti dell'intero corpo poiché si trova il più vicino possibile al centro di massa del corpo [33], è in una posizione che non ostacola alcun tipo di movimento ed è possibile fissarlo stabilmente.

Il modulo wireless è in grado di campionare ed inviare i dati dell'accelerometro fino a 640hz. L'acquisizione è stata eseguita a 160hz: un compromesso tra la quantità di dati da classificare e l'accuratezza della descrizione dei movimenti più veloci, come una caduta.

Durante gli esperimenti sono stati raccolti dati da tre differenti volontari, uomini ed in buone condizioni di salute ma con differenti caratteristiche di peso ed altezza. I volontari hanno compiuto i movimenti richiesti della durata di pochi secondi e successivamente i dati sono stati etichettati per poter misurare le prestazioni della classificazione. Sono stati raccolti almeno 150 eventi per ognuna delle seguenti tipologie: *stare fermo, saltare, camminare avanti, cadere, sedersi, rialzarsi da seduto, raccogliere un oggetto a terra, girarsi a sinistra, girarsi a destra, camminare*

Tabella 2.1: Wireless Sensor Module: caratteristiche tecniche

Accelerometer	ST LIS3LV02DL; BW: up to 640Hz; range: $\pm 2.0g$; max res: 1mg
Microprocessor	MICROCHIP PIC18F67J11; Max CPU Freq: 40MHz
Transmission range	outdoor 100mt; indoor: 10mt (0dBm output power)
Dimensions	h:60 mm X w:39 mm X t:10 mm with $Li - SOCI^2$ ER14250 battery
Memory	on node flash memory for local storage: 16Mb FLASH, 1Mb EEPROM
On-board sensors	on node temperature and humidity sensor (SHT11 digital sensor)
Expandability	SPI, I2C, RS232, Analog IOs for additional sensors

all'indietro per un totale di 1755 eventi ovvero circa 71 minuti di registrazione di movimenti.

Per una prima valutazione dell'approccio è stato deciso di utilizzare solamente i movimenti appartenenti alle categorie *stare fermo,saltare,camminare avanti e cadere* e limitare il numero di eventi da considerare: 20 eventi per ogni categoria nel *training set* e 10 eventi per ogni categoria nel test set. Tale scelta è stata effettuata per accorciare i tempi di addestramento senza perdere troppa generalità sui risultati ottenuti.

Preprocessing dei dati

Per addestrare correttamente una Hierarchical Temporal Memory è necessario fornire dati che rappresentaino pattern spazio-temporali ripetuti, senza alcun tipo di categoria. Al contrario, per addestrare il classificatore SVM supervisionato al vertice della rete HTM è necessario considerare l'informazione di categoria.

Il preprocessing si è limitato a individuare l'esatto istante di inizio e di fine di un evento, separandolo dalla parte iniziale e finale quando il soggetto era fermo. In figura 2.2 sono rappresentati alcuni dei dati utilizzati per l'addestramento. È possibile osservare in tale figura come gli assi dell'accelerometro non siano perfettamente allineati: anche quando il soggetto è fermo il valore medio di (x,y,z) è differente da (0,-1,0).

Questo avviene poiché la posizione dell'accelerometro wireless non può essere allineata perfettamente. Un possibile soluzione al problema è applicare una matrice di rotazione 3D per compensare questo errore e permettere una auto-calibrazione del sistema ogni volta che viene indossato. Nonostante ciò, in tale esperimento questa soluzione non viene applicata e si è deciso di lasciare al sistema di classificazione il compito di essere *robusto* rispetto a questo genere di rumore.

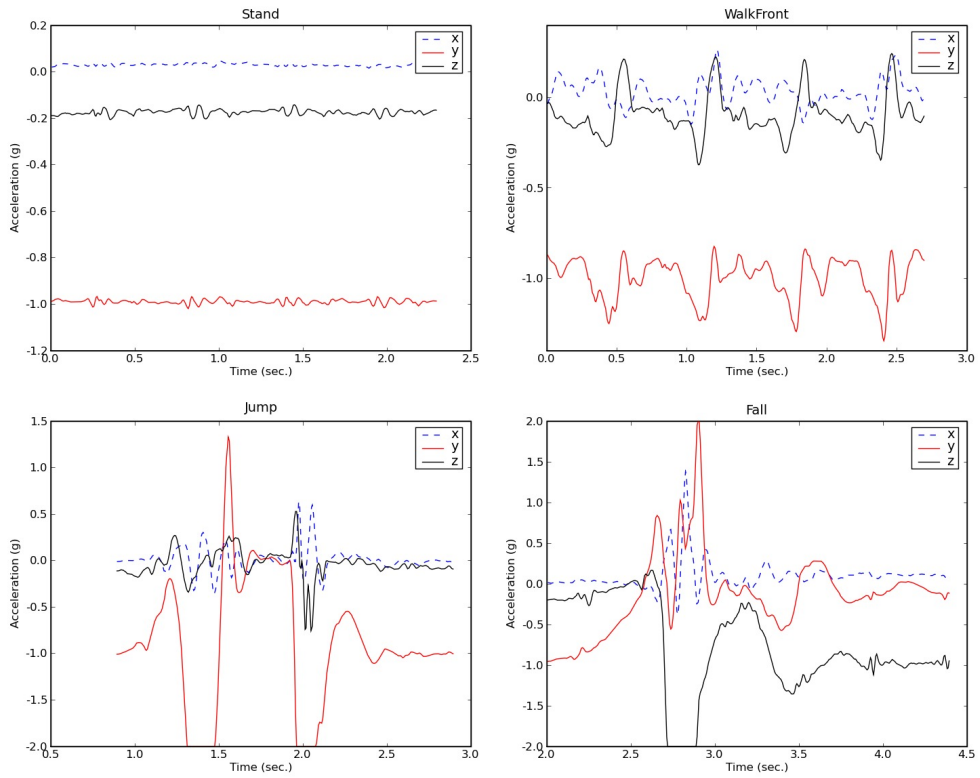


Figura 2.2: Grafico dei valori di accelerazione sui tre assi per ogni classe considerata nell'esperimento.

Progetto del sistema di classificazione

La rete HTM progettata è composta da 2 livelli. Nel primo livello è presente un nodo HTM differente per ogni asse dell'accelerometro; in questa maniera ognuno dei tre nodi potrà imparare e riconoscere solamente gli eventi che avvengono sul suo specifico asse. Il secondo livello è composto da un singolo nodo HTM che riceve in ingresso tutte le uscite dei tre nodi sottostanti; tale nodo è in grado di trovare pattern sull'intero campo percettivo. L'uscita di tale nodo viene poi utilizzata per addestrare un classificatore supervisionato, nel caso specifico una Support Vector Machine. Tale architettura è rappresentata in figure 2.3.

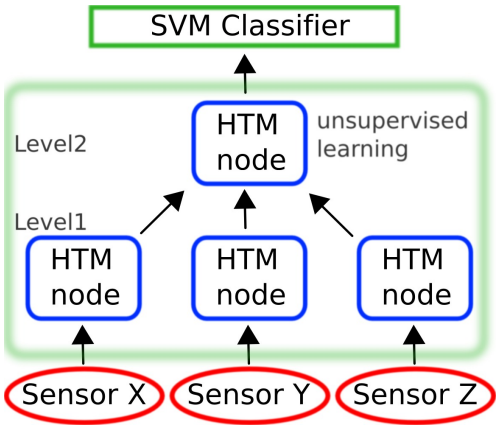


Figura 2.3: Archiettura per il riconoscimento di semplici movimenti

In particolare, i nodi al primo livello della rete HTM sono in grado di riconoscere brevi movimenti, come un incremento di accelerazione su un singolo asse, mentre il nodo al secondo livello è in grado di riconoscere eventi più lunghi, come uno o più passi durante una camminata o una parte di un salto. Come precedentemente descritto non è possibile programmare in senso classico una rete HTM ma solamente presentare i dati alla stessa in modo da memorizzare dei pattern.

Risultati sperimentali

Per valutare le prestazioni di tale approccio è stata confrontata l'architettura proposta col solo utilizzo di una Support Vector Machine. I risultati presentati riguardano la classificazione dell'insieme degli eventi del test-set presentati prima singolarmente, poi come un flusso continuo di dati ed infine aggiungendo del rumore bianco gaussiano.

In tabella 2.2 sono riportati i risultati sul test eseguito classificando gli eventi interi presi singolarmente. La classe predetta è stata quella che ha ricevuto il numero maggiore di attivazioni durante l'intero evento. Dalla tabella si evince come entrambi i sistemi abbiano una elevata accuratezza sul test set; inoltre questo tipo di test non è effettuabile in *real time* poiché richiede che gli eventi siano separati.

	Stand	Jump	WalkFront	Fall
Stand	96.29%	0%	3.70%	0%
Jump	1.85%	98.14%	0%	0%
WalkFront	1.85%	0%	98.15%	0%
Fall	0%	0%	0 %	100%

(a)

	Stand	Jump	WalkFront	Fall
Stand	98.15%	0%	1.85%	0%
Jump	0%	96.29%	3.70%	0%
WalkFront	0%	0%	100%	0%
Fall	0%	0%	0%	100%

(b)

Tabella 2.2: Matrice di confusione su eventi completi:(a) HTM+SVM (b) solo SVM

In una situazione maggiormente realistica il sistema di classificazione è continuamente esposto ad un flusso di informazione senza possibilità di segmentarlo a priori. Per simulare tale situazione abbiamo testato il classificatore su un flusso continuo di eventi differenti. L'uscita predetta ed attesa sono state raggruppate in *sliding windows* della durata di 3 secondi (ovvero 480 timepoint che è la durata media di un evento) e con uno *step* di mezzo secondo (80 timepoint). L'uscita del classificatore è la clas-

se che è stata selezionata il maggior numero di volte durante la finestra corrente. I risultati sono presentati in tabella 2.3. I classificatori ottengono delle prestazioni leggermente inferiori al caso precedente poiché all'interno di una stessa finestra possono essere presenti due differenti eventi.

	Stand	Jump	WalkFront	Fall
Stand	96.81%	1.45%	0.58%	1.16%
Jump	3.67%	90.61%	5.71%	0%
WalkFront	11.80%	0%	88.19%	0%
Fall	2.30%	0%	0.66%	97.05%

(a)

	Stand	Jump	WalkFront	Fall
Stand	98.84%	0%	1.16%	0%
Jump	0.41%	86.12%	13.47%	0%
WalkFront	0%	0%	100%	0%
Fall	0%	0%	2.62%	97.38%

(b)

Tabella 2.3: Matrice di confusione su sliding windows di 3 secondi:(a) HTM+SVM (b) solo SVM

Una caratteristica importante di un sistema di classificazione è data dalla sua resistenza al rumore. L'ultimo test è stato eseguito sommando ai segnali di accelerazione un rumore gaussiano bianco con deviazione standard di 0.05 ovvero del 2.5% (dato il *range* dei segnali pari a ± 2). In tabella 2.4 sono presenti i risultati del sistema. È possibile notare come l'approccio ibrido HTM+SVM mantenga un livello di prestazioni più alto. Questo risultato proviene dalla natura temporale degli algoritmi di riconoscimento all'interno di ogni nodo HTM. In figura 2.4 è possibile osservare come variano le prestazioni dei due sistemi al variare della percentuale di rumore aggiunto.

In conclusione questi risultati mostrano le potenzialità dell'approccio e dell'utilizzo delle Hierarchical Temporal Memories per riconoscere un insieme di dati monodimensionali provenienti da sensori di accelerazione.

	Stand	Jump	WalkFront	Fall
Stand	99.42%	0%	0%	0.58%
Jump	3.67%	91.43%	4.90%	0%
WalkFront	4.80%	0%	95.20%	0%
Fall	2.30%	0%	0.33%	97.38%

(a)

	Stand	Jump	WalkFront	Fall
Stand	0%	0%	99.42%	0.58%
Jump	0%	84.49%	15.51%	0%
WalkFront	0%	0%	100%	0%
Fall	0%	0%	2.95%	97.05%

(b)

Tabella 2.4: Matrice di confusione sui dati sommati a AWGN stdev=0.05:(a) HTM+SVM (b) solo SVM

2.2 Architettura multi-sensore

Un sistema di rilevamento più avanzato utilizza una architettura multi-sensore composta da telecamere e sensori di accelerazione all'interno di un ambiente scarsamente strutturato dove le attività sono costantemente monitorate e automaticamente classificate in maniera indipendente dal soggetto sotto osservazione [34].

Il sistema utilizza quattro telecamere installate nella stanza monitorata e un accelerometro wireless indossato da ogni persona presente. Il sistema sfrutta le differenti caratteristiche di ogni sensore per massimizzare l'accuratezza di riconoscimento, migliorare la scalabilità ed affidabilità. Gli eventi ricercati sono movimenti complessi come «camminare», «sedersi», «saltare», «cadere», anche effettuati da più persone contemporaneamente.

Tale sistema utilizza la visione artificiale per individuare le persone all'interno della stanza ed effettuare una prima classificazione dell'attività in corso. Nel caso si rilevi una condizione di pericolo viene attivato il sensore di accelerazione eventualmente indossato dalla persona (la cui trasmissione wireless è normalmente disabili-

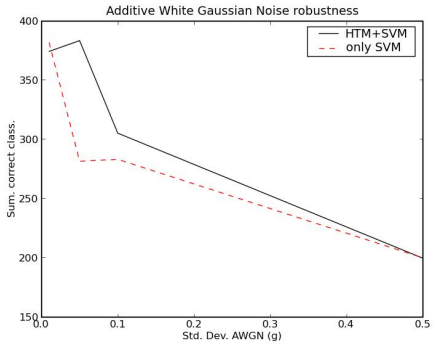


Figura 2.4: Comparazione della robustezza tra HTM+SVM e SVM aggiungendo rumore

tata per risparmiare batteria, ma in costante monitoraggio) per una seconda classificazione con una differente rete HTM. Nel caso di più persone presenti nella stanza è stato realizzato un prototipo di localizzazione tramite la scheda wireless per collegare la persona presente a video con il relativo accelerometro.

Una particolare attenzione è stata posta ad un efficace ed affidabile rilevamento delle cadute.

2.2.1 Sistema di visione artificiale

Il primo sottosistema si basa su quattro telecamere ma può essere esteso ad un numero arbitrario. Il sistema è stato realizzato in condizioni *indoor*, come una stanza, anche se non sono poste limitazioni a riguardo. In particolare la stanza utilizzata per implementare e testare il sistema è larga 5.4 metri e lunga 6.3 metri. In figura 2.5 è presente uno schema che la rappresenta.

Le quattro telecamere utilizzate (Firefly MV-03MTC by Point Grey [2]) sono state posizionate ai quattro angoli della stanza ad una altezza media di 2.3 metri e catturano

[2]http://www.ptgrey.com/products/fireflymv/fireflymv.pdf equipaggiate con lenti con apertura F1.4 e lunghezza focale di 2.8mm

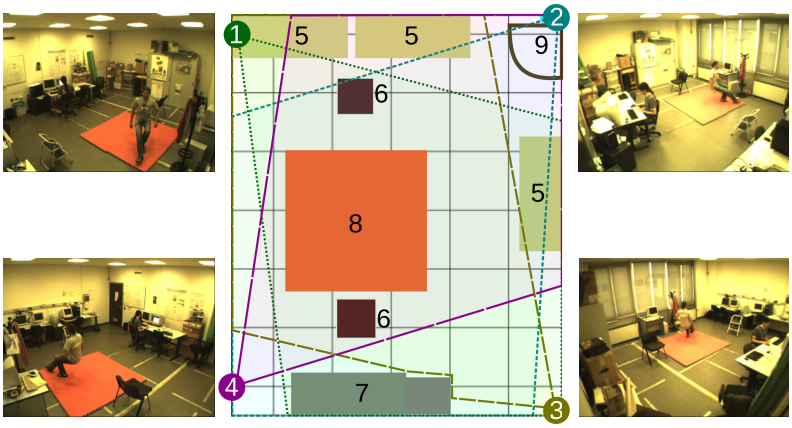


Figura 2.5: Mappa della stanza utilizzata per gli esperimenti. I quattro cerchi (numeri 1-4) agli angoli della stanza rappresentano le telecamere e le linee corrispondenti rappresentano gli estremi del loro *campo visivo*. Per ogni telecamera, la figura mostra una immagine catturata. Scrivanie e oggetti bassi sono identificati dal numero 5; le sedie dal numero 6; armadi e oggetti alti che limitano la visuale dal numero 7 , il numero 8 è il tatami utilizzato per ammorbidire l'impatto delle cadute dei soggetti, infine la porta da cui i soggetti entrano è il numero 9

immagini ad una risoluzione 640x480 pixel a 15 fps per ogni telecamera.

Le quattro telecamere sono state calibrate per poter mettere in corrispondenza tra loro viste differenti di un medesimo oggetto.

L'obiettivo del sistema di elaborazione delle immagini è quello di identificare le persone, seguirle nel tempo ed estrarre dei parametri che ne descrivano il moto da passare poi ad un classificatore che ne determini le azioni corrispondenti. Le operazioni sono state organizzate in un sistema modulare in cui le immagini di ogni telecamera sono analizzate indipendentemente in maniera da aumentarne l'efficienza (tramite parallelizzazione) e la robustezza, nel caso una telecamera avesse dei problemi o avesse la visuale occlusa.

I passi computi sono schematizzati in figura 2.6 e si compongono delle seguenti operazioni:

- figura 2.6(b): Individuazione dello sfondo tramite una *running average* e *background subtraction*. Sono state utilizzate le immagini in coordinate colore HSV per essere maggiormente indipendenti dalle condizione di illuminazione [35];
- figura 2.6(c): Segmentazione dell'immagine per calcolare il numero e posizione delle persone;
- figura 2.6(d): Calcolo del flusso ottico sull'intera immagine per individuare i movimenti presenti nella scena [36];
- figura 2.6(e): unione delle informazioni di movimento e posizione delle persone sulla scena.

L'immagine di ogni persona individuata sulla scena viene suddivisa in tre parti: testa, torso e gambe. Per ognuna di queste tre parti viene calcolato lo spostamento medio rispetto al frame precedente. Le quattro immagini che descrivono una persona vengono associate ad uno stesso oggetto tramite il criterio di prossimità. Se una vista non è presente per una persona saranno presenti un numero inferiori di dati nella fase di classificazione.

Prima di poter utilizzare i dati di movimento delle persone estratti tramite il flusso ottico è necessario effettuare una codifica che permetta al sistema di classificazione

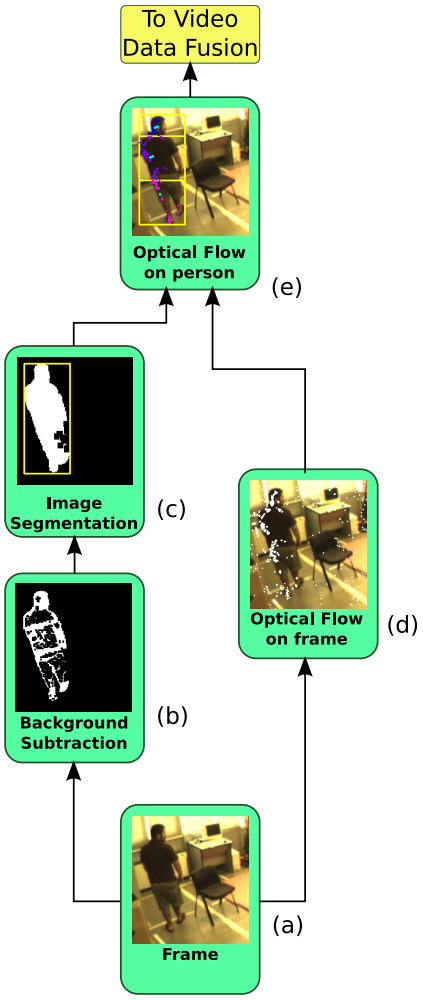


Figura 2.6: Schema elaborazione immagini: (a) immagine catturata dalla telecamera, (b) background subtraction, (c) segmentazione dell'image, (d) calcolo del flusso ottico sull'intera immagine, (e) calcolo del flusso ottico calcolato solo sulla persona

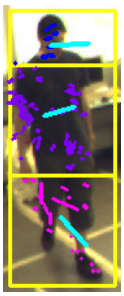


Figura 2.7: Rappresentazione di una persona: testa (superiore 20% , torso (successivo 40%) e gambe (inferiore 40%). Per ogni parte viene calcolato lo spostamento rispetto al frame precedente e visualizzato.

di poter individuare più facilmente pattern ripetibili. Il sistema di classificazione impiegato è un ibrido Hierarchical Temporal Memory associato ad una Support Vector Machine. Tale classificatore, come dimostrato precedentemente, è in grado di analizzare sequenze di dati monodimensionali da cui estrarre pattern ripetuti e classificarli. In particolare la codifica dei dati che viene utilizzata si compone di due passi:

- Conversione a coordinate polari del vettore movimento per ognuna delle tre parti della persona: tale codifica permette di ottenere delle sequenze di velocità e orientamento dello spostamento invarianti rispetto alla posizione assoluta della persona

- Quantizzazione di tali coordinate in modo da rendere insensibile la HTM a piccole variazioni. Questo passo non è obbligatorio ma rende il processo di classificazione maggiormente controllabile.

Al termine di questo passo di pre-processing i dati di movimento relativi ad ognuna delle tre parti del corpo sono rappresentati da due interi: velocità del movimento e direzione. Per ogni istante di tempo, per ogni persona e per ogni telecamera il classificatore riceverà in ingresso un vettore di 6 elementi.

Sistema di Classificazione

Il sistema di classificazione da utilizzare deve essere flessibile rispetto al numero di telecamere disponibili e al numero di persone nell'ambiente. Per questo motivo è stato realizzato un classificatore modulare.

L'unità di classificazione di base è denominata Camera-Classifier (figura 2.8 a sinistra) e processa i dati di una persona provenienti da una telecamera. Questa unità riceve in ingresso i sei numeri interi che descrivono direzione e velocità delle tre parti in cui è suddiviso il corpo di una persona. Tale modulo è composto da una Hierarchical Temporal Memory a due livelli e una Support Vector Machine come classificatore supervisionato in cima. I nodi HTM al livello inferiore memorizzano sequenze di movimenti di base di ogni parte del corpo separatamente mentre il nodo al secondo livello unisce tali informazioni per creare sequenze di movimenti di tutto il corpo. In particolare i nodi a basso livello di torso e testa hanno individuato 30 gruppi mentre il nodo di gambe ne ha individuati 25. Al secondo livello sono stati necessari 150 gruppi per descrivere i movimenti.

La SVM classifica i movimenti nelle seguenti categorie: camminare, fermi in piedi, sedersi, alzarsi dalla sedia, cadere, alzarsi da terra. Il Camera-Classifier è addestrato da dati provenienti da tutte le telecamere.

Ogni volta che un nuovo soggetto è identificato sulla scena viene instanziato un secondo tipo di classificatore: il Personal-Classifier. Un Personal-Classifier è composto da quattro Camera-Classifier (uno per telecamera) le cui quattro uscite sono processate da una SVM (figura 2.8). Questa Support Vector Machine agisce da decisore e calcola la classificazione finale del movimento. Il Personal-Classifier deve essere robusto alla mancanza di dati da uno (o più) dei quattro canali; risultati sperimentali presentanti di seguito dimostrano la bontà dell'approccio.

Questa architettura modulare permette di ottenere un alto grado di parallelizzazione che permette di eseguire il sistema in tempo reale con un numero di soggetti multiplo nella scena tramite un ordinario computer. Inoltre se il numero delle telecamere varia è sufficiente ri-addestrate solamente la Support Vector Machine del Personal-Classifier.

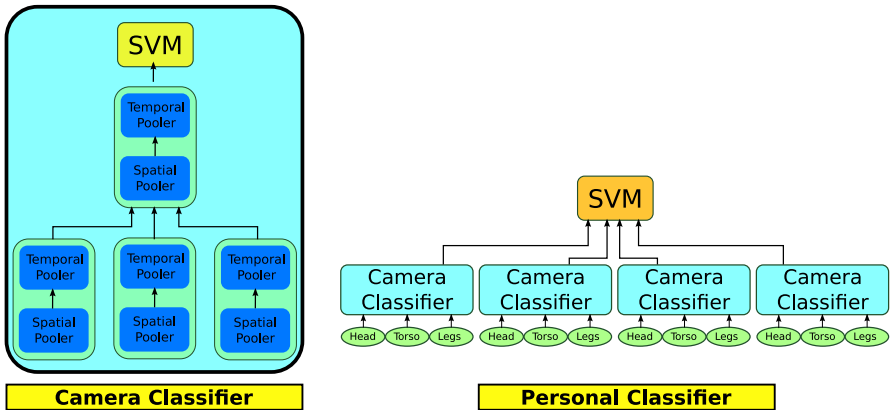


Figura 2.8: Schema del sistema di classificazione. A destra Personal-Classifier mentre a sinistra Camera-Classifier.

Risultati sperimentali

I testi sono stati effettuati utilizzando come server un computer composto da una CPU Intel Core i7 2.67Ghz quad-core con 4GB di RAM.

Il dataset creato per gli esperimenti è composto da dati acquisiti da 11 volontari (10 uomini e 1 donna), che hanno eseguito varie azioni in maniera continuativa all'interno della stanza, uno alla volta. La durata delle sequenze varia da 90 a 200 secondi. L'ordine delle azioni non è stato predeterminato. I dati provenienti da 4 persone sono state utilizzate come *training set* e le restanti come *testing set*.

In tabella 2.5 sono riportati alcuni risultati sul test set nella classificazione delle azioni mentre in tabella 2.6 raggruppando gli eventi in caduta/non caduta.

Per ogni frame della sequenza video è stata generata una classificazione e i risultati sono stati mediati su una finestra di 0.5 secondi per evitare errori dovuto a classificazioni errate isolate. Ad ogni finestra è stata associata una etichetta in base alla classe più frequente.

Nel secondo caso, un evento è stato definito caduta quando almeno una finestra che lo compone è stata etichettata come caduta. Questo comportamento differente è stato scelto data la brevità dell'evento caduta e della sua criticità. Dalla tabella 2.6 è

Tabella 2.5: Risultati in percentuale del sistema video, per l'individuazione di eventi sul test set.

action/class	stand	walk	sit	get up	fall	rise	n.r.
stand	83.7	0.0	0.0	4.1	2.0	6.1	4.1
walk	3.5	91.2	0.0	0.0	0.0	3.5	1.8
sit	9.5	0.0	76.2	0.0	0.0	0.0	14.3
get up	0.0	0.0	4.8	71.4	0.0	4.8	19.0
fall	0.0	4.8	4.8	0.0	85.6	0.0	4.8
rise	0.0	0.0	0.0	5.0	0.0	70.0	25.0

possibile notale come questo approccio non innalzi eccessivamente il numero di falsi positivi mentre permette di non avere falsi negativi.

Tabella 2.6: Risultati in percentuale del sistema video, specifico per l'individuazione delle cadute sul test set.

event	not fall	fall
stand	100	0
walk	94.7	5.3
sit	95.5	9.5
get up	100	0
fall	0	100

In conclusione questo sistema ottiene buoni risultati nel riconoscere e tracciare le persone e nel classificare le loro azione ma produce dei falsi allarmi nel riconoscimento delle cadute. Inoltre la tabella 2.6 mostra come al diminuire del numero di immagini disponibili si abbia un decadimento delle prestazioni quando mancano 2 o più immagini. Tali risultati sono ottenuti rieseguendo i test sugli stessi dati ma privando il sistema di alcune sorgenti di immagini.

2.2.2 Sistema basato su accelerometri

Il sistema impiegato è architetturalmente simile al sistema descritto nella sezione 2.1.

Il classificatore ibrido *Hierarchical Temporal Memory* / *Support Vector Machine* è stato riaddestrato sul training set utilizzato per il sistema video descritto precedentemente. In particolare il sistema è stato addestrato per riconoscere le categorie: stare fermi in piedi, camminare, sedersi, alzarsi da seduto, cadere, rialzarsi dal pavimento, essere seduti e restare stesi a terra.

La tabella 2.7 mostra i risultati della classificazione che appaiono meno accurati da quelli ottenuti dal sistema video.

Tabella 2.7: Risultati in percentuale del sistema basato su accelerometri. Riconoscimento di eventi sul test set.

action/class	stand	walk	sitdown	get up	fall	rise	sit	lie	n.r.
stand	25.0	58.3	0.0	0.0	0.0	0.0	16.7	0.0	0.0
walk	0.0	100.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
sit down	0.0	9.5	47.6	19.1	0.0	23.8	0.0	0.0	0.0
get up	0.0	14.3	33.3	47.6	0.0	4.8	0.0	0.0	0.0
fall	0.0	0.0	0.0	0.0	90.5	9.5	0.0	0.0	0.0
rise	0.0	0.0	5.0	0.0	25.0	55.0	0.0	5.0	10.0
sit	14.3	38.1	0.0	0.0	0.0	0.0	42.8	0.0	4.8
lie down	0.0	6.2	0.0	0.0	25.0	12.5	6.3	50.0	0.0

Nonostante ciò il sistema ha ottime prestazioni nel riconoscimento specifico dell'evento caduta. Se il sistema è utilizzato come un riconoscitore di cadute esso ottiene una prestazione ottima sul test set, come mostrato in tabella 2.8. È importante notare come questo sistema funzioni meglio del precedente su tale applicazione specifica. I dati sono stati raggruppati nella stessa maniera del sistema video.

I risultati sperimentali evidenziano ottimi risultati ma questa soluzione ha degli svantaggi pratici per una sua applicazioni nel mondo reale:

- un accelerometro che invia continuamente i dati consuma un quantitativo considerabile di energia e perciò richiede una frequente sostituzione delle batterie

Tabella 2.8: Risultati in percentuale del sistema basato su accelerometri. Riconoscimento cadute sul test set.

event	not fall	fall
stand	100	0
walk	100	0
sit	100	0
get up	100	0
fall	0	100

- il quantitativo di dati richiesto per una classificazione del movimento affidabile richiede una banda relativamente larga, che permette solo a pochissimi accelerometri (uno o due al massimo) di inviare dati contemporaneamente.

Nella sezione successiva viene presentata una architettura che integra il sistema video e quello basato su accelerometri per superare tale problema.

2.2.3 Sistema ibrido

L'obiettivo di questo sistema è quello di superare gli svantaggi dei sistemi a singolo sensore sviluppati precedentemente, in partcolare:

- prestazioni non sufficienti del sistema video nel riconoscere l'evento caduta
- limiti fisici del sistema basato su accelerometri (autonomia energetica e numero di monitoraggi contemporanei)

Il sistema ibrido sfrutta un monitoraggio costante per individuare eventi "pericolosi" tramite il sistema di visione artificiale per poi richiedere in caso di evento potenialmente "pericoloso" gli ultimi 2 secondi di dati mmorizzati all'accelerometro eventualmente indossato dalla persona per individuare se è stata rilevata una caduta. La figura 2.9 rappresenta tale sistema.

Per raggiungere questo obiettivo è necessario ottenere una associazione (e mantenerla sincronizzata) tra i soggetti identificati dal sistema video e i corrispondenti

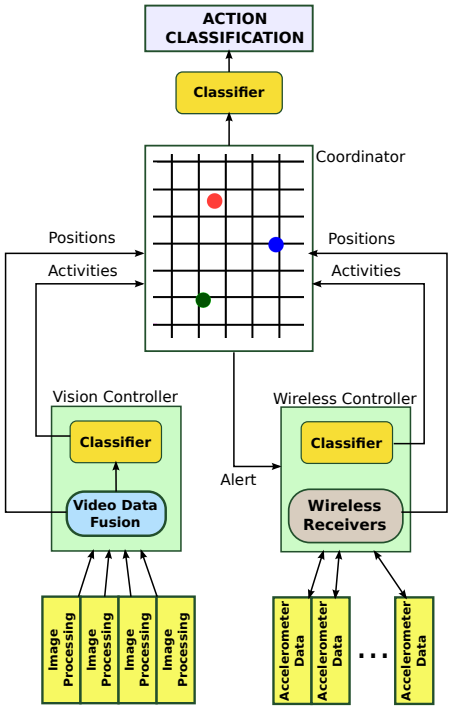


Figura 2.9: Architettura sistema multi-sensore.

accelerometri. Per ottenere tale unione dei due sottosistemi è necessario effettuare una cross-localizzazione delle persone all'interno della stanza. Il sistema video, grazie alla calibrazione delle telecamere, possiede già l'informazione di posizione delle singole persone; al contrario il sistema basato su accelerometri non possiede questa informazione, ma può sfruttare una localizzazione approssimativa tramite l'analisi della potenza del segnale radio ricevuto da ciascun ricevitore.

Localizzazione basata sulla trasmissione wireless

La localizzazione di un nodo wireless è stata ampiamente esaminata in letteratura (vedi ad es. [37]) ma, al presente, non esistono metodi robusti ed affidabili per determinarne la posizione. La maggior parte delle tecniche si basa sulla misura dell'intesità del segnale proveniente dai nodi che devono essere localizzati. I problemi principali in tale approccio sono:

- non-uniformità del pattern di radiazione dell'antenna, in particolare quando il modulo viene posizionato addosso ad una persona [38];
- localizzazione non-univoca dovuta ad un numero limitato di stazioni riceventi
- instabilità della misura della potenza ricevuta, specialmente se la persona si sta muovendo

Il primo problema è causato dall'assorbimento da parte del corpo della potenza di trasmissione e dalla posizione dei ricevitori. Infatti, se questi sono posizionati dietro il soggetto sottostimano la potenza ricevuta. L'orientamento della persona gioca perciò un ruolo fondamentale.

Una possibile soluzione crea una mappa dell'ambiente [39] misurando i livelli di potenza ricevuta da un numero limitato di posizioni. Successivamente, per riconoscere la posizione, si associa il livello di potenza ricevuto al più simile sulla mappa.

In attesa di ulteriori progessi in questo campo è stato implementato un sistema prototipale con una antenna omnidirezionale posizionata sul capo del soggetto in

modo da eliminare il problema dell'orientamento del soggetto ed è stata creata una mappa della stanza come descritto.

Inoltre il nostro sistema possiede già una informazione sulla probabile posizione dei nodi; infatti è sufficiente che il sistema wireless localizzi il soggetto tra quelli presenti sulla scena determinati dal sistema video.

Per valutare le prestazioni di tale sistema di localizzazione ibrido sono stati eseguiti dei test per distinguere tra due persone ferme a differenti distanze (2,3,4 e 5 metri). In tabella 2.9 sono riportati i risultati.

Tabella 2.9: Risultati in percentuale della localizzazione di due persone rispetto alla distanza reciproca.

distance	correct	wrong
2m	53	47
3m	70	30
4m	83	17
5m	100	0

É possibile notare come l'accuratezza cresca all'aumentare della distanza tra i soggetti e che per distanze maggiori di 3 metri tale informazione sia particolarmente affidabile.

Cooperazione multi-sensore

In questa nuova architettura il sensore wireless posizionato sul corpo del soggetto invia un *beacon* (segnale di presenza) ogni secondo. Questo segnale è ricevuto da altri moduli installati vicino ad ogni telecamera del sistema video (come mostrato in figura 2.10) ed è utilizzato per stimare la posizione della persona nella stanza tramite il confronto con le posizioni possibili date dal sistema video.

Durante il modo di funzionamento "normale" l'accelerometro è continuamente in modalità di registrazione e salva i dati in un buffer circolare (fino a 2 secondi) quando rileva un movimento e ogni secondo invia il *beacon* di presenza.

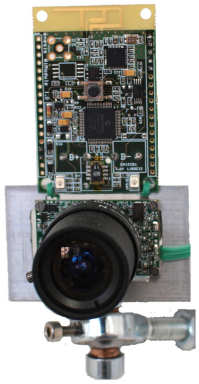


Figura 2.10: Componente del sistema multi-sensore: telecamera con un modulo ricevitore wireless.

Tale modalità di funzionamento permette di estendere notevolmente la durata della batteria rispetto ad un invio continuo di dati.

Quando il sistema video individua una situazione potenzialmente pericolosa il sistema invia un segnale al sensore interessato ed inizia ad inviare i dati dell'accelerometro, partendo da quelli memorizzati nel buffer circolare e fino a che non viene fermato. Se il soggetto non indossa un sensore accelerometrico viene utilizzato solamente il sistema video per la classificazione.

Risultati sperimentali

Il sistema presentato richiede un elevato numero di operazioni, perciò è stata sviluppata una architettura multi-thread unita a varie strategie di ottimizzazione. Grazie a questa implementazione si è potuti passare da 0.153 secondi per immagine (6.53 frame per secondo) per un sistema monoprocessore a 0.056 secondi per immagine (17.86 frame per secondo) su un processore quad-core. Il collo di bottiglia del sistema è la frequenza di acquisizione delle telecamere (15 frame per secondo).

Il dataset utilizzato per valutare le prestazioni del sistema è lo stesso utilizzato nei due sottosistemi distinti. I criteri di valutazione delle prestazioni sono:

- rilevamento corretto delle situazioni di pericolo e le relative attivazioni dell'accelerometro

- tempestività nell'attivazione degli accelerometri

La tabella 2.10 mostra le statistiche di attivazione dell'accelerometro per i vari eventi.

Tabella 2.10: Sistema Multi-sensore. Risultati in percentuale delle attivazioni dell'accelerometro per evento.

event/switch on	No	Yes
stand	92	8
walk	63	37
sit down	4	96
stand up	4	96
fall	0	100

La tabella 2.11 mostra i risultati di rilevamento di cadute. Una caduta può essere rilevata solamente se l'accelerometro ha potuto memorizzare ed inviare tutti i dati relativi all'evento; perciò la caduta deve avvenire entro i 2 secondi precedenti all'attivazione o appena dopo. Dalla tabella è possibile osservare come tutte le cadute del test set siano state correttamente rilevate e classificate.

Tabella 2.11: Sistema Multi-sensore. Risultati in percentuale dei rilevamenti corretti di caduta sul test set.

present/detected	Yes	No
Yes	100	0
No	0	100

In conclusione l'archiettura multi-sensore presentata per il riconoscimento e classificazione di attività umane (ed nello specifico cadute) ha raggiunto buoni risultati.

In particolare il sotto-sistema video raggiunge buoni risultati nella classificazione di eventi ma produce un numero troppo elevato di falsi positivi per le cadute. Inoltre le performance degradano se il soggetto è fuori dal campo di vista di 2 o più telecamere.

Il sotto-sistema accelerometrico ottiene ottimi risultati nel rilevare le cadute ma limiti fisici del sensore (autonomia e banda) non lo rendono ideale per monitorare un numero elevato di persone per un tempo lungo.

Perciò l'architettura multi sensore combina i due sottositemi per superare tali svantaggi ed essere in grado di riconoscere movimenti umani, ed in particolare le cadute, con buona accuratezza e tempestività.

Inoltre questo sistema dimostra nuovamente come il classificatore ibrido Hierarchical Temporal Memory e Support Vector Machine sia particolarmente adatto per riconoscere eventi che si svolgono nel tempo con buone caratteristiche di generalizzazione.

Capitolo 3

WSN per la raccolta e l'analisi di dati ambientali

3.1 Wireless Sensor Networks

Negli ultimi anni si sta assistendo ad un fiorente sviluppo delle reti wireless di telecomunicazioni; il mercato dei dispositivi senza fili è in continua crescita e grazie alla progressiva diminuzione dei prezzi, questa tecnologia, che un tempo veniva realizzata solo quando l'utilizzo di cavi fisici si rivelava problematico, sta ora sostituendo in diversi ambiti le reti cablate.

Data la natura locale e decentralizzata di queste reti, oltre alla mobilità che nella maggior parte dei casi caratterizza i nodi che le compongono, un aspetto di particolare importanza nell'implementazione di tali reti è costituito dalle tecniche di realizzazione delle interconnessioni tra i nodi (topologia di rete) e dai protocolli di instradamento (routing) utilizzati per la propagazione dei dati.

Una particolare categoria di reti wireless per la raccolta e distribuzione di informazione sono le cosiddette reti di sensori senza fili (Wireless Sensor Network, WSN). Esse sono composte da dispositivi a basso consumo energetico che comunicano tra loro senza bisogno di cavi e che, proprio per la scarsa alimentazione a disposizione, si accendono e si spengono per preservare il consumo di batteria. Tali sensori sono uti-

lizzati per un vasto spettro di applicazioni: militari, ambientali, medico-sanitarie, domestiche (automazione della casa) e anche commerciali (ad esempio, il rilevamento della posizione e del movimento dei veicoli).

Una ulteriore sottocategoria delle WSN sono le reti di monitoraggio distribuito. Tale tipologia di rete prevede un vasto numero di sensori, al più basso costo possibile, alimentati per la maggior parte a batteria, dispersi in un ambiente non strutturato. Tali sensori devono monitorare parametri ambientali e comunicarli a uno o più nodi concentratori che raccoglieranno l'infomazione per poi memorizzarla o spedirla via Internet ad un server centrale.

In tale scenario è importante mantenere le caratteristiche di reti wireless a basso consumo, di lunga autonomia e basso costo perciò la larghezza di banda e in generale la capacità computazionale dei nodi è limitata. Da questi vincoli è possibile desumere come non tutte le tipologie di sensori producano dati che possono essere raccolti e trasmessi tramite una rete di questo tipo. In particolare è molto difficile, e probabilmente non vantaggioso, inviare tramite una WSN delle immagini o del segnale audio non pre-elaborato. Le tipologie di sensori adatte alla raccolta tramite WSN sono quelle la cui uscita è codificabile con un basso numero di grandezze scalari che variano lentamente nel tempo, come ad esempio sensori di temperatura, umidità, pressione, luminosità, intensità sonora, concentrazione di sostanze chimiche, elongazione, attivazione (pulsanti), presenza e via dicendo.

L'insieme di dati raccolti tramite una Wireless Sensor Network descrive lo stato dell'area posta sotto monitoraggio e la sua evoluzione nel tempo.

Per poter analizzare questo insieme di dati è necessario utilizzare una tecnologia che permetta l'analisi di dati eterogenei, abbia una rappresentazione temporale dell'evoluzione dei fenomeni e che possa essere estrarre eventi e pattern a partire dai dati stessi senza bisogno di applicare delle regole fisse.

Le Hierarchical Temporal Memories, descritte le capitolo 1, hanno le caratteristiche desiderate, in più posseggono una struttura gerarchica particolarmente adatta a rappresentare fenomeni naturali complessi.

3.1.1 Lo standard IEEE 802.15.4

Lo standard IEEE 802.15.4 [40, 41, 42] definisce il protocollo di trasmissione a basso livello, tramite comunicazione radio, tra diversi dispositivi rientranti in una rete in area personale (Personal Area Network, PAN). Lo standard è in grado di garantire una modalità di connessione wireless a basso tasso di trasmissione tra dispositivi fissi, portatili o mobili che necessitano di un basso consumo di potenza, ovvero lunga durata delle batterie a bordo. Tali reti sono note in letteratura come reti personali senza fili a basso consumo di potenza (Low Rate Wireless Personal Area Network, LR-WPAN), ovvero reti di comunicazione semplicie a basso costo, selettivamente orientate verso applicazioni a basso consumo.

Lo standard definisce le specifiche del livello fisico (PHY) e di accesso al mezzo (Media Access Control, MAC) le cui caratteristiche principali sono:

- data-rate di 250 kbit/s, 40 kbit/s e 20 kbit/s;
- accesso al canale in modalità di ascolto del mezzo (Carries Sense Multiple Access with Collision Avoidance, CSMA/CA);
- 16 canali nella banda intorno a 2.4 GHz (e 10 canali attorno a 915 MHz e un canale attorno a 868 MHz)

Una rete può essere realizzata architetturalmente con diverse topologie come mostrato su figura3.1. Di particolare interesse sono le topologie a maglia (mesh), completamente connessa (fully connected), a catena (line) e ad albero (tree).

La topologia di tipo mesh è costituita da una struttura di tipo decentralizzato, senza quindi la presenza di server centrali, è molto adattabile e resistente grazie al fatto che ogni nodo deve solamente trasmettere un segnale al massimo fino al nodo successivo. Ogni nodo funge da ripetitore per trasmettere il segnale dai nodi più vicini ai peer, nodi equivalenti, che risultano troppo distanti per poter essere raggiunti. In questo modo, si crea una rete in grado di coprire grandi distanze. Inoltre, questo tipo di rete è molto affidabile poichè ogni nodo è connesso a molti altri nodi. Se un nodo viene meno alla rete, per qualsiasi motivo, i nodi vicini semplicemente

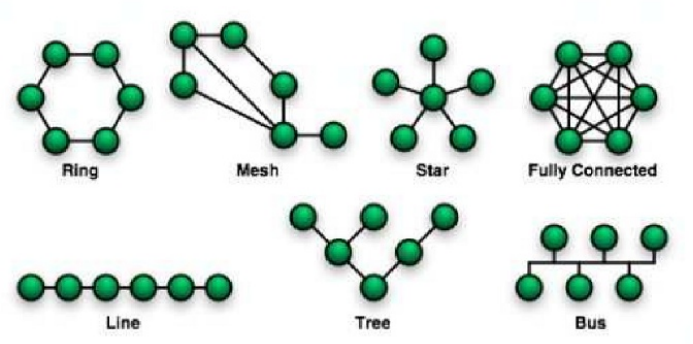


Figura 3.1: Topologie di rete per lo standard IEEE 802.15.4

cercano altri percorsi per trasmettere l'informazione. Le capacità della rete possono essere incrementate semplicemente installando altri nodi. L'utilizzo di tecnologie magliate permette di massimizzare l'affidabilità complessiva della rete, garantendo la possibilità di instradare l'informazione su percorsi diversi. Di contro utilizzano però protocolli di routing, gestione e sicurezza molto complessi e il mezzo fisico risulta condiviso provocando una riduzione della banda.

Parte molto importante del protocollo è sicuramente il livello MAC, approfondito in [40, 41], i cui principali compiti sono il coordinamento dell'accesso al mezzo da parte del trasmettitore, la creazione dei pacchetti, la generazione e il riconoscimento dell'indirizzo e la verifica del numero di sequenza dei pacchetti. Deve inoltre gestire il processo di rilevamento, da parte di un dispositivo, di quelli ad esso vicini.

Esistono 4 possibili tipi di frame a livello MAC:

- frame di dati
- frame ACK
- frame di comando MAC
- frame di beacon.

Il frame dati è costituito al massimo da 128 byte ed è numerato per assicurare l'instradamento di tutti i pacchetti.

Il frame ACK fornisce la conferma che il pacchetto inviato è stato ricevuto correttamente garantendo la consistenza dei dati ma, ovviamente, aumentando la latenza.

Il frame beacon ha il compito di informare i dispositivi nel raggio di trasmissione della presenza del nodo (utilizzato per sincronizzare i vari nodi)

Trattandosi di una trasmissione in cui il mezzo è condiviso da tutti i dispositivi, è necessario disporre di metodi di arbitraggio della trasmissione, affinché due dispositivi non trasmettano pacchetti contemporaneamente. Il CSMA/CA (Carrier Sense Multiple Access with Collision Avoidance) è un meccanismo per il quale ogni dispositivo prima di iniziare una trasmissione deve ascoltare il mezzo per rilevare una eventuale trasmissione già in corso. Se una trasmissione già in corso, sarà effettuata la ritrasmissione del dato in un tempo successivo con un ritardo casuale, secondo un algoritmo di backoff.

Lo standard IEEE 802.15.4 non definisce il comportamento dei vari nodi ed in particolare non definisce il livello di rete. Esistono vari standard o protocolli che sfruttano questo livello PHY e MAC, tra cui lo ZigBee.

Zigbee

Zigbee [43] è una suite di protocolli basata su IEEE 802.15.4 e, in particolare sui livelli fisico e MAC, che gestisce i livelli più alti, soprattutto il livello di rete. È lo standard più conosciuto per reti mesh di sensori e quindi per applicazioni a bassa potenza e limitato data-rate. Tale standard prevede la presenza all'interno della rete di tre tipi di nodi:

- nodi coordinatori,
- nodi router,
- nodi terminali.

Ogni nodo può inviare e ricevere dati ma sussistono specifiche differenze nel ruolo ricoperto da ognuno. Il coordinatore è uno solo per rete ed è quello con le maggiori

capacità in quanto può immagazzinare tutte le informazioni della rete incluse quelle di sicurezza.

I nodi router agiscono da intermediari rilanciando i dati provenienti da altri dispositivi.

Infine i terminali sono dispositivi a bassa potenza dotati di batteria (sensori) che possono parlare con router o coordinatore ma non possono rilanciare dati provenienti da altri dispositivi.

Limitazioni

Zigbee è costituito da differenti *profili* che cercano di coprire quasi tutti i casi d'uso di una rete di sensori senza fili, questo ha portato ad una relativa complicazione del protocollo stesso e alla difficoltà di implementazione in maniera compatibile da parte dei diversi prodottutori.

La maggiore limitazione del protocollo Zigbee è nel funzionamento dei nodi *router*. Tali nodi, nel funzionamento standard, devono mantenere il ricevitore sempre acceso in modo da poter ricevere in qualsiasi momento una comunicazione da un nodo *terminale* o da un altro nodo *router*.

Per ridurre questa limitazione sono state introdotte le reti *beacon-enabled* dove i nodi *router* inviano un *beacon* per comunicare quando sono attivi e per il resto del tempo possono entrare in fase di *sleep*. Nonostante ciò il protocollo non considera molti fattori necessari ad una effettiva minimizzazione del consumo, in particolare la necessità di una sincronizzazione tra i nodi estremamente precisa.

Nelle reti di monitoraggio ambientale distribuito è importante che tutti i nodi possano essere alimentati a batteria (per applicazioni outdoor, in luoghi senza corrente elettrica) inclusi i nodi *router*, ovvero quelli in grado di estendere la copertura della rete.

3.1.2 La piattaforma Hardware

La piattaforma hardware considearata, figura 3.2, è stata interamente progettata e sviluppata da Henesis s.r.l. ed è utilizzata per una piattaforma di monitoraggio di-

stribuito all'interno di WISNP[1]. Gli stessi moduli sono stati utilizzati per raccogliere dati di accelerazione per la classificazione del movimento umano come descritto nel capitolo 2. Le caratteristiche di tali moduli sono presenti nella tabella 2.1.

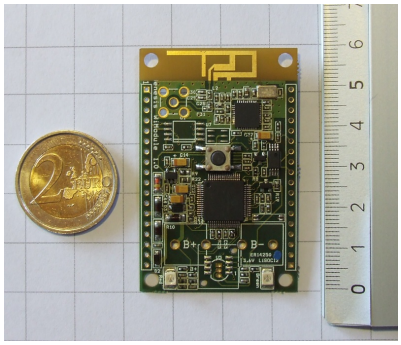


Figura 3.2: Modulo wireless per WSN

In particolare questi moduli utilizzano un trasmettitore compatibile IEEE 802.15.4 a 2.4 Ghz ed un processore della famiglia Microchip PIC18F a 8bit a 40Mhz. Tali moduli sono stati realizzati cercando di ottenere i minori consumi possibili.

3.2 Protocollo di comunicazione

Il protocollo è finalizzato alla soluzione di alcune delle più importanti problematiche tipiche delle reti di monitoraggio wireless (WSN). In particolare è stato concepito per consentire:

- Raccolta e concentrazione di dati verso la rete internet
- Numero elevato di nodi sensori connessi alla rete (Potenzialmente infinito)
- Copertura di aree superiori alla portata radio dei singoli nodi senza utilizzo di nodi ripetitori dedicati e mantenendo una lunga durata dei nodi con alimentazione a batteria (>3anni)

[1] Wireless Sensor Network Platform: http://wisnp.henesis.eu

- Bassissimo costo dei nodi (<100$)

Il particolare l'ultimo punto porta alla scelta di utilizzare la tecnologia RF attualmente a minor costo ovvero i transceiver 2.4GHz che implementano il livello fisico e parte del livello MAC del protocollo IEEE802.15.4 unitamente ad economiche CPU a 8bit dotate quindi di ridotta capacità sia computazionale che di memorizzazione.

Questo protocollo è stato ideato all'interno di Henesis s.r.l. in stretta collaborazione con l'Ing. Chiesi Lorenzo, il quale ha successivamente realizzato l'implementazione sulla piattaforma hardware, che è in corso di brevettazione.

La rete di sensori si compone di:

- **Uno o più Nodi NetworkSink** che si propongono come destinazione finale dei dati raccolti da tutti i nodi della rete. Tali nodi sono necessariamente dotati di una connessione verso internet (Via LAN, GPRS, altri collegamenti) che consente poi il trasferimento dei dati verso un server remoto che provvede al ragruppamento dei dati appartenenti alla stessa rete anche se trasmessi da NetworkSink differenti, alla memorizzazione ed all'elaborazione degli stessi.
- **Un numero arbitrario di nodi sensori** in grado di connettersi e autoorganizzarsi per inviare pacchetti dati contenenti i risultati delle proprie misurazioni verso un nodo di tipo NetworkSink che li trasferirà sul server. I nodi sensori a loro volta possono essere differenziati in:
  - Nodi con capacità di routing (standard) ovvero in grado di accettare ed inoltrare verso un NetworkSink i dati provenienti da altri nodi per estendere la copertura della rete senza bisogno di nodi ripetitori dedicati.
  - **Nodi volatili** (speciali) ovvero nodi che trasmettono i propri dati ad altri nodi della rete senza entrare a farne parte non eseguendo routing. In particolare i nodi mobili che possono venire spostati velocemente all'interno della rete è conveniente siano di questo tipo.

La topologia della rete è quindi di tipo "Dynamic Tree" ovvero i nodi con capacità di routing si organizzano in alberi ognuno dei quali ha alla radice un un NetworkSink

come mostrato in figura 3.3 Le connessioni tra i nodi possono modificarsi nel tempo per sopperire alla scomparsa di nodi, ripristinare connessioni interrotte o ottimizzare le connessioni esistenti.

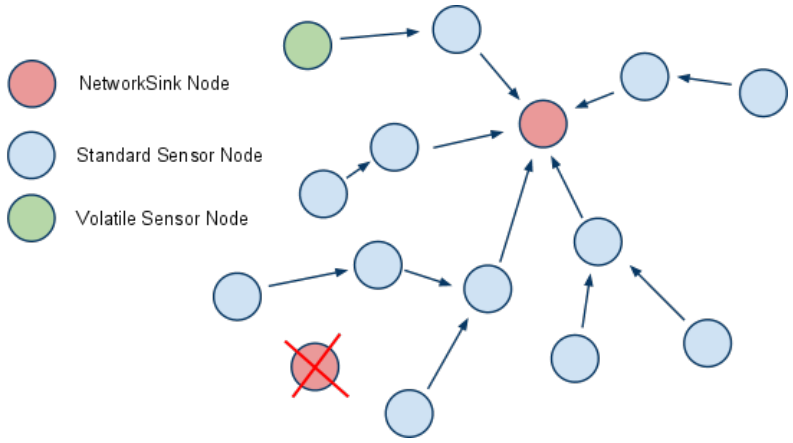


Figura 3.3: Topologia della rete

Il protocollo può essere definito di tipo **"Distribuited Beaconed"** poichè ogni nodo sensore in grado di inoltrare dati nella direzione di un nodo *NetworkSink* invia ciclicamente un pacchetto di beacon con periodicità T_{Beacon} (tipicamente 8 secondi) mediante il quale si rende visibile ai nodi vicini che possono, entro un breve intervallo di tempo, inoltrargli i loro pacchetti dati che il nodo inoltrerà a sua volta in direzione del *NetworkSink*. Quest'approccio consente di mantenere la CPU dei nodi e il transceiver, che rappresenta il maggiore consumo energetico, disattivati per gran parte del tempo realizzando il requisito di lunga durata delle batterie.

La rete così formata presenta una capacità di comunicazione fortemente asimmetrica, orientata prevalentemente alla raccolta dati dai nodi sensori e non all'attuazione. I pacchetti infatti viaggiano dal nodo che li genera al *NetworkSink* attraversando tutti i livelli intermedi in un tempo dell'ordine di $T_{Beacon} * Hop2Sink$. La comunicazione dal *NetworkSink* ai nodi figli è invece possibile solo mediante un approccio di tipo *broadcast* basato sulla propagazione di un messaggio inizialmente inserito nel pacchetto

di beacon dal *NetworkSink* che ogni nodo figlio replica a sua volta nel proprio beacon appena ne viene a conoscenza, ovvero in occasione di una comunicazione con il proprio padre dovuta alla necessità di inviare un proprio dati. Ipotizzando quindi che ogni nodo nella rete trasmetta dati in modo ciclico ogni T_{Data} (ragionevolmente qualche decina di minuti) il tempo di propagazione del messaggio da un *NetworkSink* ai nodi figli sarà dell'ordine di $T_{Data} * Hop2Sink$ ovvero piuttosto elevato. Questo meccanismo di comunicazione non rende possibile utilizzare la rete per implementare sistemi domotici che necessitino di eseguire attuazione in tempi rapidi ma consente al livello applicativo di trasmettere a tutti i nodi della rete comandi di configurazione.

3.2.1 Livello MAC

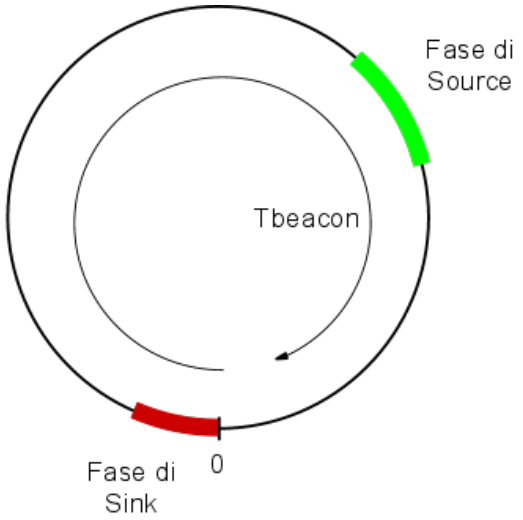
Il protocollo utilizza interamente il livello fisico del protocollo IEEE 802.15.4 e alcune caratteristiche del livello MAC, consentendo così l'utilizzo di transceiver a basso costo a 2.4GHz ormai largamente presenti sul mercato che implementano in hardware la maggior parte delle operazioni necessarie riducendo la complessita del software del microcontrollore. In particolare:

- Vengono utilizzati i meccanismi di accesso al canale CSMA-CA
- Viene rispettata la formattazione dei pacchetti prevista dallo standard (Indirizzamento, Sequence number, CRC)
- Vengono utilizzati i meccanismi di filtraggio dei pacchetti ricevuti sulla base dell'indirizzo di destinazione
- Vengono utilizzati i meccanismi di acknowledging e di ritrasmissione automatica

L'estensione del livello MAC introdotta dal protocollo realizza un approccio di tipo *beaconed distribuito* ed è volta a limitare il più possibile i periodi in cui i nodi mantengono il transreceiver, che rappresenta il consumo maggiore, attivo in ricezione o in trasmissione. Il protocollo infatti prevede che durante il normale funzionamento sia attivato ciclicamente solo durante due brevissime fasi denominate:

- **Fase di Sink** (Fase in cui il nodo osservato emette un pacchetto di beacon rendendosi visibile agli altri e può ricevere dati dai nodi figli)
- **Fase di Source** (Fase in cui il nodo osservato inoltra i dati al padre)

controllate mediante un timer con periodo T_{Beacon} di 2, 4, 8 o 16 secondi regolato da un oscillatore quarzato a basso consumo che può essere invece mantenuto sempre attivo. In figura 3.4 è mostrato uno schema di questo ciclo di base ed in figura 3.5 come i nodi comunichino tra loro senza un tempo di ciclo unico in tutta la rete.



SCHEMATIZZAZIONE DEL CICLO DI SINK E SOURCE
ESEGUITO DAI NODI CON CAPACITA' DI ROUTING

Figura 3.4:

Fase di Sink

La fase di Sink viene **eseguita da tutti i nodi** con capacità di routing **ad ogni ciclo** e **sempre nello stesso istante**, ovvero in corrispondenza dell'overflow della propria base dei tempi e prevede nell'ordine:

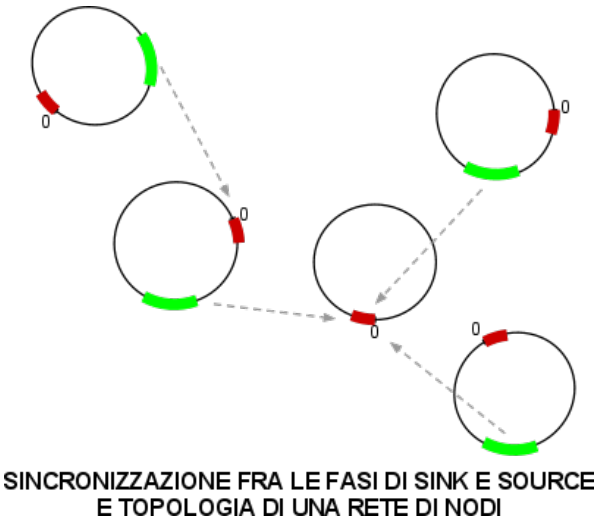


Figura 3.5:

1. Trasmissione in broadcast, dunque senza richiesta di acknowledging, di un pacchetto denominato beacon che contiene informazioni sullo stato del nodo stesso. Tale trasmissione avviene istantaneamente, senza meccanismo di CSMA-CA, per ottenere un elevata precisione del periodo di invio del beacon necessaria al funzionamento del protocollo.

2. Intervallo di ricezione di durata massima $T_{WaitData}$ durante il quale possono essere ricevuti pacchetti dati inviati dai nodi figli che eseguono la fase di Source. In questa fase il nodo padre accetta solo pacchetti a lui destinati operando un filtraggio sulla base dell'indirizzo di destinazione. Ogni pacchetto valido ricevuto viene immagazzinato in un'area di memoria denominata RoutingBuffer e provoca la spedizione di un pacchetto di acknowledge al figlio e l'estensione dell'intervallo di ricezione di un ulteriore $T_{WaitData}$ per permettere ad altri nodi figli in competizione di trasmettere i propri dati. In caso di saturazione del RoutingBuffer il pacchetto di acknowledge non verrà inviato e la fase di Sink terminerà immediatamente.

$T_{WaitData}$deve essere correttamente dimensionato per permettere la ricezione di un pacchetto dati con lunghezza massima (128 byte @ 256 kbit/s) ovvero deve essere maggiore di 4ms.

In figura 3.6 è presente uno schema di tale fase e in figura 3.7 l'*header* del pacchetto beacon inviato.

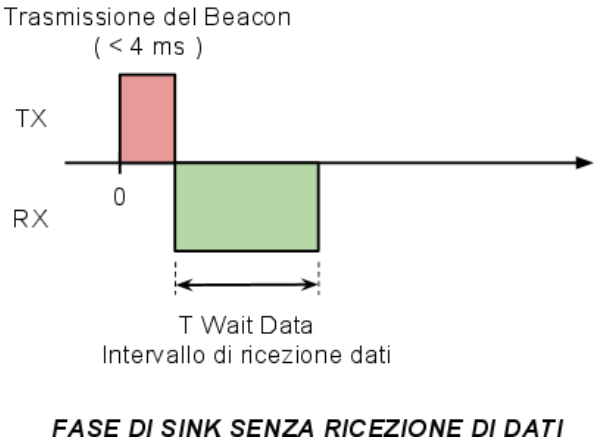


Figura 3.6:

Fase di Sync/Source

La fase di Source viene eseguita dai nodi figli solo nel momento in cui si rende necessario inoltrare al proprio nodo padre un pacchetto dati generato internamente o dei pacchetti ricevuti dai propri nodi figli e temporaneamente immagazzinati nel RoutingBuffer. Condizione necessaria per l'esecuzione della fase di Source è la conoscenza dell'istante temporale, riferito alla propria base dei tempi, in cui il nodo padre emetterà il proprio pacchetto di beacon e successivamente attiverà il proprio transceiver in ricezione garantendo così una finestra temporale per la comunicazione verso di lui.

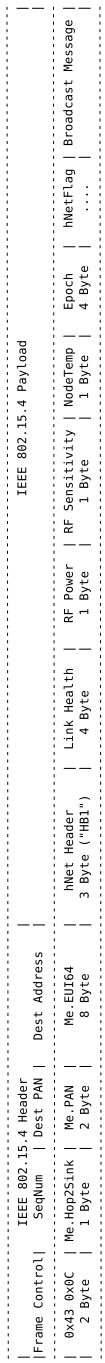


Figura 3.7: Composizione del pacchetto *beacon*

La prima parte della fase di Source, definita **fase di Sync**, consiste per un determinato nodo nell'attivare il proprio transceiver del nodo in ricezione per un tempo massimo di $T_{WaitBeacon}$ durante il quale deve essere ricevuto il pacchetto di beacon del nodo padre garantendo la corretta sincronizzazione con la sua fase di Sink. In questa possono essere convenientemente utilizzati i meccanismi di filtraggio dei pacchetti sulla base dell'indirizzo di destinazione implementati in hardware nel transceiver poichè nel campo destinazione del pacchetto di beacon viene riportato l'indirizzo del nodo padre stesso.

Il transceiver deve essere attivato con un anticipo $T_{Advance} = T_{TransceiverWake} + T_{Drift}$ rispetto all'istante T_{EBS} (Estimated Beacon Start Time) in cui si stima che il nodo padre inizierà la trasmissione del pacchetto di beacon; tale anticipo deve essere sufficiente per garantire il corretto risveglio del transceiver e la ricezione dell'intero pacchetto di beacon del padre anche in presenza di un certo drift tra le basi dei tempi. L'intervallo di ricezione può quindi essere correttamente dimensionato considerando che deve essere sufficientemente ampio per garantire la ricezione di un pacchetto di beacon di lunghezza massima (128 byte @ 256 kbit/s) e deve consentire la presenza del drift considerato in entrambe le direzioni: $T_{WaitBeacon} = T_{Advance} + 4ms + T_{Drift} = 4ms + T_{TransceiverWake} + 2 * T_{Drift}$.

Figura 3.8 mostra un caso di ricezione terminata con successo.
Se al termine dell'intervallo di ricezione non è stato ricevuto il pacchetto di beacon atteso la fase di Sync termina immediatamente riportando un errore di tipo Missed-Beacon, come mostrato in figura 3.9.

La fase di Sync che termina correttamente assolve quindi ai seguenti compiti:

- Verifica della presenza del nodo padre
- Campionamento delle informazioni contenute nel beacon del nodo padre e conseguente aggiornamento di quelle trasmesse dal nodo nel proprio beacon:
  - $Self.Hop2Sink = Parent.Hop2Sink + 1$
  - $Self.LinkHealth = Parent.LinkHealth * Self.NodeHealth (Self.NodeHealth < 1 \geq Self.LinkHealth < Parent.LinkHealth)$

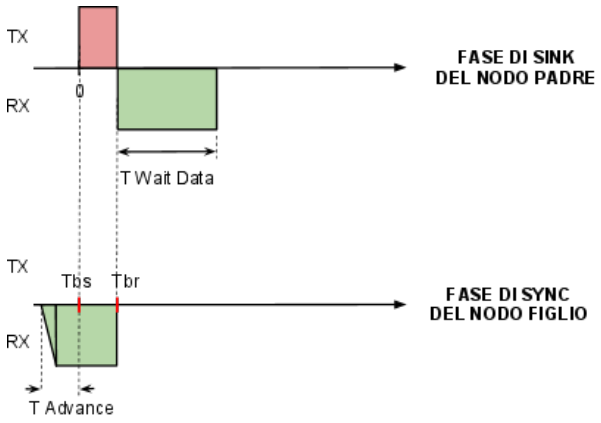


Figura 3.8:

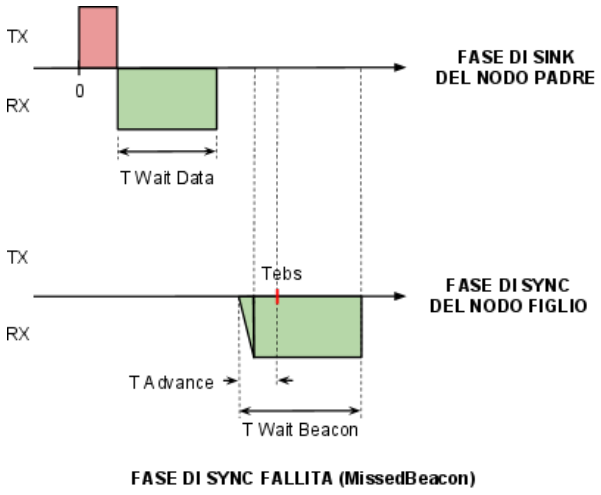


Figura 3.9:

- $Self.DA\_Flag = Parent.DA\_Flag$
- $Self.Epoch = Parent.Epoch$

– $Self.Broadcast\_Message = Parent.Broadcast\_Message$

- Re-sincronizzazione della stima dell'istante di inizio della trasmissione del beacon del nodo padre: $T_{EBS} = T_{BS}$. Poichè il pacchetto di beacon può avere lunghezza variabile, in particolare a causa della propagazione di messaggi in broadcast da parte del NetworkSink, è necessario risalire all'istante di inizio della trasmissione del pacchetto di beacon, che è periodico. Questo è possibile considerando l'istante T_{BR} (Beacon Reception Time), determinato campionando la base dei tempi locale nell'istante in cui il transceiver comunica la ricezione del pacchetto, unitamente alla lunghezza del pacchetto stesso e alla velocità di trasmissione $v_T = 250 kbit/s$ secondo la formula: $T_{BS} = T_{BR} - L_B/v_T$.

La fase di Sync può essere attivata dai livelli protocollari superiori anche in assenza di pacchetti da trasferire al nodo padre, in modo indipendente quindi dalla fase di Source, con lo scopo di forzare la sincronizzazione con il nodo padre e assicurare la propagazione e l'aggiornamento delle informazioni contenute nei pacchetti di beacon relative allo stato della connessione con il Network Sink.
Nel caso il nodo abbia internamente generato un pacchetto dati o ne abbia ricevuti dai nodi figli in una fase di Sink precedente la fase di Sync prosegue dando luogo alla **fase di Source** vera e propria.

Avendo appena eseguito con successo la fase di Sync infatti ci si trova nell'intervallo in cui il nodo padre sta mantenendo attivo in ricezione il proprio transceiver ed è quindi possibile iniziare la trasmissione dei pacchetti dati disponibili dando priorità a quelli generati internamente. La trasmissione di ogni pacchetto avviene utilizzando il meccanismo di CSMA-CA e di ritrasmissione automatica in seguito a mancata ricezione del pacchetto di acknowledge del nodo padre (fino a tre volte consecutive come definito dallo standard IEEE 802.15.4). I pacchetti correttamente inviati vengono quindi eliminati dalla memoria mentre in caso di fallimento della procedura di trasmissione la fase di Source terminerà immediatamente riportando un errore di tipo MissedAck e i pacchetti saranno mantenuti nei buffer provocando l'attivazione della fase di Source successiva.

In figura 3.10 è presente uno schema di tale fase e in figura 3.11 l'*header* del pacchetto dati inviato.

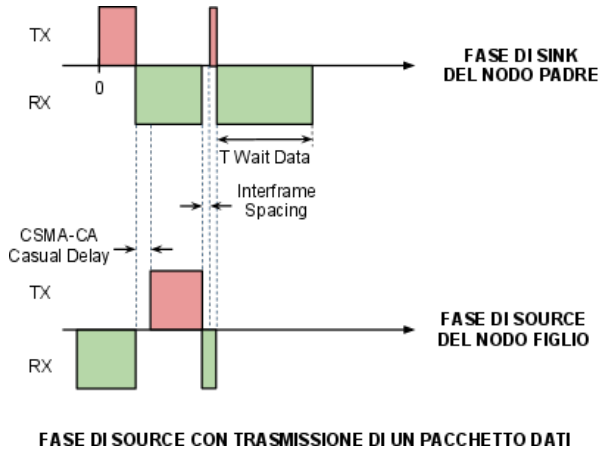


Figura 3.10:

Fase di Scan

La fase di Scan può essere eseguita in qualunque momento da ogni nodo che debba determinare la presenza di vicini potenzialmente in grado di trasportare i pacchetti dati fino al NetworkSink. In questa fase il transceiver viene attivato in ricezione per un tempo T_{Beacon} durante il quale vengono ricevuti esclusivamente pacchetti di tipo beacon inviati dai vicini a portata radio (Promiscuous Mode).
Per ogni pacchetto ricevuto viene campionato il valore assunto dalla base dei tempi locale determinando T_{BR}, risalendo poi all'istante di inizio della spedizione del beacon T_{BS} analogamente a quanto illustrato per la fase di Sync. I pacchetti ricevuti devono quindi superare una serie di filtraggi volti a verificare l'affidabilità della connessione bidirezionale e la possibilità di raggiungere il Network Sink attraverso ogni nodo di cui è stato ricevuto il beacon.

Figura 3.11: Composizione del pacchetto *data*

Filtraggio dei nodi non affidabili

Per ogni pacchetto beacon ricevuto, corrispondente ad un determinato vicino, il transceiver riporta un valore numerico detto RSSI proporzionale alla potenza in dB del segnale ricevuto, indice della qualità della connessione dal nodo remoto a se stesso, figura 3.12. Utilizzando le informazioni sulla potenza di trasmissione e sulla sensibilità del nodo remoto, contenute nel pacchetto di beacon stesso, unitamente alle proprie il nodo locale può quindi stimare il valore di RSSI della connessione nel verso opposto (da se stesso al nodo remoto): $RSSI_R = P_S - P_R + S_S - S_R + RSSI_S$ con S: Self e R: Remote

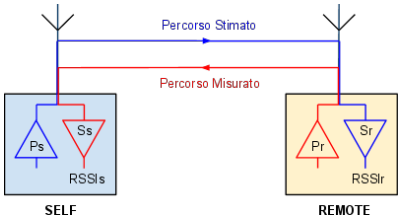


Figura 3.12:

La qualità della connessione bidirezionale può quindi essere assunta come proporzionale al minimo dei due valori: $RSSI_{Min} = min(RSSI_S, RSSI_R)$. Vengono quindi scartati tutti i nodi ricevuti per cui $RSSI_{Min}$ è al di sotto del valore di soglia $RSSI_{ReliabilityTreshold}$ che assicura la stabilità della comunicazione bidirezionale.

Filtraggio dei nodi con il flag DA attivato

I nodi che mostrano nel beacon il flag DA (Don't Attach) attivato vengono filtrati poiché tale flag indica che un nodo del ramo ha manifestato problemi di connessione e quindi il ramo stesso al momento non assicura una strada affidabile per la trasmissione dei dati verso il NetworkSink.

Le seguenti informazioni relative ai nodi che hanno superato i filtraggi precedenti vengono quindi memorizzate in una tabella:

- T_{BS}
- *EUI64*
- *Hop2Sink*
- *Health*
- *NodeTemp*
- $RSSI_{Min}$

Poichè la memoria del sistema è estremamente limitata e i nodi a portata radio possono essere molto numerosi la tabella ha necessariamente un numero di elementi limitato (fino a qualche decina di nodi) e viene quindi mantenuta ordinata in funzione del parametro Hop2Sink mostrato dai nodi nel beacon, privilegiando quelli con valori minori.Tale metrica di ordinamento è sub-ottima poiché non tiene conto dell'affollamento dei nodi, dello stato delle batterie e della qualità del collegamento radio ma ha il pregio di essere basata su numeri interi in formato "unsigned char" che assicurano un calcolo rapido del punto di inserimento in tabella anche con CPU ad 8 bit. La rapidità di queste operazioni è necessaria per assicurare la ricezione e la memorizzazione delle informazioni di tutti i beacon che possono giungere al transceiver in modo anche molto ravvicinato rendendo la fase di Scan potenzialmente time-critical.

Al termine della fase di Scan i livelli protocollari superiori potranno agire sulle informazioni memorizzate nella tabella per stabilire secondo una metrica maggiormente ottimizzata ma più complessa il vicino migliore che verrà scelto come padre.

Inseguimento della deriva del nodo padre

Ogni nodo è dotato di una base dei tempi indipendente incrementata da un oscillatore locale quarzato la cui frequenza è affetta da un errore tipicamente quantificato dai

costruttori in $\delta = \pm 20 ppm @ 25°C$ e dipendente dalla temperatura come mostrato in figura 3.13.

Lo scostamento dalla frequenza nominale è inoltre dipendente dalla temperatura secondo la relazione:

$\frac{\Delta f(T)}{f_0} = 0.04 ppm \cdot (T - T_0)^2$

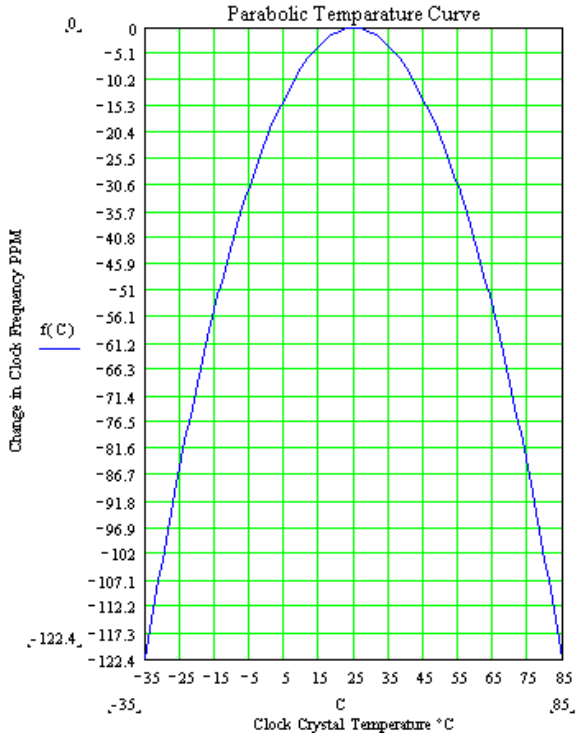


Figura 3.13:

Consideriamo un nodo S posto in ambiente esterno durante il periodo invernale (-20°C) che trasmette ad un nodo padre P posto in un ambiente riscaldato in prossimità di apparecchiature elettriche che sviluppano calore (+50 °C).

$\frac{\Delta f_S}{f_0} = 0.04 ppm \cdot (50°C - 25°C)^2 = 25 ppm$

$\frac{\Delta f_P}{f_0} = 0.04 ppm \cdot (-20°C - 25°C)^2 = 81 ppm$

Tenendo conto della componente di errore aleatoria e dell'influenza della temperatura è possibile stimare la differenza di frequenza tra l'oscillatore del nodo considerato e quello del nodo padre mediante la relazione:

$\frac{\Delta f_{SP}}{f_0} = (\frac{\Delta f_S}{f_0} + |\delta|) + (\frac{\Delta f_P}{f_0} + |\delta|) = (25+20) + (81+20) = 146ppm$

Considerando che il nodo osservato trasmetta dati saltuariamente appare evidente come la differenza di frequenza tra gli oscillatori possa portare in breve tempo alla perdita della sincronizzazione tra le fasi di Source del nodo osservato e quella di Sink del nodo padre come mostrato in figura 3.14.

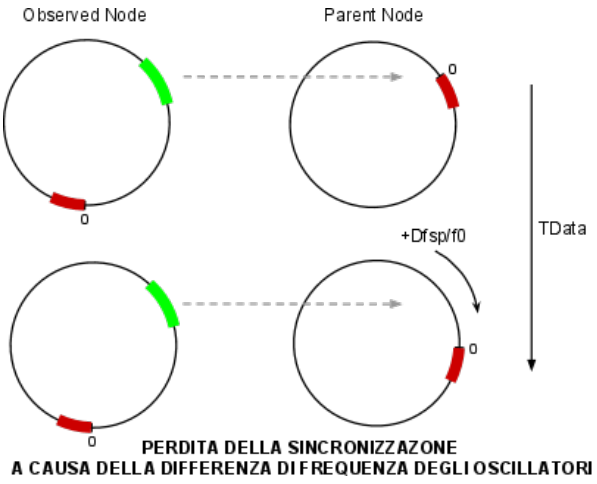


Figura 3.14:

In particolare considerando la trasmissione di un dato ogni tempo $T_{Data} = 15min$ nelle condizioni dell'esempio si accumulerebbe una deriva esprimibile con la relazione $T_{Drift} = T_{Data} \cdot \frac{\Delta f_P}{f_0} = 131ms$ che, per le relazioni illustrate precedentemente, obbligherebbe a mantenere attivo il transceiver in ricezione durante la fase di Sync per un tempo eccessivamente lungo stimabile in $T_{WaitBeacon} \simeq 2 \cdot T_{Drift} \simeq 260ms$ con grande consumo di energia.

È quindi necessario implementare un meccanismo di stima della deriva e inseguimento dell'istante di trasmissione del beacon del nodo padre che può essere

concettualmente descritto in questo modo:

Inizializzazione delle variabili eseguita dal livello protocollare Network successivamente all'esecuzione di una fase di Scan e alla selezione di un nuovo nodo padre:

- $EstimatedDrift = 0.04 * (Ts - Tp)^2$
  (La stima della deriva congiunta fra le basi dei tempi viene inizializzata sulla base della temperatura misurata al nodo locale e di quella misurata dal nodo padre trasmessa nel pacchetto di beacon secondo la relazione mostrata precedentemente)
- $OldBeaconStartTime = ActualBeaconStartTime$
- $OldEpoch = ActualEpoch$

La fase di Sync/Source verrà quindi programmata in corrispondenza dell'istante stimato di inizio della trasmissione del beacon da parte del padre, ricavato dal valore calcolato durante la precedente fase di Sync aggiungendo un termine dipendente dal tempo trascorso e dalla deriva stimata:

$EstimatedBeaconStartTime = OldBeaconStartTime + (ActualEpoch - OldEpoch) * EstimatedDrift$

Al termine della fase di Sync verrà quindi aggiornata la stima della deriva congiunta tra le basi dei tempi considerando la differenza tra l'istante di inizio della trasmissione del beacon appena rilevato e quello memorizzato durante la fase di Sync precedente, unitamente all'informazione sul tempo trascorso tra le due fasi di Sync stesse:

- $EstimatedDrift = (ActualBeaconStartTime - OldBeaconStartTime)/(ActualEpoch - OldEpoch)$
- $OldBeaconStartTime = ActualBeaconStartTime$
- $OldEpoch = ActualEpoch$

In questo modo sarà possibile ridurre al minimo l'intervallo T_{Drift} e conseguentemente $T_{WaitBeacon}$ riducendo considerevolmente il consumo di energia durante la fase di Sync/Source.

Conflitto fra fase di Sink e fase di Source

Il meccanismo di inseguimento della fase di Sink del nodo padre porterà in un certo tempo all'avvicinamento e alla conseguente interferenza fra la fase di Sink e Source del nodo osservato. Considerando un drift di 150 ppm e T_{Beacon} = 8 sec si determina che questo avviene nel giro di qualche giorno.

Ogni nodo implementa quindi un meccanismo di controllo che verifica la separazione tra la propria fase di Sink e la fase di Source verificando che quest'ultima venga programmata al di fuori dell'intervallo di sicurezza $T_{Clearence}$ posto attorno alla fase di Sink, figura 3.16. Qualora si verifichi un conflitto tra le due fasi il nodo provvede immediatamente a traslare la propria fase di Sink di $T_{Beacon}/2$ ripristinando la separazione e assicurandola per un ulteriore periodo dell'ordine di qualche giorno, come mostrato in figura 3.15.

I nodi figli del nodo che ha compiuto questa procedura si accorgeranno dello spostamento della fase di Sink del padre durante la loro successiva fase di Sync/Source che terminerà riportando l'errore di tipo MissedBeacon. Prima di attivare la procedura di ripristino della connessione (che richiede l'esecuzione di una fase di Scan energeticamente molto costosa) il livello Network deve verificare il possibile spostamento della fase di Sink del padre traslando la propria fase di Source di $T_{Beacon}/2$. Nel caso venga riportato ulteriormente un errore di tipo MissedBeacon può quindi essere iniziata la procedura di ripristino della connessione.

3.2.2 Livello NETWORK (Standard Node)

Il livello *Network* sfrutta le funzionalità del livello *MAC* per garantire la corretta formazione della rete, la continua ottimizzazione e il ripristino in caso di errori di comunicazione.

Di seguito vengono descritte i vari stati e le fasi della figura 3.17.

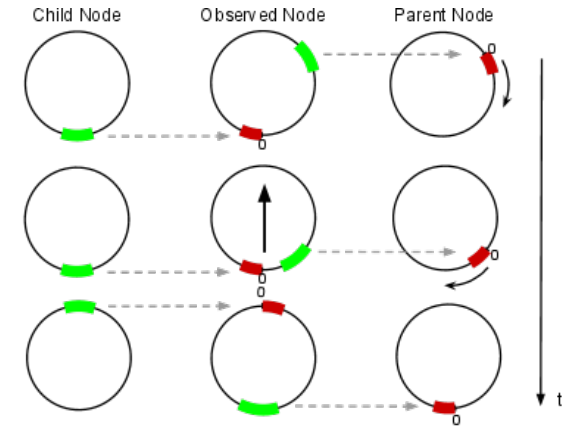


Figura 3.15:

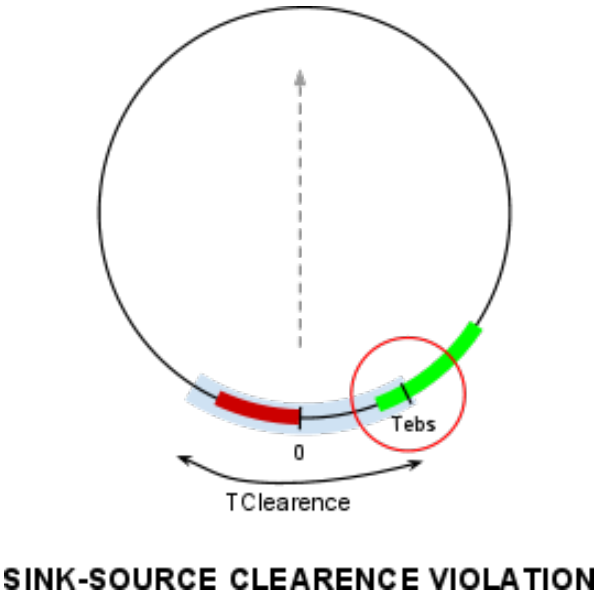


Figura 3.16:

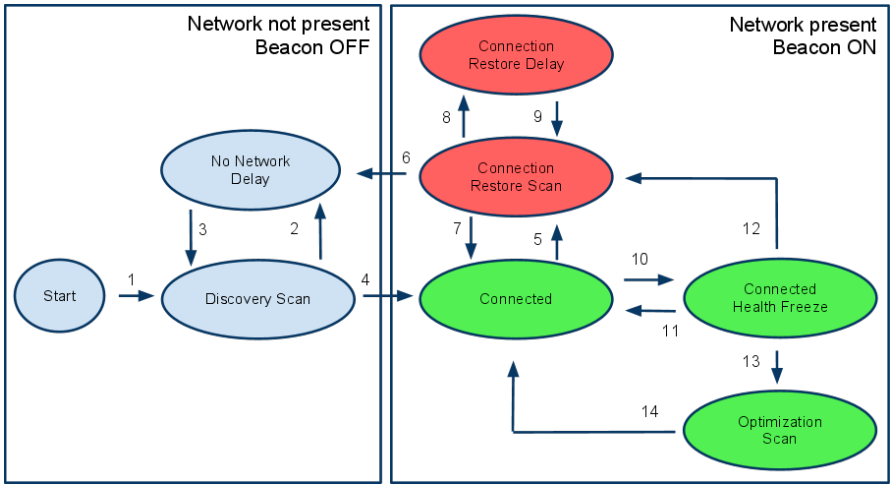


Figura 3.17: Livello di rete

(1) Alla prima accensione ogni nodo con capacità di routing passa dallo stato *Start,* in cui non appartiene ad alcuna rete e contiene tutti i parametri al valore di default, allo stato *DiscoveryScan.* In particolare T_{Wait} viene inizializzato al valore $T_{WaitStart}$.

(DiscoveryScan) Il nodo esegue una fase di Scan acquisendo così la tabella dei nodi in grado di raggiungere il NetworkSink presenti nel vicinato.

(2)Al termine delle fase di Scan se la tabella dei vicini restituita è vuota si passa allo stato *NoNetworkDelay.*

(NoNetworkDelay)Viene eseguita una attesa pari ad un tempo T_{wait}. Durante questa attesa il trasmettitore wireless è spento ed il processore è in modalità di sleep a bassissimo consumo.

(3) Al termine dell'attesa viene posto $T_{Wait} = T_{Wait} * k$ e si ritorna allo stato *Discove-*

ryScan. Il valore di T_{Wait} viene in ogni caso saturato al valore $T_{WaitSAT}$ realisticamente dell'ordine di qualche decina di ore.

(4) Al termine della fase di Scan, nel caso siano stati rilevati potenziali vicini in grado di veicolare dati in direzione del NetworkSink il nodo procede alla determinazione del migliore secondo una metrica ottimizzata che tiene conto di molteplici fattori.

Viene calcolato infatti il valore
$Self.LinkHealth = Self.NodeHealth * Neighbour.LinkHealth$
dove
$Self.NodeHealth = f(Parent.Hop2Sink, Parent.RSSimin, Self.Battery)$
considerando di volta in volta ogni vicino le cui informazioni sono memorizzate nella tabella come padre. Viene quindi scelto come padre il nodo per cui si ottiene il valore $Self.LinkHealth$ più elevato.

Determinato il nodo padre viene forzata la spedizione di un pacchetto dati vuoto, denominato *PresencePacket*, che ha lo scopo di far raggiungere al NetworkSink l'informazione di presenza del nuovo nodo connesso e consente, mediante l'esecuzione di una fase di Sync di acquisire il beacon completo dal nodo padre da cui il nodo deriverà il proprio propagando le informazioni sulla qualità di connessione del ramo e il messaggio propagato in broadcast dal NetworkSink.

Il nodo passa quindi allo stato *Connected iniziando ad eseguire ciclicamente* la fase di Sink durante la quale trasmetterà il proprio beacon.

(Connected) Durante questo stato sono attive sia la fase di Sink che quella di Source quindi il nodo effettua il routing dei pacchetti inviati dagli altri nodi e i livelli superiori possono generare e inviare propri pacchetti dati. Per assicurare una minima velocità di propagazione dei messaggi in broadcast dal NetworkSink ai nodi figli se non vengono inviati pacchetti per un tempo superiorie a T_Resync il livello di rete forzerà un fase di Sync. Il livello di rete in questo stato si occupa inoltre di mantenere costantemente aggiornato il valore Self.NodeHealth propagato nel beacon.

(5) Durante la spedizione di un dato è possibile che la *Fase di Sync/Source* falli-

sca un numero consecutivo di volte superiore a $Fail_{threshold}$ perciò il sistema entra in uno stato di risoluzione del problema denominato *ConnectionRestoreScan*.

Viene quindi salvato l'attuale $Self.Hop2Sink$ nella variabile Hop2SinkSafe, attivato il flag DA (Don't Attach) nel beacon e iniziata una nuova fase di Scan. Tutti i nodi vicini attivi rilevati al termine della scansione con $Neighbour.Hop2Sink < Hop2SinkSafe$ potranno essere considerati per il ripristino della connessione verso il NetworkSink poichè non possono essere figli del nodo stesso.

(ConnectionRestoreScan) Stato durante il quale viene eseguita la fase di scansione per la ricerca di vie alternative per inoltrare messaggi al NetworkSink.

(6) Se al termine della scansione la lista dei nodi vicini risulta vuota significa che non c'è nessun nodo a cui è possibile connettersi nelle vicinanze, perciò il nodo entra nella fase *No Network Delay* e ripristina allo stato iniziale il suo stato interno disattivando in particolare la fase di Sink e la fase di Source.

(7) Se al termine della scansione nella lista è presente un sottoinsieme di nodi che rispettano la condizione Neighbour.Hop2Sink $<$ Hop2SinkSafe verrà applicato limitatamente ad essi l'algoritmo di determinazione del migliore illustrato al punto 4. Tale nodo verrà quindi designato come nuovo padre e gli sarà immediatamente inviato un presence packet propagando il nuovo beacon, inoltre verrà disattivato il flag DA nel beacon poichè la connessione è stata ripristinata.

(8) Se al termine della scansione sono presenti dei nodi nella lista ma nessuno rispetta la condizione $Neighbour.Hop2Sink < Hop2SinkSafe$ sarà necessario attendere un tempo T_{Resync} al termine del quale il limite $Hop2SinkSafe$ potrà essere incrementato di un'unità. L'attesa infatti garantisce che i nodi figli siano raggiunti dal flag DA che si propaga attraverso il beacon e quindi vengano esclusi dalla tabella dei vicini evitando la formazione di loop.

(ConnectionRestoreDelay) Questa fase ha durata T_{Resync} durante la quale il nodo

continua ad eseguire la fase di Sink inviando il proprio beacon con il flag DA attivato. Il nodo non può eseguire la fase di Source poichè non è sincronizzato con alcun parent e quindi scarta i dati inoltrati dagli altri nodi per non saturare il RoutingBuffer e continuare a trasmettere l'acknowledge mantenendo il proprio ramo attivo.

(9) Al termine della fase di ConnectionRestoreDelay viene aggiornato $Hop2SinkSafe = Hop2SinkSafe + 1$ e si ritorna nello stato di ConnectionRestoreScan nel quale viene eseguita una nuova scansione dei nodi presenti nel vicinato.

(10) Dopo un tempo $T_{Optimize}$ in cui un nodo si trova nello stato Connected viene avviato un processo di ottimizzazione che porta alla rivalutazione del nodo padre. L'intervallo di ottimizzazione è dell'ordine della decina di ore ed è in parte ottenuto in modo casuale per evitare che tutti i nodi eseguano l'ottimizzazione contemporaneamente.

(ConnectedHealthFreeze) Questo stato transitorio replica esattamente il comportamento dello stato Connected con la sola differenza di mantenere invariato il valore Self.LinkHealth che il nodo trasmette nel beacon.

In questo modo dopo un tempo $N_{layer\_back} * T_{Resync}$ i nodi figli dei successivi N_{layer_back} livelli mostreranno nel beacon un parametro LinkHealth strettamente inferiore a quello del nodo stesso, consentendo di escluderli facilmente dalla ricerca di un nuovo nodo padre ed evitando così la possibilità di creare loop nella topologia della rete.

(11) Se durante lo stato ConnectedHealthFreeze il valore di Hop2Sink del nodo cambia in relazione ad un cambiamento della topologia della rete avvenuto nei livelli inferiori la fase di ottimizzazione viene annullata e riprogrammata successivamente.

(12) Se durante lo stato ConnectedHealthFreeze viene superato il numero di errori consentiti alla fase di Sync/Source la fase di ottimizzazione viene annullata e riprogrammata successivamente e si ha un comportamento analogo a quello descritto nel

punto 5.

(13) Una volta atteso l'intervallo di tempo $N_{layer\_back} * T_{Resync}$ viene iniziata una fase di Scan e si passa allo stato OptimizationScan.

(OptimizationScan) In questo stato si attende il termine della fase di Scan.

(14) Al termine della fase di Scan vengono presi in considerazione per l'ottimizzazione il sottoinsieme per cui $Neighbour.Hop2Sink \leq Self.Hop2Sink + N_{layer\_back}$ (In questo sottoinsieme possono essere comunque presenti dei nodi figli ma avranno certamente $Child.LinkHealth < Self.LinkHealth$). Verrà quindi applicato l'algoritmo di determinazione del migliore illustrato al punto 4 e solo in caso tale nodo risulti diverso e migliore dal nodo padre attuale si provvederà a sostituirlo trasmettendo un presence packet al nuovo nodo padre e memorizzandone il beacon.

In ogni caso si torna allo stato di Connected.

3.2.3 Livello Applicativo (Nodo Standard)

Tale livello utilizza i servizi offerti dai livelli sottostanti per svolgere un determinato compito che dipende dall'applicazione implementata. L'esempio più rilevante è quello di un nodo sensore per il monitoraggio distribuito.

Il livello applicativo agisce quando il livello di rete si trova negli stati Connected, ConnectedHealthFreeze, OptimizationScan e compie il seguente ciclo:

1. Attesa che $Epoch > ceil(Epoch/T_{Data}) \cdot T_{Data}$
2. Campionamento dei sensori e generazione di un pacchetto dati
3. Attesa di un tempo casuale diverso tra nodo e nodo e per ogni campionamento compreso tra 0 e T_{Data}.
4. Invio del pacchetto.

Il punto (1) garantisce che tutti i nodi che hanno lo stesso passo di campionamento T_{Data} effettuino il campionamento in corrispondenza della stessa transizione dell'epoch.

Il punto (3) garantisce invece che ogni nodo spedisca il proprio pacchetto in rete in istanti casuali garantendo statisticamente lo sfruttamento di tutte le possibili fasi di Source ed evitando così l'aggregazione di tutte le spedizioni nello stesso ciclo.

3.3 Classificazione dati ambientali

Il protocollo di rete descritto nel precedente capitolo è stato utilizzato per acquisire vari parametri ambientali utilizzando le schede *WiModule* per creare una rete di monitoraggio. A conclusione di questo capitolo riportiamo i risultati di alcuni test preliminari effettuati per valutare la funzionalità del protocollo e le potenzialità offerte dalle HTM relativamente ai dati provenienti dalle reti che il protocollo consente di realizzare.

In particolare sono stati raccolti dati relativi a circa un anno di temperatura ed umidità in vari punti di un ambiente, sia all'esterno che all'interno. Tali dati sono stati analizzati tramite le Hierarchical Temporal Memory descritte nel capitolo 1 per individuare le tipologie di giornata partendo dall'analisi dei dati. L'analisi che verrà presentata di questi dati riguarda lo studio solamente dei dati di temperatura ed umidità esterni; inoltre essa è un punto di partenza per studi più complessi e con un numero maggiore di sensori.

Preparazione del dataset

I dati di temperatura ed umidità in ingresso devono essere temporalmente sequenziali in modo che la HTM sia in grado di estrapolare autonomamente l'informazione temporale delle sequenze di interesse.

I dati trattati sono dati puntuali campionati ogni 5 minuti, perciò è ragionevole pensare che possano essere affetti da un certo rumore. Inoltre parametri ambientali come temperatura ed umidità hanno un tasso di variazione abbastanza lento perciò è

ragionevole applicare un processo di *smoothing* al segnale per filtrare la componente del rumore, come mostrato in figura 3.18.

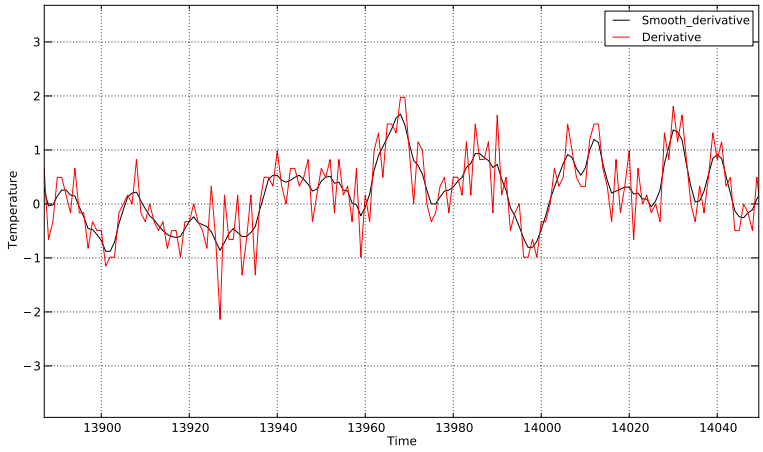


Figura 3.18: Funzione di smoothing

L'algoritmo di smoothing più semplice è quello rettangolare in quanto sostituisce ogni punto del segnale con la media di m punti adiacenti, dove m è un intero positivo chiamato larghezza di smooth. L'algoritmo di smoothing utilizzato, invece, è quello triangolare che, a differenza del primo, implementa una funzione di smoothing ponderata, con m (finestra di smoothing) pari a cinque, questo tipo di smoothing con la finestra di valori scelta è efficace nel ridurre il rumore alle alte frequenze del segnale. L'algoritmo di *smoothing triangolare* equivale ad un doppio smooth rettangolare tra valori contigui nel tempo e la funzione matematica che esprime l'algoritmo è:

$$S = \frac{Y_{j-2}+2Y_{j-1}+3Y_j+2Y_{j+1}+Y_{j+2}}{9}$$

Un esempio degli effetti dello smoothing triangolare è mostrato in figura 3.18.

Durante l'analisi preliminare dei dati si è affrontato il problema della sequenzialità dei rilevamenti poichè una giornata con misurazioni mancanti all'interno della

HTM causa imprecisioni nella formazione delle catene temporali. Per ovviare a tale problema si è proceduto ad interpolare i dati, nel caso di pochi campioni mancanti, oppure ad eliminare l'intera giornata per intervalli più lunghi delle 4 ore.

Definizione delle sequenze

Come discusso nel capitolo 1, nonostante le Hiearchical Temporal Memories prevedano un apprendimento non supervisionato, è possibile evidenziare nel dataset l'inizio e fine di sequenze di dati tramite i *punti di reset*. Tali punti sono utilizzati per non memorizzare delle transizioni temporali all'interno della memoria delle HTM ed evitare che si creino delle memorie contenenti sequenze che attraversano tali punti.

Per analizzare temperatura ed umidità sono stati definiti come *punti di reset* gli istanti di massimo e minimo del valore sulla giornata, come da figura 3.19. In tale maniera la Hiearchical Temporal Memory memorizzerà diverse sequenze corrispondenti a differenti profili di crescita e decrescita di temperatura ed umidità corrispondenti a differenti tipologie di giornate.

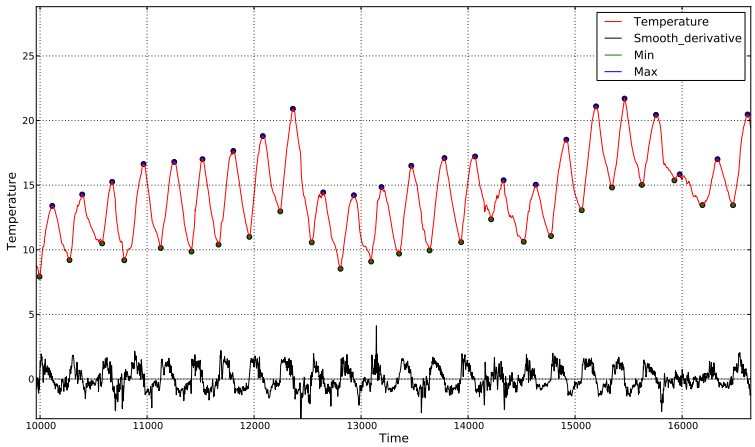


Figura 3.19: Punti di massimo e minimo

Topologia HTM

I canali di temperatura ed umidità sono stati analizzati separatamente con una rete composta da un solo nodo HTM (composto da SpatialPooler e TemporalPooler) come mostrato in figura 3.20.

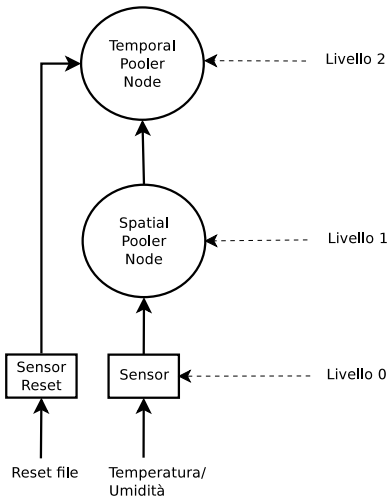


Figura 3.20: HTM per l'analisi di temperatura o umidità

Per quanto riguarda i dati di temperatura lo SpatialPoolerNode è stato addestrato con un file contenente una lista ordinata di valori di temperatura che rappresentano i centri di quantizzazione desiderati, equi-distanziati di 1° C e con un range da -1 a 50 rappresentante i possibili valori di temperatura del dataset. Per quanto riguarda l'umidità, invece, lo SpatialPoolerNode è stato addestrato con un file di valori di umidità che rappresentano i centri di quantizzazione desiderati nel quale i valori hanno un range da 0 a 100 e sono distanziati di un grado percentuale.

Tale tipo di addestramento non sfrutta le caratteristiche di quantizzazione intrinseche alle Hierarchical Temporal Memories. È stato utilizzato questo tipo di preprocessing data la natura nota dei dati in ingresso; resta indiscussa la necessità di una

tale caratteristica per i nodi ad un livello più alto della rete o di quelli di cui non si possono determinare in maniera chiusa le caratteristiche degli ingressi.

In maniera sperimentale, si è fissato a 0,5 il parametro *maxDistance*, che definisce la distanza massima per cui un modello di ingresso è diverso da quello salvato, in modo da non perdere generalità.

Il parametro *sigma* (che specifica la deviazione standard della gaussiana che determina quante coincidenze si attivano in uno stesso momento), è stato settato a 0.5 in quanto i dati in ingresso sono poco rumorosi, essendo già stati sottoposti ad un processo di filtraggio.

Analisi Temperatura

L'obiettivo principale dei test è stato valutare che i gruppi attivi nei fronti di salita e discesa della giornata fossero identificativi del tipo di evento. Per semplicità si è scelto di iniziare addestrando e testando la struttura con i dati relativi ad una settimana primaverile, caratterizzata da un andamento della temperatura abbastanza regolare e da giorni simili tra di loro.

La figura 3.21 mostra l'uscita del sistema testato sui una differente sequenza appartenente alla medesima tipologia di giornata. Nella figura è rappresentato il segnale di ingresso in rosso, in blu l'output dello SpatialPoolerNode e nella parte inferiore l'output del TemporalPoolerNode. Le curve color viola e giallo rappresentano i gruppi temporali costruiti dal TemporalPoolerNode. La prima determina il fronte di salita della giornata mentre la seconda il fronte di discesa. I gruppi creati risultano essere abbastanza precisi e descrittivi in quanto il gruppo viola identifica il solo fronte di salita e il gruppo giallo il solo fronte di discesa.

La figura 3.22 mostra la Time Adjacency Matrix (TAM) "Sorted" relativa all'apprendimento della settimana primaverile. È possibile osservare come la matrice abbia la maggior parte dei valori attorno alla diagonale, ovvero come gli stati del TemporalPooler si susseguano in maniera ordinata, indice di un buon apprendimento.

Per testare ulteriormente le performance di tale rete è stato utilizzato un dataset contenente un mese di rilevamenti caratterizzanti giornate primaverili e il risultato è mostrato in figura 3.23.

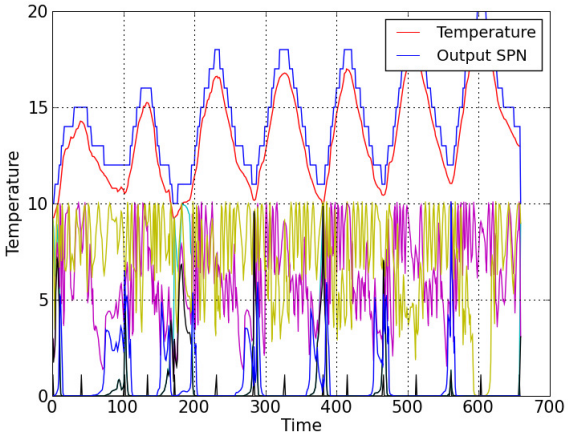


Figura 3.21: Andamento della temperatura in una settimana primaverile

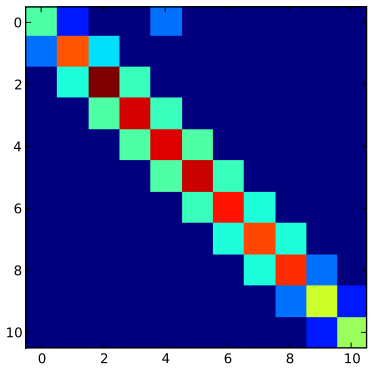


Figura 3.22: Time Adjacency Matrix. L'ascissa rappresenta gli stati di ingresso, l'ordinata gli stati d'uscta.

Come nel caso precedente, il numero di gruppi che descrivono gli eventi di nostro interesse sono due: quello viola individua correttamente i fronti di salita e quello

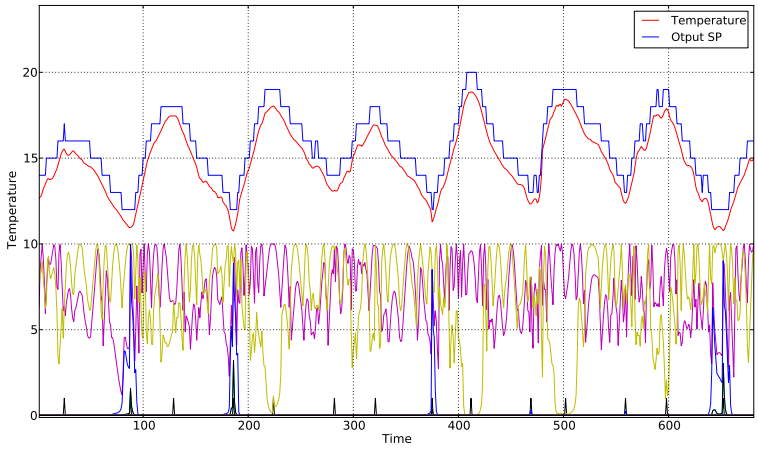


Figura 3.23: Analisi temperatura in un mese primaverile (dettaglio)

giallo i fronti di discesa. All'interno del nodo si sono formate ulteriori memorie, dato l'apprendimento non supervisionato, ma che risultando sempre inferiore agli altri segnali non sono mai attivate.

Una tipologia differente di dati di temperatura è costituita dalle sequenze corrispondenti alle temperature estive che presentano una escursione più marcata e un insieme di profili di salita e discesa più vario.

Addestrando una rete HTM con la medesima topologia (1 nodo) le sequenze estratte dai dati, nonostante i punti di reset, non corrispondono alle salite o discese complete ma solo a parti di esse, come è possibile osservare in figura 3.24(a).

Per poter riconoscere sequenze più complesse è necessario sfruttare le caratteristiche di gerarchia di una rete HTM, in particolare aggiungendo un secondo nodo che riceve come ingresso l'uscita di quello a più basso livello è possibile individuare delle *sequenze di sequenze*. Tale nodo combina le varie sotto parti di salita e discesa e descrive gli eventi completi. È possibile osservare dalla figura 3.24(b) come questo procedimento porti ad un buon risultato.

L'altra tipologia di dati raccolti dai nodi della wireless sensor network riguardano la misura dell'umidità dell'ambiente. La procedura di analisi dei dati è stata effettuata in maniera analoga a quella della temperatura e definendo il medesimo obiettivo di individuare fronti di salita e discesa.

Per raggiungere tale scopo è stata creata una rete a due livelli che, come mostrato in figura 3.25, è stata in grado di riconoscere le due tipologie.

Limitazioni

I test effettuati su queste parti del dataset hanno dimostrato come le Hiearchical Temporal Memories riescano ad estrarre dal un insieme di sequenze temporali le loro componenti di base e a combinarle secondo una gerarchia per memorizzare delle sequenze più complesse.

Nonostante ciò se forniamo alla implementazione HTM tutto l'insieme di dati, sia di umidità che di temperatura, è possibile osservare che la rete incontra varie difficoltà nell'individuare gli *invarianti* nel dataset.

In particolare le problematiche individuate riguardano:

- durante la fase di addestramento:
  - i nodi *TemporalPooler* creano un insieme troppo vasto di stati poichè non sono in grado di modellare in maniera adeguata eventi di lunghezza differente, in particolare quelli eventi in cui un valore di temperatura/umidità e ripetuto più volte;
  - l'algoritmo di formazione dei gruppi non si comporta in maniera prevedibile
  - la quantizzazione dei dati in ingresso è una procedura che dovrebbe essere integrata direttamente nei nodi di più basso livello in maniera da rendere l'utilizzo di questa procedura maggiormente integrato
- durante la fase di inferenza:
  - nodi TemporalPooler ad ogni istante aggiornano il proprio stato utilizzando l'ingresso corrente e cambiano di conseguenza la loro uscita. Tale

meccanismo potrebbe essere ottimizzato cercando di privilegiare, se possibile, la stabilità di uscita mantenendo il gruppo correntemente vincitore come tale in maniera da facilitare il compito di raggruppamento delle sequenze del nodo superiore.

- la rete non utilizza le informazioni di feedback previste dalla teoria fondante delle HTM e non è in grado di effettuare delle predizioni sui futuri valori attesi.

Tali limitazioni hanno motivato la creazione di un insieme di algoritmi e strutture dati innovative che cercando di risolvere la maggior parte dei problemi individuati. Tali soluzioni saranno presentate nel capitolo successivo.

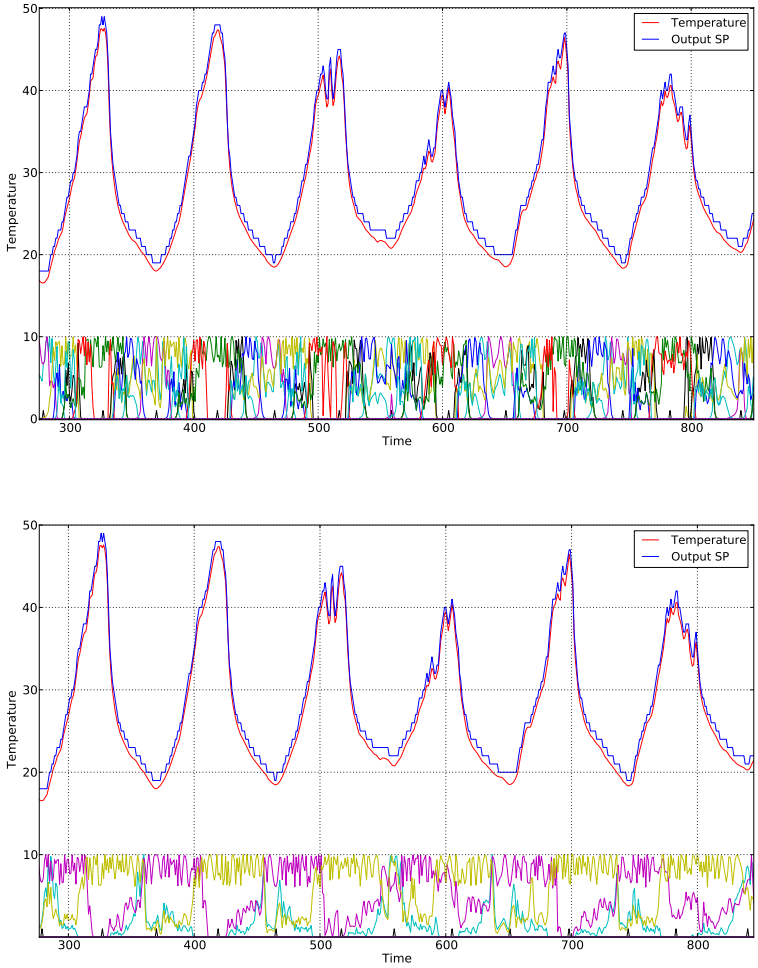


Figura 3.24: Output del (a) TemporalPooler del primo livello e del (b) TemporalPooler del secondo livello nel mese estivo. Nella figura (a) i gruppi vincenti non identificano nessun evento ma tratti dell'evento. La situazione migliora nella figura (b) in cui i gruppi vincenti sono due: uno per il fronte di salita e uno per il fronte di discesa.

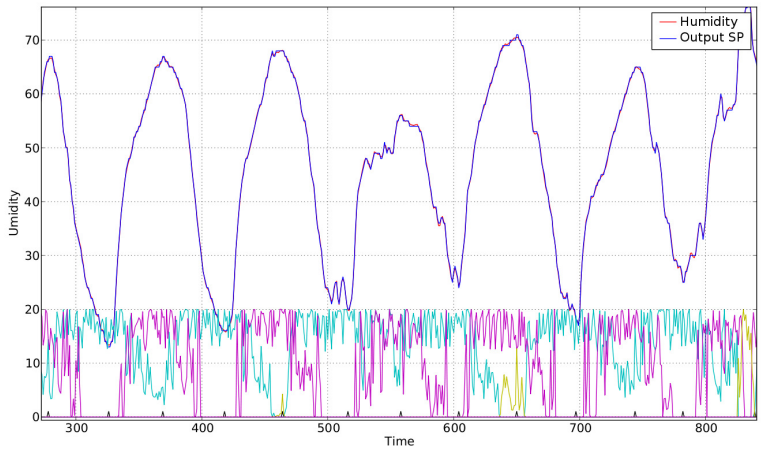


Figura 3.25: Analisi dell'umidita nel mese estivo

Capitolo 4

Estensione alle HTM

L'arte di scoprire le cause dei fenomeni,
o le vere ipotesi, è come l'arte della decrittazione,
in cui un'ingegnosa congettura accorcia grandemente la strada.

– Gottfried Wilhelm von Leibniz

Nei precedenti capitoli sono state presentate delle applicazioni delle Hiearchical Temporal Memories e ne sono stati evidenziate anche alcune limitazioni 3.3. Tali limitazioni sono dovute principalmente alla non completa rappresentazione della teoria di riferimento, Memory Prediction Framework, introdotta nel capitolo 1.1.

In questo capitolo viene proposta una estensione delle Hiearchical Temporal Memories finalizzata a renderle più adatte all'analisi di un insieme di segnali monodimensionali. Tale estensione introduce nuovi componenti architetturali, come l'introduzione di componenti esterne per la gestione dell'ordine di esecuzione dei livelli della rete, componenti algoritmiche, come nuove tipologie di nodi, e strategie di addestramento differenti per i vari livelli della rete. Nella presentazione di questa nuova architettura verranno evidenziate le componenti non presenti nella implementazione delle Hiearchical Temporal Memories realizzata da Numenta inc.

Una particolare attenzione è stata posta alle prestazioni del sistema per ottenere un funzionamento senza interruzioni di una rete a vari livelli, capace di analizzare

(classificare, ricostruire, predire fino a 70 step avanti) dati ad forniti ad una frequenza di circa 30 hz. Al variare della dimensione della rete e della complessità e numero delle memorie imparate le prestazioni possono variare anche considerevolmente.

Questa innovativa architettura ha portato alla sottomissione di un brevetto internazionale dal titolo "*Artificial memory system and method for use with a computational machine for interacting with dynamic behaviours*" [44].

Nell'ultima parte di questo capitolo sarà presentato un procedimento per addestrare una rete che implementa da questi nuovi algoritmi.

4.1 Una nuova architettura per un Sistema di Memoria Artificiale

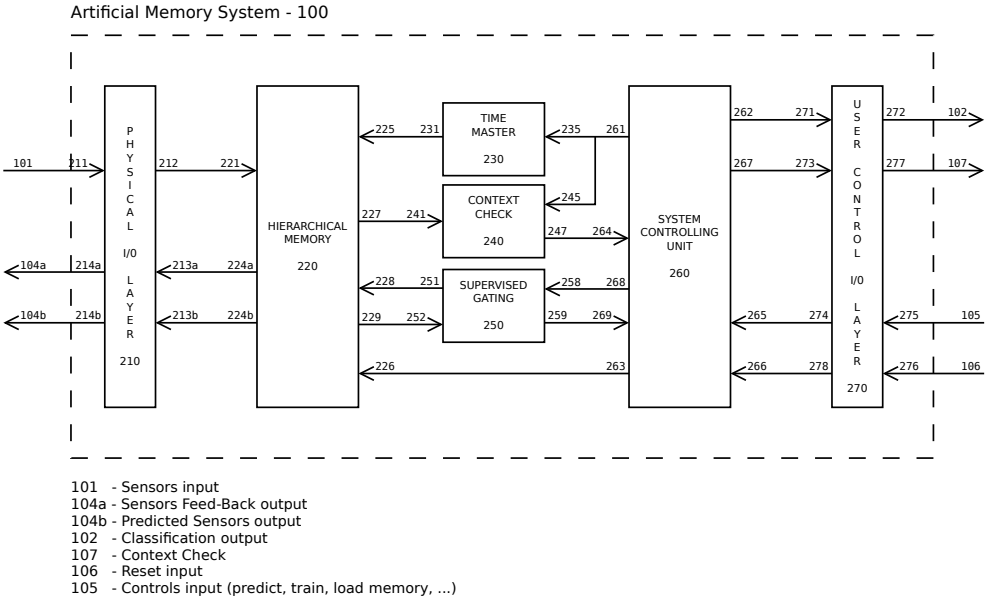


Figura 4.1: Architettura del Sistema di Memoria Artificiale

L'architettura proposta è rappresentata in figura 4.1 e si compone di :

- *Physical I/O Layer*: questo modulo traduce gli input dei sensori in un formato trattabile dalla *HierarchicalMemory*, senza modificarne il contenuto e viceversa, ovvero traducre le uscite di feedback delle *HierarchicalMemory* in un formato compatibile con eventuali attuatori. Tale modulo è parzialmente presente nella corrente implementazione Numenta, in particolare non è presente la componente di traduzione inversa.

- *HierarchicalMemory*: questo modulo corrisponde alla funzione svolta dalle intere HTM Numenta e corrisponde alla parte centrale del sistema. I suoi componenti saranno descritti nel paragrafo 4.1.3.

- *TimeMaster*: questo modulo è un server esterno che gestisce il *clock* di ogni componente della *HierarchicalMemory*. Esso converte comportamenti di alto livello (come per esempio la predizione) in una sequenza specifica di esecuzione e di attivazione delle sequenze della rete. Una descrizione dettagliata sarà fornita nel paragrafo 4.1.1.

- *ContextCheck*: questo modulo misura il grado di "comprensione" della *HierarchicalMemory* rispetto allo stato e agli ingressi correnti. Maggiori dettagli saranno presentati nel paragrafo 4.1.2.

- *SupervisedGating*: questo modulo traduce i comandi della *SystemControlUnit* per abilitare solamente specifiche sottoparti della *HierarchicalMemory* in maniera da ottenere classificazioni anche da livelli intermedi della rete.

- *SystemControlUnit*: questo modulo gestisce la comunicazione tra i vari sottomoduli del sistema di memoria artificiale per ottenere comportamenti ad alto livello definiti dall'utente.

- *User Control I/O Layer*: questo modulo riceve i comandi dall'interfaccia utente e li traduce in comandi per la *SystemControlUnit*.

4.1.1 TimeMaster

Il *TimeMaster* può essere considerato lo *scheduler* interno del Sistema Artificiale di Memoria ed è controllato a sua volta dalla *System Controlling Unit*. Quando un comportamento ad alto livello è selezionato (ad esempio classificazione, predizione, filtraggio dati), il *TimeMaster* genera la sequenza ottimale di esecuzione di ciascun livello della memoria gerarchica.

Il *TimeMaster* indica ad ogni nodo il tipo di computazione da eseguire (anche più di una allo stesso tempo):

- calcolo feed-forward
- calcolo feed-back
- nessuna operazione
- salvataggio stato interno, propedeutico all'inizio di un ciclo di predizione
- ripristino stato interno, al termine di un ciclo di predizione

Il *TimeMaster* può comunicare in due modi differenti con i nodi:

1. *MasterMode*: ogni nodo contatta il *TimeMaster* (per esempio via socket unix) prima dell'esecuzione e riceve la tipologia di calcolo da eseguire per il corrente istante temporale o per un periodo più lungo;
2. *BackgroundMode*: ogni nodo si registra al *TimeMaster* ed esso contatta i nodi indipendentemente ed in maniera asincrona per determinare il loro ciclo di esecuzione.

La scelta tra le due modalità è un compromesso tra un maggior grado di libertà del sistema ed un minor ritardo di comunicazione durante l'esecuzione del sistema di memoria.

La prima modalità permette di cambiare modo di esecuzione solamente quando i nodi comunicano con il *TimeMaster*: più è lungo il set di istruzioni comunicato ogni volta e maggiore sarà il ritardo con cui la rete può cambiare modalità di funzionamento ma anche l'overhead di comunicazione sarà ridotto.

La seconda modalità ha un maggior overhead poiché richiede che i nodi ascoltino costantemente le comunicazioni fornite dal *TimeMaster*, ma permette una velocità di commutazione di funzionamento maggiore.

Per ogni nuovo pattern in ingresso il sistema di memoria artificiale necessita di varie iterazioni per poter completare la classificazione o la predizione, perciò il *TimeMaster* deve permettere al sistema di operare ad una velocità maggiore (SystemClock) rispetto alla velocità dei dati in ingresso (WorldClock)

La figura 4.2 mostra il ciclo di esecuzione per ogni livello di una rete a tre livelli in fase di inferenza (con i segnali di feedback e feedforward abilitati). La parte sinistra della figura rappresenta una possibile rete mentre la parte destra contiene i comandi impartiti dal *TimeMaster* ad ogni livello step-by-step. La parte inferiore della figura rappresenta il tempo.

La figura 4.3 mostra il ciclo di esecuzione per ottenere una predizione a 4 passi avanti. È possibile notare che per generare una predizione di 4 passi avanti sono necessari 7 passi, e quindi 7 cicli di esecuzione della rete. Il numero di cicli necessari cresce linearmente con il numero di passi di predizione richiesti.

4.1.2 ContextCheck

Il modulo *ContextCheck* riceve i segnali *State Consistency Check* direttamente da tutti i nodi presenti nella *HiearchicalMemory* ed è controllato dalla *System Controlling Unit*.

L'obiettivo di questo modulo è quello di fornire una misura del grado di "comprensione" della *HierarchicalMemory* rispetto alla corrente sequenza di ingressi.

Una possibile realizzazione di questo modulo può confrontare le informazioni correnti di feedback e feedforward: se l'intersezione tra esse non è nulla significa che vi è accordo tra quanto osservato dal sensore e lo stato dell'intera rete.

Una differente strategia si basa sull'accumulo di statistiche a breve termine durante l'esecuzione del nodo e l'esecuzione di predizioni locali (ovvero solo sul nodo stesso). Se negli ultimi campioni i valori di predizione si sono successivamente

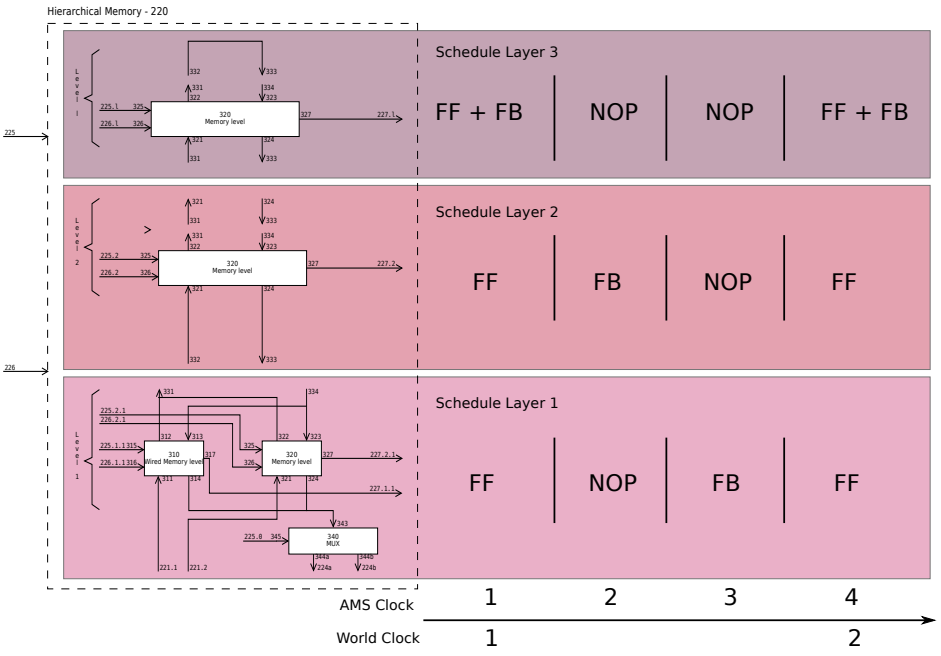


Figura 4.2: Schedule di esecuzione standard

realizzati è possibile considerare come il nodo abbia effettivamente "compreso" la sequenza corrente.

Per esempio la *SCU* può richiedere la valutazione di consistenza dell'intera *HierarchicalMemory*; tale informazione può essere utile per decidere se richiedere o meno delle predizioni: se lo stato non è consistente le predizioni ottenute sarebbero certamente errate.

Per ulteriore esempio la *SCU* può richiedere tale valutazione solo su una parte della memoria, ad esempio quella di basso livello per verificare che i segnali di ingressi rientrino fra quelli conosciuti dal sistema. Nel caso non lo siano è possibile pensare di memorizzare tali segnali per una successiva fase di addestramento

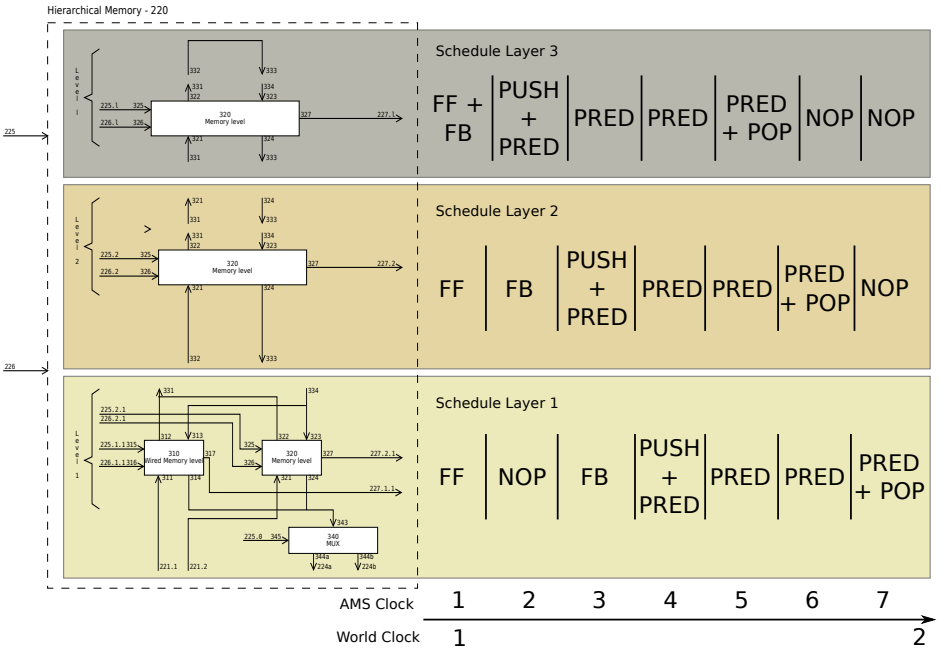


Figura 4.3: Schedule di esecuzione per ottenere 4 passi di predizione

incrementale.

4.1.3 Hierarchical Memory

La figura 4.4 rappresenta il cuore dell'intero sistema e sostituisce l'intera implementazione Hierarchical Temporal Memory creata da Numenta. Il suo scopo principale è quello di imparare, riconoscere, capire e, sotto richiesta, predire sequenze di pattern in ingresso. Per raggiungere tale scopo vari *MemoryLevel* sono impegati e connessi insieme in maniera gerarchica. Il numero di questi livelli può variare in base alla particolare applicazione. Ogni livello di memoria può essere eventualmente associato ad

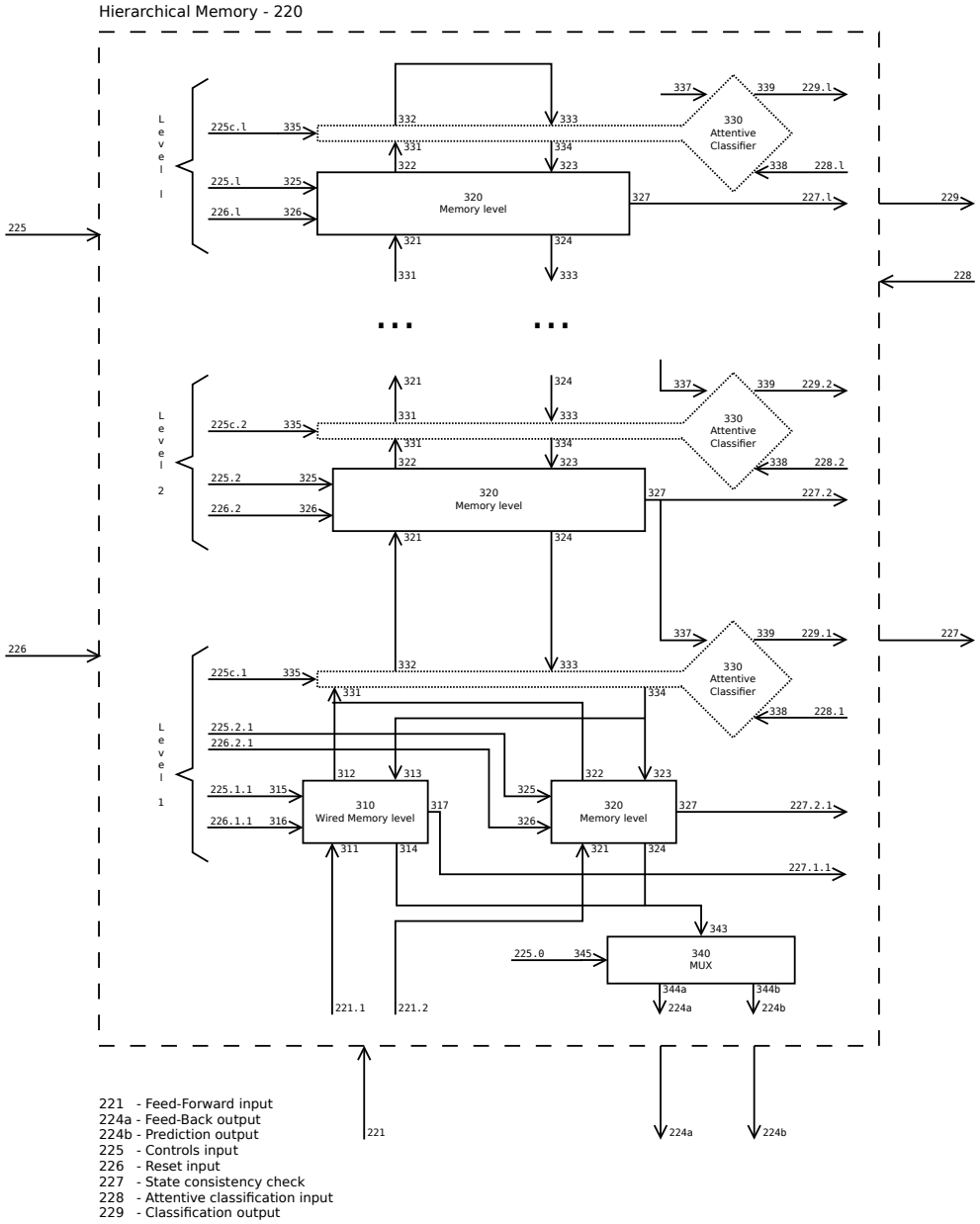


Figura 4.4: Hierarchical Memory

un *AttentiveClassifier* per analizzare tramite apprendimento supervisionato le uscite di tale livello e per così implementare meccasismi di *focus-of-attention*.

I livelli della rete sono in un numero arbitrario e tutti costituiti dagli stessi algoritmi ad eventuale eccezione del primo livello. Il primo livello può essere del tipo *Wired Memory Level*, presentato nel paragrafo 4.1.4. Tale livello può essere impiegato per analizzare ingressi mono-dimensionali per cui è richiesto un elevato grado di accuratezza nella predizione/ricostruzione. Nonostante ciò è possibile impiegare ugualmente un *Memory Level* standard anche al primo livello se questi vincoli vengono a mancare.
Di seguito verranno descritti vari blocchi della figura 4.4.

Multiplexer

Il blocco *Multiplexer* permette che le uscite di feedback e di predizione siano disponibili al di fuori del nodo su uscite differenti.

4.1.4 Wired Memory Level

La creazione del *Wired Memory Level* proviene dal riconoscimento all'interno del nostro cervello di differenti tipologie di memorie. In particolare un ruolo importante viene giocato dalle memorie istintive. Se prendiamo come esempio le memorie istintive motorie possiamo renderci conto di come queste siano in grado di descrivere brevi e semplici movimenti con estrema perfezione senza la nostra supervisione. Inoltre è possibile apprendere nuove memorie istintive purché semplici.

La figura 4.5 include un insieme di nodi organizzati in colonne distinte. Il numero totale delle colonne può variare in base al numero dei segnali monodimensionali in ingresso.

Una colonna è composta da un *Signal Quantizer node (SQ)* e da un *Forced Temporal Pooler node (FTP)*. Uno *Smoother node* può essere aggiunto all'uscita di feedback della colonna per post-processare il segnale predetto o ricostruito. Inoltre è possibile aggiungere uno o più *Forced Event Pooler node (FEP)* in cima alla colonna per

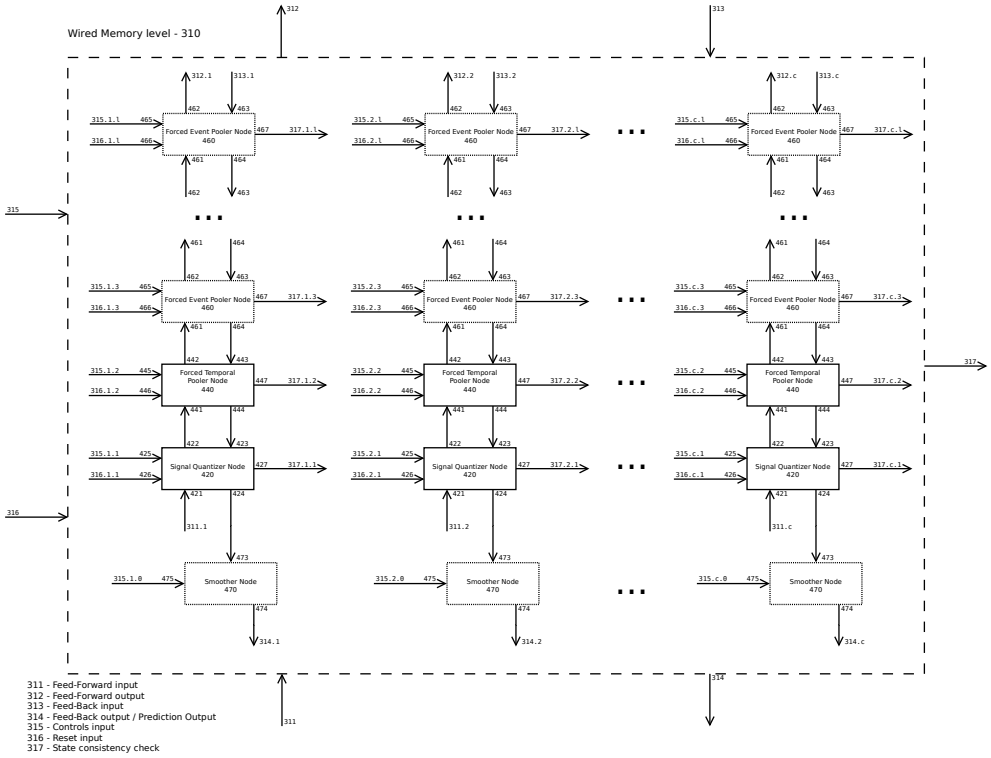


Figura 4.5: Wired Memory Level

aumentare la “durata” temporale e la complessità delle memorie descritte da questo livello.

L’obiettivo di questo blocco è descrivere un certo numero di ingressi monodimensionali in termini di ”primitive”. Ogni colonna elabora un singolo ingresso monodimensionale, perciò sono necessarie tante colonne quanti sono gli ingressi al sistema che si vogliono descrivere in tale maniera.

I nodi *SQ* permettono di quantizzare i segnali di ingresso in maniera controllata e molto efficiente, a patto di conoscere la dinamica e caratteristiche di tali segnali. Tali nodi saranno presentati nel paragrafo 4.2.2.

Le “primitive” del segnale possono essere estratte tramite analisi statistica dei dati in ingresso per individuare brevi pattern ricorrenti. La figura 4.20 mostra un possibile

insieme di primitive estratte da un dataset.

Una volta estratte è possibile "insegnare" queste sequenze ai nodi FTP o FEP in maniera forzata, ovvero creando delle sequenze di stati e gruppi ad-hoc per descrivere tali memorie. Durante la fase di inferenza questi nodi potranno descrivere i segnali in ingresso in funzione delle "primitive" memorizzate. Tali primitive devono essere in grado di coprire la maggior parte (idealmente tutti) delle possibili componenti dei segnali in ingresso.

Il principale vantaggio di questo approccio può essere apprezzato durante l'utilizzo del sistema in modalità di predizione/ricostruzione. In particolare le uscite di feedback del livello potranno descrivere solamente delle sequenze "note" poiché sono state imposte forzatamente nella fase di addestramento. Questa caratteristica determina una più elevata qualità dei segnali predetti poiché tali predizione seguiranno delle memorie note della cui veridicità è stato possibile assicurarsi. In particolare sarà possibile per una applicazione utilizzare i segnali predetti/ricostruiti al posto di quelli reali, se necessario.

I nodi *Smoother*, se presenti, filtrano l'uscita di feedback del *SQ* per ridurre gli effetti che la quantizzazione del segnale introduce.

4.1.5 Generic Memory Level

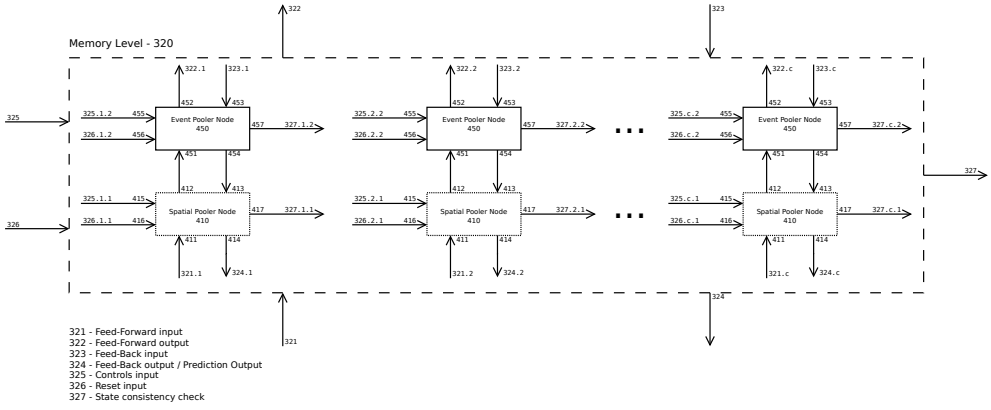


Figura 4.6: Generic Memory Level

Il *Generic Memory Level*, descritto in figura 4.6, comprende un insieme di nodi di base organizzati in colonne distinte. Il numero totale di colonne dipende dalla struttura generale della rete e non dipende direttamente dal numero di ingressi al sistema complessivo. Ogni singola colonna è composta da uno *Event Pooler node (EP)* ed, opzionalmente, da un *Spatial Pooler node (SP)*.

Lo scopo di questo blocco è quello di apprendere e riconoscere sequenze di "coincidenze" segnalate dai nodi ai livelli inferiori. Questa funzionalità è assicurata dalla presenza del nodo *EP* e dalle sue peculiari caratteristiche di "registrazione compressa" e "riconoscimento selettivo" degli eventi.

Il nodo *SP* agisce invece da quantizzatore all'ingresso del nodo per ridurre la complessità spaziale del segnale di ingresso.

All'interno di una *Hiearchical Memory* vi possono essere vari blocchi di tale tipologia adibiti al riconoscimento di sequenze temporali sempre più "lunghe" man mano che si sale dal fondo fino alla cima della gerarchia.

4.1.6 Attentive Classifier

Il modulo *Attentive Classifier* è opzionale ed è trasparente rispetto ai segnali scambiati tra i livelli. Le sue principali caratteristiche sono:

- filtra i segnali tra i livelli in modo da creare *focus-of-attention* silenziando alcuni gruppi in uscita dal nodo
- abilita la ricezione da parte di un nodo di segnali di feedback da più nodi, implementando un controllo di coerenza o di priorità
- classifica eventualmente gli output di un nodo intermedio per identificare dei comportamenti a differenti scale di tempo

Durante un ciclo "normale" di esecuzione tale componente opera da pass-through. Nel caso una colonna di un livello di memoria riceva segnali di feedback da più sorgenti è necessario l'intervento di questa componente per ridurli ad un segnale univoco. Possibili strategie sono:

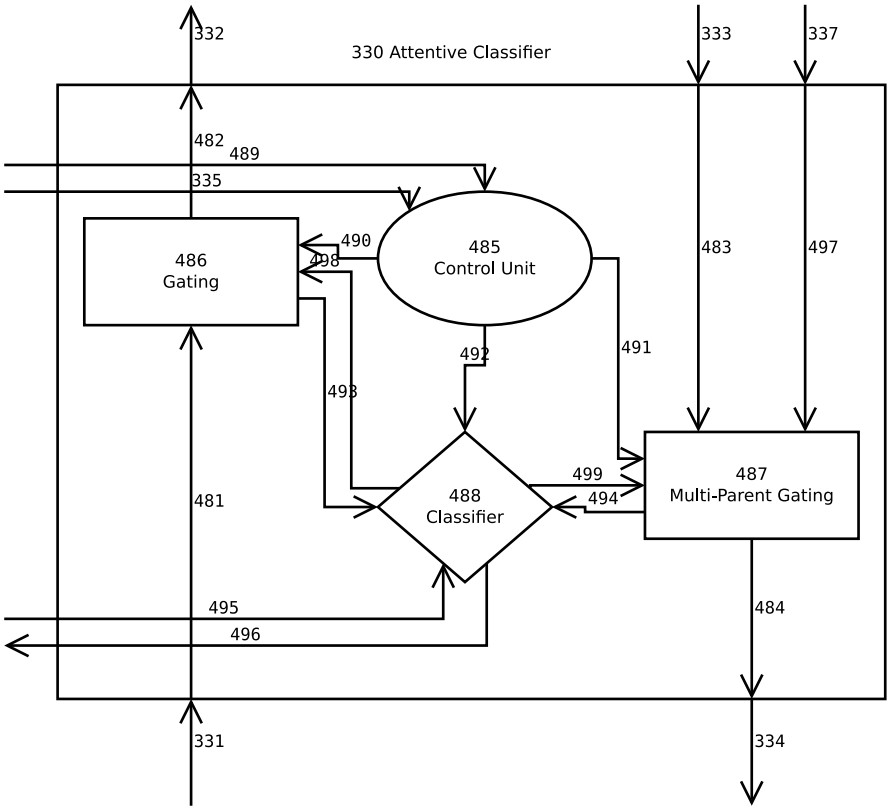


Figura 4.7: Attentive Classifier

- considerare sempre il segnale proveniente da una determinata sorgente
- considerare il segnale che proviene dalla sorgente con un livello di "stabilità" maggiore.

Nel caso la *System Controlling Unit* imponga la creazione di una *focus-of-attention* a livello di rete, tale componente dovrà silenziare le parti indicate del vettore di feedforward o di feedback. È possibile ricordare come tali vettori contengano la descrizione attuale, agli "occhi" del nodo, degli ingressi. Silenziandone una parte, ad

esempio quella maggiormente attiva, è possibile evidenziare componenti secondarie nei segnali in ingresso.

4.1.7 Basic Network Nodes

Le tipologie di nodi presenti nei vari livelli di memoria sono i seguenti:

- Spatial Pooler Node (SP)
- Signal Quantizer Node (SQ)
- Temporal Pooler Node (TP)
- Forced Temporal Pooler Node (FTP)
- Event Pooler Node (EP)
- Forced Event Pooler Node (FEP)
- Smoother Node (Sm)

Tutti questi nodi condividono la rappresentazione dei segnali di feed-forward e di feed-back che restano compatibili tra loro.

Nella prossima sezione sarà presentata l'implementazione proposta di tali nodi.

4.2 Implementazione dei nodi

4.2.1 Spatial Pooler Node (SP)

Differenze dall'implementazione Numenta:

- aggiunto il calcolo del feed-back;
- aggiunto il supporto per il *TimeMaster*;
- diminuzione dei tempi di calcolo tramite ottimizzazioni dell'implementazione.

Caratteristiche:

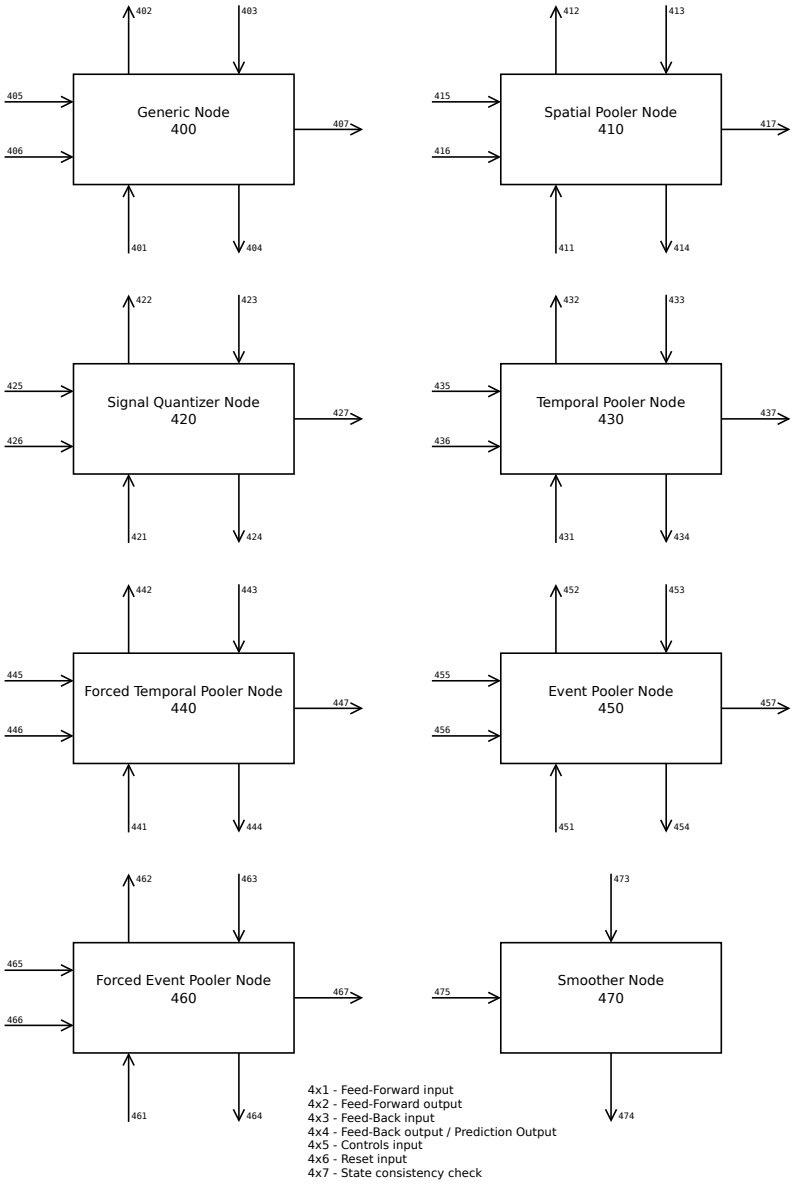


Figura 4.8: Basic Network Nodes

- riceve in ingresso input multidimensionali (le uscite di altre colonne)
- scopre relazioni spaziali tra gli ingressi quantizzando lo spazio di ingresso in un numero massimo predeterminato di "coincidenze".

Fase di Addestramento

Durante la fase di addestramento lo *Spatial Pooler Node* memorizza al massimo *maxCoincN* coincidenze. Una coincidenza è una combinazione degli ingressi del nodo. Tale nodo aggiunge una nuova coincidenza ogni qualvolta la distanza Euclidea da tutte quelle già memorizzate è $\geq maxDist$.

Fase di Inferenza - Feedforward

Durante la fase di inferenza feedforward il nodo calcola la distanza dell'ingresso corrente rispetto a tutte le coincidenze memorizzate. L'uscita FF_{out} del nodo rappresenta la distribuzione di probabilità dell'ingresso attuale di essere rappresentato dalle coincidenza memorizzate, ovvero quale delle coincidenze memorizzate è più simile all'ingresso corrente. La dimensione di FF_{out} è pari al numero di coincidenze memorizzate.

Fase di Inferenza - Feedback

Durante la fase di inferenza feedback il nodo riceve sull'ingresso FB_{in} la distribuzione di probabilità di attivazione delle proprie coincidenze memorizzate da parte di un nodo superiore. Per individuare la coincidenza vincente è possibile scegliere quella con il valore di attivazione più alto. Successivamente il valore di tale coincidenza viene recuperato dalla memoria dello *Spatial Pooler* e copiato sul link di uscita FB_{out}. In differenti implementazioni è possibile pensare ad una modalità di calcolo di FB_{out} che consideri tutte le coincidenze attivate sopra una certa soglia e non solo quella con il valore massimo. Tale metodologia porta ad ottenere una uscita di feedback differente dai valori memorizzati.

È possibile individuare alcuni svantaggi di questo algoritmo:

- maggiore è il numero delle coincidenze memorizzate e maggiore sarà il tempo di esecuzione del nodo (per ogni iterazione è necessario calcolare la distanza da tutte le coincidenze memorizzate)
- questo algoritmo non è ottimizzato per segnali monodimensionali (mentre è adeguato per i segnali multidimensionali che si incontrano tra i vari *Memory Level*)
- non è possibile controllare agevolmente il processo di formazione delle coincidenze

4.2.2 Signal Quantizer Node (SQ)

Questo nodo non è presente nell'implementazione Numenta.

Il suo scopo è simile a quello dello *SpatialPooler Node* ma è progettato per gestire segnali monodimensionali provenienti dai sensori con un *range* noto.

Il *Signal Quantizer node* utilizza coincidenze statiche pre-definite all'interno di un intervallo in maniera da lasciare il pieno controllo sul posizionamento dei centri/-livelli di quantizzazione. Se non specificate le coincidenze sono definite uniformemente nell'intervallo impostato.

I parametri del nodo sono:

- *maxCoincidenceCount*: numero predefinito di coincidenze
- *minInputValue*: limite inferiore dell'intervallo
- *maxInputValue*: limite superiore dell'intervallo
- *sigma*: parametro *sigma* della gaussiana utilizzata per calcolare il FF_{out}
- *minOutputValue*: minimo valore che può assumere un elemento di FF_{out}. Gli elementi con un valore di attivazione minore sono azzerati, sia per semplificare i segnali tra i livelli sia per diminuire i tempi di calcolo dei livelli superiori.

- *coincidenceList*: parametro opzionale, è possibile specificare la lista delle coincidenze manualmente.

Fase di Addestramento

La fase di addestramento di questo nodo si riduce all'impostazione dei parametri.

Fase di Inferenza

La fase di inferenza è analoga a quella del *Spatial Pooler node*, in particolare vengono calcolate le distanze rispetto alle coincidenze memorizzate e da esse viene calcolata la probabilità che l'ingresso corrente possa essere descritto da essa. Infine la prima e l'ultima coincidenza sono speciali: si attivano quando il valore corrente esce dall'intervallo descritto da *minInputValue* e *maxInputValue*.

In figura 4.9 è mostrato un esempio di distribuzione dei centroidi e delle rispettive coincidenze mentre nel listato 4.1 vi è un esempio di come calcolare l'indice della coincidenza vincitrice nel caso di una distribuzione uniforme dei centri di quantizzazione.

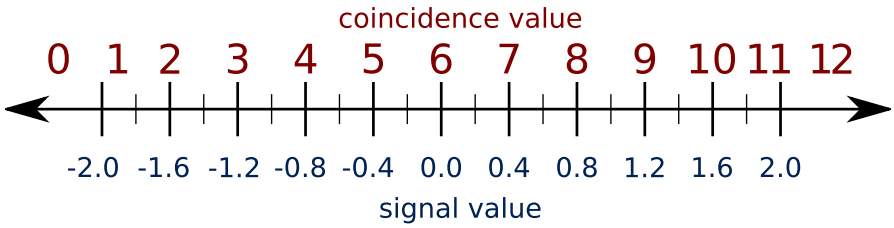


Figura 4.9: Meccanismo di quantizzazione del Signal Quantizer node: in questo esempio si suppone che il range di ingresso del segnale sia $[-2.0, 2.0]$ e il numero desiderato di coincidenze sia 13

Rispetto ad uno *Spatial Pooler node* questo nodo ha i sequenti vantaggi:

- centri di quantizzazione predefiniti: è possibile minimizzare l'errore di quantizzazione

Listing 4.1: Winning Coincidence Index computation for the Signal Quantizer Node

```
minInputValue_ //lower limit for the input range
maxInputValue_ //upper limit for the input range
maxCoincidenceCount_ //pre−defined number of coincidences
inputValue //current (raw) input value
coincidenceID //FFoutput value to parent node

distance_ = (maxInputValue_ − minInputValue_) / (maxCoincidenceCount_ − 3);

if (inputValue > maxInputValue_)
        coincidenceID = maxCoincidenceCount_ −1;
else if (inputValue < minInputValue_)
        coincidenceID = 0;
else
        coincidenceID = (int) rint( (inputValue − minInputValue_) / distance_ ) + 1;
```

- esecuzione più veloce (nel caso di distribuzione uniforme)
- l'indice della coincidenza è direttamente proporzionale al valore del segnale. Tale proprietà sarà utile in fase di predizione.

Gli svantaggi invece sono:

- l'intervallo e la dinamica del segnale devono essere noti in anticipo
- il numero di coincidenze e la posizione dei centri di quantizzazione non può essere determinata in base ai dati

4.2.3 Temporal Pooler Node (TP)

Temporal Pooler node

Differenze dall'implementazione Numenta:

- aggiunto il calcolo del feed-back;
- aggiunto il supporto per il *TimeMaster*;
- diminuzione dei tempi di calcolo tramite ottimizzazioni dell'implementazione.

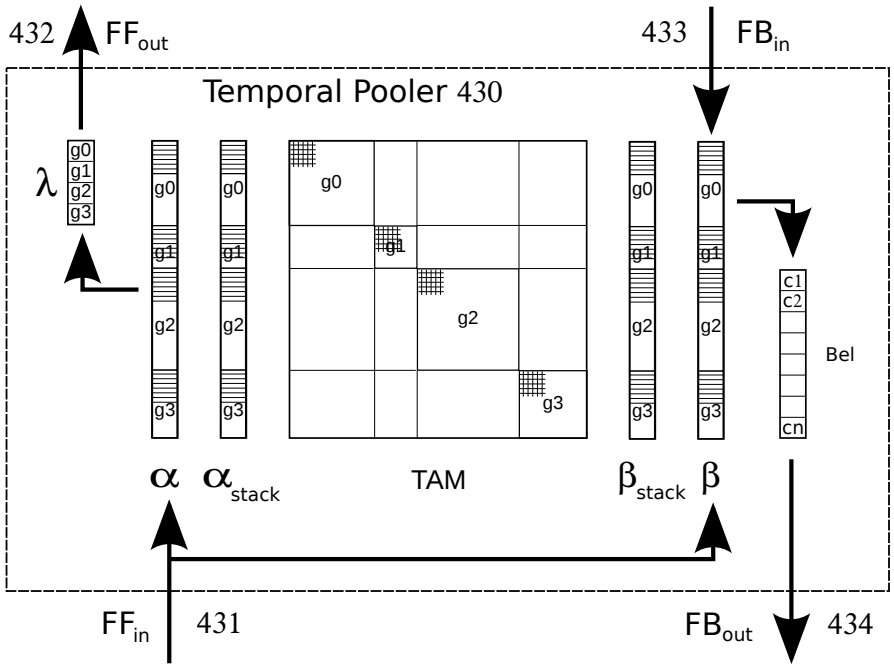


Figura 4.10: Struttura interna di un Temporal Pooler node.

- ridefinito il significato del segnale di *reset* per migliorare il processo di *splitting* (Markers)

Il compito principale di un *Temporal Pooler node* è quello di ricevere in ingresso un vettore multi-dimensionale rappresentante una distribuzione di probabilità delle cause (solitamente nello spazio delle coincidenze) e dalla loro evoluzione nel tempo, scoprire relazioni spazio temporali per poi formare delle sequenze (gruppi).

Fase di Addestramento

Durante la fase di addestramento è possibile descrivere il nodo come una *Finite State Machine* dove ogni stato interno è fatto corrispondere ad una singola coincidenza in ingresso. Allo stesso tempo, una *tabella di transizione degli stati* è mantenuta per

essere in grado di specificare differenti "stati destinazione" (dipendenti dallo stato corrente) per le stesse coincidenze in ingresso.

Ad ogni ingresso il nodo aggiorna una *Time Adjacencty Matrix - TAM* incrementando un contatore per la transizione "stato precedente $\rightarrow$ stato corrente". Nel caso durante tale ingresso l'ingresso di *reset* del nodo sia attivo, tale transizione non viene marcata; tale procedura viene impiegata per meglio evidenziare l'inizio e la fine di una particolare sequenza di ingresso e perciò per prevenire che esse siano raggruppate insieme nella TAM finale. È possibile addestrare un *Temporal Pooler node* anche senza alcun ingresso di *reset*.

Inoltre è possibile richiedere periodicamente al nodo il controllo di stati altamente interconnessi per verificare la necessità di una fase di *splitting*, come mostrato in figura 4.11. Tale processo permette di separare differenti sequenze che condividono alcune coincidenze in ingresso, perciò l'algoritmo di *grouping* viene aiutato ad identificare sequenze differenti come gruppi differenti. Dopo ogni fase di splitting in cui almeno un nodo è stato raddoppiato l'intera *TAM* è azzerata poiché sono stati introdotti nuovi stati.

Nonostante tale meccanismo di replica degli stati, è impossibile separare due differenti catene che condividono la parte iniziale. Per questo motivo è probabile che l'algoritmo di creazione gruppi incontri una situazione come quella rappresentata in figura 4.12.

Per comprendere meglio il problema basti pensare a due differenti canzoni che iniziano con la stessa sequenza di note prima di "prendere direzioni differenti". Fin dall'inizio sarà possibile individuare queste due canzoni come possibili candidati e si potrà escludere una delle due solo quando le canzoni di differenzieranno. Quanto è descritto nella figura 4.12 è molto differente: dall'inizio solamente una canzone può essere identificata come l'unico possibile candidato mentre solo dopo la differenziazione sarà possibile scoprire che la canzone corretta era un'altra, non ancora presa in considerazione.

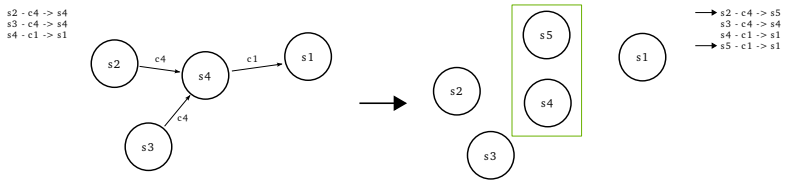


Figura 4.11: Meccanismo di *splitting* di un Temporal Pooler Node

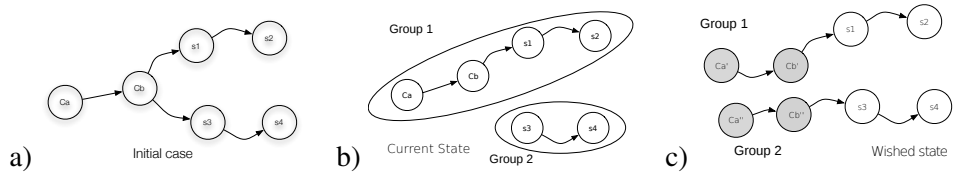


Figura 4.12: Possibili problemi di *grouping*: a) due separate sequenze/eventi che condividono la parte iniziale; b) possibili gruppi derivanti dall'algoritmo corrente; c) risultato desiderate con catene completamente separate.

Algoritmo di addestramento Marker-based

Per poter separare le catene parzialmente sovrapposte è stato introdotto un nuovo meccanismo che permette di utilizzare il segnale di *reset* in una maniera differente.

Quando il *Temporal Pooler node* riceve un ingresso sul canale di *reset* diverso da 0 o 1 esso crea un nuovo "marker". Un *marker* è uno stato interno del nodo che non è mappato su nessuna coincidenza.

In tale maniera modalità di addestramento viene marcata la transizione "marker x $\rightarrow$ stato corrente" (invece di quella classica "stato precedente $\rightarrow$ stato corrente") ogni volta che tale particolare segnale di reset è ricevuto. Questo porta alla situazione descritta in figura 4.13. Se due eventi differenti condividono la parte iniziale delle loro due sequenze, come in figura 4.13(a), ma hanno due differenti *marker*, come in figura 4.13(b), l'algoritmo di *splitting* può separare completamente le due catene permettendo all'algoritmo di raggruppamento di formare (una volta che saranno scartati i marker, ora non più utili) due gruppi separati, come in figura 4.13(c).

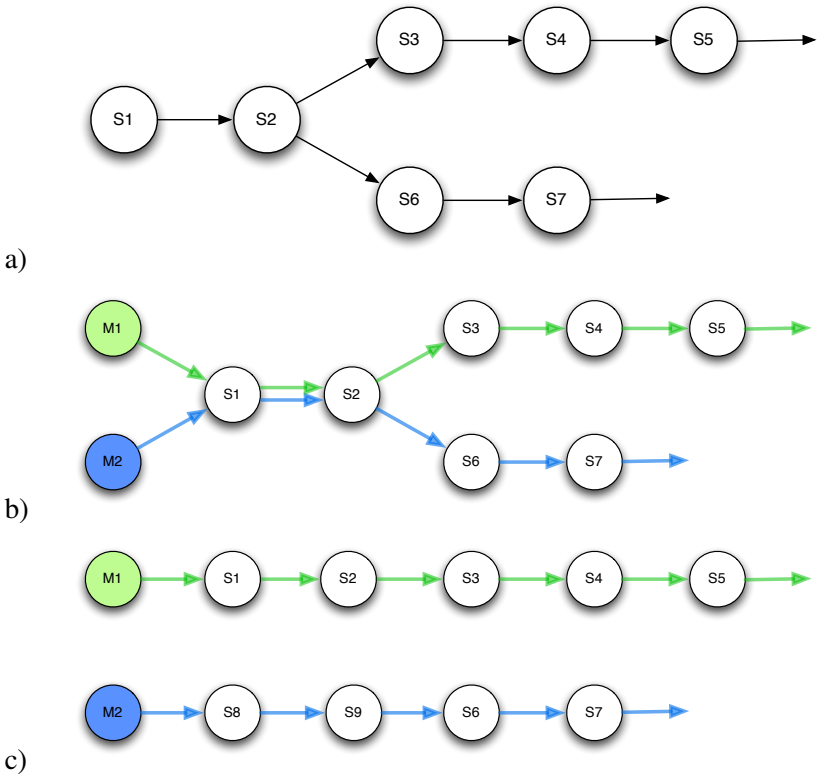


Figura 4.13: Meccanismo di *splitting* che utilizza i *marker*: a) catena in origine non-separabile; b) inserimento dei markers durante l'addestramento; c) separazione finale delle catene dopo lo splitting

Al termine della prima fase di addestramento vi è la fase di "grouping". Lo scopo di tale fase è quello di formare "gruppi" o "sequenze" degli stati interni del nodo che descrivono le sequenze più frequenti degli ingressi osservati durante la prima fase di addestramento. Per raggiungere tale scopo viene utilizzata la *TAM* come un *connection graph* dal quale gli stati più strettamente connessi tra loro sono estratti e raggruppati insieme. Un singolo gruppo è perciò un insieme di stati del nodo che determinano una propria *sotto-TAM*. Un gruppo può anche essere visto come una catena di Markov.

È possibile utilizzare vari algoritmi di raggruppamento, tra cui un "grouping gerarchico" o un raggruppamento iterativo che parte dal nodo con più connessioni fino a generare gruppi a dimensione massima.

Fase di inferenza - Feedforward

Durante la fase di inferenza il *Temporal Pooler node* cerca di individuare a quale gruppo appartiene la corrente sequenza di ingresso, o meglio calcola la probabilità che la sequenza di ingresso appartenga a ciascun gruppo in memoria per aggiornare il link di uscita FF_{out}. Le equazioni utilizzate in questa fase sono simili a quelle impiegate nella implementazione Numenta, descritte nella tabella 1.1.

Facendo riferimento alla figura 4.10, ogni volta che il *Time Master* richiede al nodo di aggiornare il percorso di feedforward lo stato interno di feed-forward (α) è aggiornato nel modo seguente:

$$\alpha_{tmp} = \text{TAM}\ \alpha \tag{4.1}$$

$$\alpha(i) = \alpha_{tmp}(i)\ \text{FF}_{in}(C(i)), \qquad i \in [1, n_s] \tag{4.2}$$

dove n_s è il numero degli stati interni alla *TAM* e C è una funziona che collega ogni stato alla coincidenza di origine. Successivamente il valore di attivazione per ogni gruppo è calcolato come il valore massimo di attivazione tra gli stati che lo compongono attraverso

$$\lambda(i) = \max_{j \in g_i} \alpha(j), \qquad i \in [1, n_g] \tag{4.3}$$

dove n_g è il numero dei gruppi creati dal nodo durante la fase di addestramento.

Infine

$$\mathrm{FF}_{out} = \lambda \tag{4.4}$$

Fase di Inferenza - Feedback

Il meccanismo di inferenza impiegato nella fase di feed-forward è esteso per utilizzare il FB_{input} per filtrare ulteriormente la distribuzione di probabilità calcolata per gli stati interni. Come precedentemente indicato l'uscita FB_{output} è la distribuzione di probabilità sugli ingressi di feed-forward del nodo.

Facendo riferimento alla figura 4.10, ogni volta che il *Time Master* richiede al nodo di aggiornare il percorso di feedback lo stato interno di feed-back (β) è aggiornato con

$$\beta_{tmp} = \mathrm{TAM}\beta \tag{4.5}$$

$$\beta(i) = \beta_{tmp}(i)\,\mathrm{FF}_{in}(C(i))\,\mathrm{FB}_{in}(G(i)), \qquad i \in [1, n_s] \tag{4.6}$$

dove n_s è il numero gli stati interni della *TAM*, C è una funzione che collega ogni stato all'indice della coincidenza corrispondente e G è una funzione che collega ogni stato all'indice del gruppo a cui appartiene. Successivamente il valore di attivazione di ogni coincidenza è calcolato considerando la somma di tutti gli stati interni della *TAM* che sono derivati da quella coincidenza con

$$Bel(i) = \sum_{j \in [1,n_s] \wedge G(j)=i} \beta(j), \qquad i \in [1, n_c] \tag{4.7}$$

dove n_c è il numero delle coincidenze in ingresso (dimensione di FF_{in}). Infine

$$\mathrm{FB}_{out} = Bel \tag{4.8}$$

Come evidenziato anche al termine del precedente capitolo 3.3, l'algoritmo alla base del *Temporal Pooler node* presenta varie limitazioni, in particolare è possibile notare che

- ogni stato di un gruppo corrisponde ad un singolo "time tick". Per tale motivo, sequenze lunghe necessitano di un numero elevato di stati
- è necessaria una corrispondenza stato-per-stato per poter seguire una catena durante la fase di inferenza: se l'ingresso non segue le esatte dinamiche spazio-temporali del gruppo in memoria non vi è garanzia di un output stabile. Per tale motivo le capacità di generalizzazione sono limitate
- infine parte dell'instabilità della fase di inferenza (FF_{out}) è dovuta a "trucchi" di implementazione per risolvere problemi all'interno delle equazioni utilizzate (propagazione di una probabilità nulla all'interno delle equazioni).

Queste limitazioni hanno portato alla concezione di due nuovi nodi: *Forced Temporal Pooler node* e *Event Pooler node*.

4.2.4 Forced Temporal Pooler Node (FTP)

Il nodo *Forced Temporal Pooler* è una variante del nodo *Temporal Pooler*. È stato aggiunto un modo di addestramento "forzato" per poter impostare differenti sequenze di ingresso direttamente come gruppi del nodo (come le "primitive"). Inoltre anche la fase di inferenza è stata modificata considerando tale caratteristica del nodo per rendere l'uscita FF_{out} più stabile e in grado di seguire gruppi/primitive anche se non sono presentate esattamente nelle stesso modo.

Tale nodo ricopre un ruolo determinante nella composizione del "Wired Memory Level" descritto nella sezione 4.1.4.

Con l'impiego di questa metodologia di addestramento il nodo presenta le sequenti caratteristiche:

- i gruppi/primitive/sequenze sono addestrati così come forniti al nodo
- gli stati interni sono perciò creati "artificialmente"
- l'algoritmo di *grouping* standard non viene utilizzati e non vi è necessità di eseguire una fase di *splitting* degli stati
- la fase di addestramento del nodo è estremamente veloce.

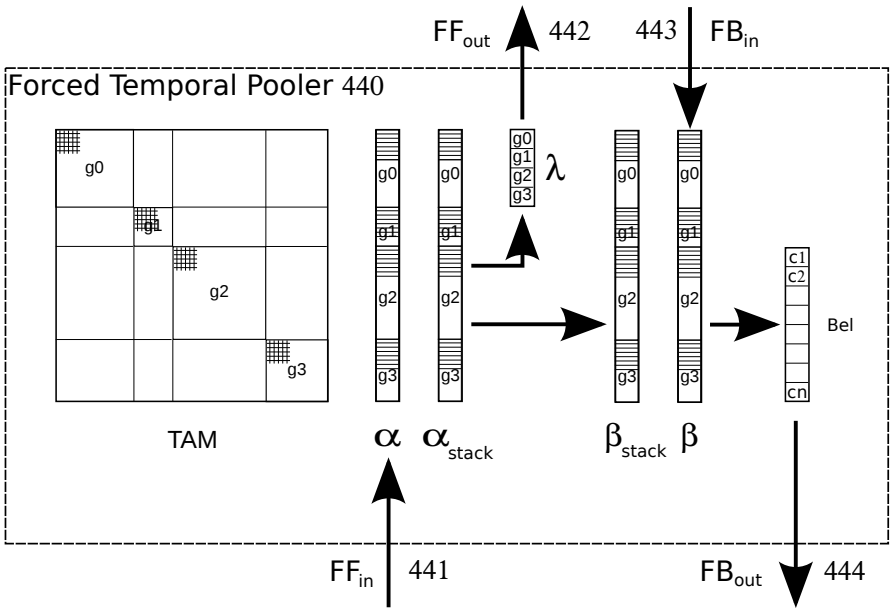


Figura 4.14: struttura interna di un Forced Temporal Pooler node.

Fase di Addestramento

Innanzitutto è necessario estrarre le sequenze desiderate dal dataset tramite meccanismi esterni, come ad esempio analisi statistica sui dati. Successivamente quando vengono presentati i dati di addestramento al nodo, ad ogni nuovo valore di ingresso viene creato un nuovo stato all'interno della *TAM*. Questo nuovo stato è collegato alla coincidenza in ingresso e, se il segnale di reset non è attivo, viene considerata la transizione dallo stato precedente aggiungendola alla tabella delle transizioni e segnata sulla *TAM*.

In tale maniera non è necessario l'impiego di un ulteriore algoritmo di *splitting*. Allo stesso tempo il processo di formazione dei *gruppi* è semplice e lineare: per costruzione, ogni stato può avere un solo predecessore ed un solo successore poiché le differenti catene sono separate dai segnali di reset.

Fase di Inferenza - Feedforward

La fase di inferenza eseguita da questo nodo prende in nome di "inferenza forzata" ed è completamente differente da quella utilizzata dal nodo *Temporal Pooler* descritta nella sezione 4.2.3.

L'obiettivo di questo nodo è quello di "seguire" nel miglior modo possibile il minor numero possibile di gruppi (ricordiamo il caso di gruppi con parti iniziali in comune come descritto nella figura 4.12. Inoltre, allo stesso tempo, è desiderabile che il link di uscita FF_{out} sia il più stabile possibile perciò viene mantenuto attivo solamente un sottoinsieme dei gruppi del nodo, denominato *ActiveGroups*, A_g. Gli stati appartenenti a tutti gli altri gruppi sono mantenuti forzatamente non attivati. Questo approccio permette, e sfrutta, il presentarsi su FF_{out} di "attivazioni multiple" ovvero il nodo può segnalare al livello superiore più di un gruppo attivo, con la medesima probabilità, nello stesso istante di tempo.

Ad ogni instante di tempo, per ogni gruppo attivo, viene controllata la successiva coincidenza attesa all'interno di FF_{in} e il valore di attivazione dello stato successivo

è calcolato come:

$$\alpha(i) = \begin{cases} \alpha(a_s)\mathrm{FF}_{in}(C(N(a_s)) & i : i = N(a_s), \forall a_s \in A_s : G(a_s) \in A_g \\ 0 & \text{altrimenti} \end{cases} \tag{4.9}$$

dove C è una funzione che collega ogni stato alla coincidenza corrispondente, N è una funzione che indica lo stato successivo all'interno del gruppo specificato, G è una funzione che collega ogni stato all'indice del gruppo a cui appartene, a_s è lo stato corrente (quello con il valore massimo di attivazione) di un gruppo specifico, A_s è l'insieme di tutti gli stati attivi e A_g è l'insieme che contiene tutti i gruppi attivi.

Allo stesso tempo i valori di attivazione per tutti i gruppi sono aggiornati come:

$$\lambda(i) = \begin{cases} \alpha(a_s) & \text{if } i = G(a_s), a_s \in A_g \\ 0 & \text{altrimenti} \end{cases}, \qquad i \in [1, n_g] \tag{4.10}$$

dove n_g è il numero di gruppi presenti nella memoria del nodo e G è una funzione che collega ogni stato all'indice del gruppo a cui appartiene, mentre a_s e A_g sono i valori calcolati al passo precedente.

Ogni volta che il valore di attivazione di un gruppo scende a zero (o sotto una soglia) il gruppo viene rimosso dall'insieme dei gruppi attivi A_g.

Ogni volta che l'insieme dei gruppi attivi A_g è vuoto, viene eseguito un "ballottaggio" per scegliere un nuovo insieme dei gruppi attivi. I gruppi che verranno selezionati come attivi sono i gruppi che contengono uno stato mappato sulla corrente coincidenza attiva e un "voto" è assegnato ad ogni gruppo tanto più alto quanto tanto più la posizione dello stato corrispondente è vicina all'inzio della catena. In questa maniera si incrementa la probabilità che un gruppo sia seguito dall'inizio e non da un punto casuale più vicino alla fine, aumentando la stabilità dell'uscita FF_{out}.

Fase di Inferenza - Feedback

Il link di uscita FB_{out} contiene la distribuzione di probabilità rispetto alle coincidenze di ingresso.

Facendo riferimento alla figura 4.14, ogni volta che il *Time Master* richiede al nodo l'aggiornamento del percorso di feedback, esso procede alle seguenti operazioni.

Per aggiornare lo stato interno β viene calcolata l'intersezione tra FF_{out} e FB_{in}. Se non è vuota (ovvero si ha un match) viene scelto un gruppo "vincitore" tra questi stati. Altrimenti, in base ai comandi ricevuti dalla *System Controlling Unit*, il nodo può scegliere un gruppo vincitore solamente dal proprio FF_{out} oppure dal FB_{in}. Nel primo caso viene privilegiato il proprio stato interno mentre nel secondo viene assecondato lo stato dei livelli superiori.

Dopo che il gruppo vincitore w_g è stato selezionato, i valore di attivazione dei suoi stati sono copiati da α a β:

$$\beta(i) \quad = \quad \alpha(i), \qquad i \in [1, n_s] \wedge G(i) = w_g \tag{4.11}$$

dove n_s è il numero degli stati interni alla *TAM* e G è una funzione che collega ogni stato all'indice del gruppo a cui appartiene.

All'interno dello stato *Bel* solamente la coincidenza w_c corrispondente allo stato con il valore di attivazione massimo w_s viene selezionata:

$$w_s \quad = \quad i : \beta(i) = \max_{j \in [1, n_s]} \beta(j) \tag{4.12}$$

$$w_c \quad = \quad C(w_s) \tag{4.13}$$

$$\mathrm{Bel}(w_c) \quad = \quad 1 \tag{4.14}$$

dove n_s è il numero degli stati interni alla *TAM* e C è una funzione che collega ogni stato alla coincidenza corrispondente.

Infine viene aggiornato il link di uscita FB_{out}:

$$\mathrm{FB}_{out} = \mathrm{Bel} \tag{4.15}$$

L'utilizzo del *Forced Temporal Pooler node* porta indubbi vantaggi rispetto ad un ordinario *Temporal Pooler node* all'interno di un *Wired Memory level* (sezione 4.1.4). In particolare possiamo evidenziare come

- l'addestramento sia molto più breve, semplice e controllabile
- l'uscita della fase di inferenza FF_{out} sia molto più stabile e permetta di seguire i gruppi in maniera più continuativa nel tempo
- le prestazioni migliori siano fornite quando utilizzato per riconoscere una primitiva di un segnale all'interno del *Wired Memory level*
- date le caratteristiche dell'uscita del nodo, questo si presenta come un candidato ideale per essere il livello inferiore di un *Event Pooler node* presentato nella prossima sezione.

4.2.5 Event Pooler Node (EP)

Il nodo *Event Pooler* non ha un corrispondente nella implementazione realizzata da Numenta. Il suo obiettivo è imparare sequenze contenenti un numero considerevole di stati ripetuti in maniera più efficiente, utilizzando un quantitativo inferiore di memoria e con caratteristiche migliori di generalizzazione.

Questo nodo è stato progettato per modellare i livelli superiori della rete e per essere utilizzato all'interno di un *Memory Level* normale descritto nella sezione 4.1.5 dove la maggior parte dell'informazione risiede nella componente temporale del segnale rispetto a quella spaziale.

Per raggiungere tale obiettivo viene modificata la precedente definizione di "stato" all'interno del nodo. Dal momento che è plausibile supporre che i livelli inferiori generino una uscita FF_{out} stabile (ovvero che mantengano lo stesso gruppo attivo per un numero considerevole di intervalli) è ragionevole introdurre all'interno dello stato il concetto di "durata" di tale coincidenza. Questo comporta che uno stato è in grado di rappresentare un insieme di intervalli temporali e non solo uno come nei *Temporal Pooler node* (sezione 4.2.3) o *Forced Temporal Pooler node*.

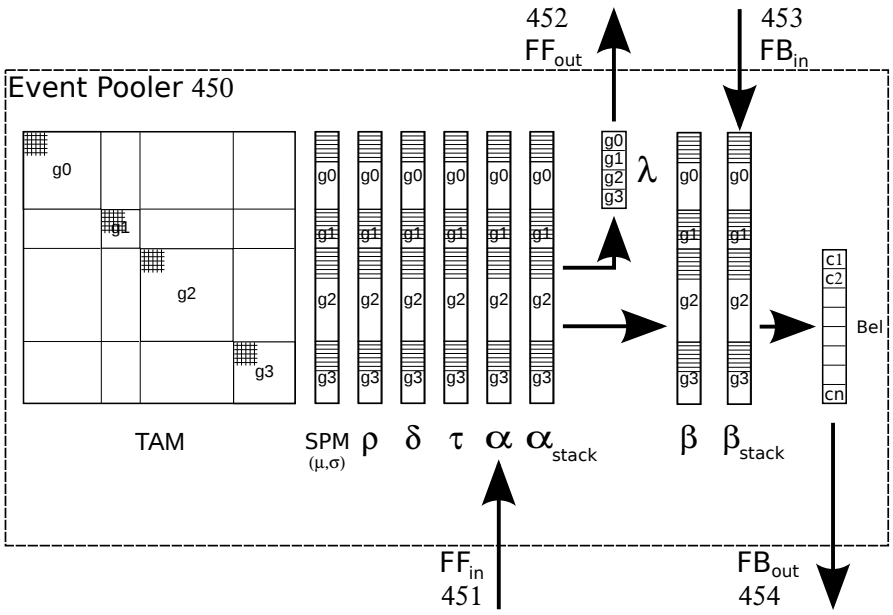


Figura 4.15: Struttura interna di un Event Pooler node.

Se in questi due nodi si potevano considerare i gruppi come delle catene di Markov di primo ordine, i gruppi di questo nodo possono essere modellati come dei semi-Markov Process [45].

Come conseguenza a questa nuova definizione la *TAM* di questo nodo sarà di dimensioni notevolmente inferiori.

Ogni stato interno del nodo può essere considerato come un "evento" che è caratterizzato da una durata media rilevata durante la fase di addestramento. Una nuova struttura dati, *State Permanence Matrix - SPM*, è utilizzata per memorizzare le statistiche di durata per ogni stato del nodo. Durante la fase di inferenza la *TAM* e la *SPM* sono utilizzate congiuntamente per seguire i gruppi.

La raccolta di queste statistiche temporali introduce la possibilità di poter considerare anche eventi a tempo variabile.

Struttura del nodo

Facendo riferimento alla figura 4.15:

- c_i - rappresenta una coincidenza in ingresso
- s_i - rappresenta uno stato interno (vari stati, appartenenti a gruppi differenti, posso essere mappati sulla medesima coincidenza)
- g_i - rappresenta un gruppo composto da vari stati s_j dove ognuno è mappato su una singola coincidenza c_k. Ogni stato appartiene esclusivamente ad un unico gruppo.
- α - contiene lo stato di feedforward corrente
- α_{stack} - è uno *stack* di memoria utilizzato per salvare e ripristinare lo stato α durante le fasi di predizione.
- α_i - contiene il valore di attivazione di feedforward per lo stato s_i
- β - contiene lo stato di feedback corrente

- β_{stack} - è uno *stack* di memoria utilizzato per salvare e ripristinare lo stato β durante le fasi di predizione.
- β_i - contiene il valore di attivazione di feedback per lo stato s_i
- *SPM* - contiene le statiche di permanenza sugli stati raccolte durante la fase di addestramento
- SPM_i - contiene il valore di permanenza medio $\mu : i$ e il relativo valore di deviazione standard σ_i per lo stato s_i
- ρ - contiene la probabilità "coincidence to group"
- ρ_i - rappresenta la probabilità che la coincidenza c_j (sulla quale è mappato lo stato s_i) faccia riferimento al gruppo g_r (che contiene lo stato s_i) quando viene vista come ingresso. Ovvero rappresenta la probabilità che g_r sia il gruppo corretto da attivare quando la coincidenza c_j è ricevuta in ingresso.
- τ - contiene la "state permanence probability"
- τ_i - modella come una distribuzione Gaussiana la probabilità di rimanere nello stato s_i. È calcolata ed aggiornata utilizzando le statistiche μ_i e σ_i salvate all'interno di SPM_i e dal numero di time-step trascorsi da quando la corrispondente coincidenza c_j è stata inizialmente rilevata all'ingresso. Tale valore è impostato a 0 quando c_j non è presente in ingresso.
- δ - contiene il termine di decadimento dello stato
- δ_i - rappresenta il termine di decadimento associato alla permanenza sullo stato s_i.

Fase di Addestramento

A differenza di quanto avviene per l'addestramento di un nodo *Temporal Pooler* (sezione 4.2.3), le transizioni tra stati della *TAM* non sono aggiornate ad ogni nuovo ingresso. Le transizioni sono riportate nella *TAM* solamente quando viene rilevata

una modifica sostanziale all'ingresso del nodo. In altre parole, una nuova transizione è considerata solo quando l'inizio di un nuovo evento è indicato da un cambio nel FF_{in}.

Al contrario, quando non viene rilevata nessuna modifica sostanziale nel FF_{in}, vengono solamente raccolte le statistiche di permanenza sullo stato all'interno della *SPM*.

Il meccanismo di "splitting" degli stati è utilizzato con lo stesso scopo del *Temporal Pooler node*. In questo caso è possibile anche pensare ad una variante maggiormente "aggressiva" di tale algoritmo poiché è ragionevole pensare che verranno formate catene più semplici. Inoltre è utilizzabile una versione che replichi gli stati in base alle statistiche di permanenza raccolte dalla *SPM*.

Riconoscere un cambiamento

Si può supporre che l'ingresso sia composto da un insieme di gruppi attivi allo stesso tempo, rappresentanti le possibili cause secondo i livelli inferiori. Tale situazione è denominata "multiple activations" e viene in qualche modo "ricercata" dai nodi *Forced Temporal pooler* descritti nella sezione 4.2.4.

Osservando le sequenze di attivazioni multiple che il nodo riceve, è possibile notare che frequentemente alcuni gruppi persistono per un numero di intervalli considerevole mentre altri, durante lo stesso intervallo, appaiono e scompaiono. Perciò è possibile pensare di filtrare sequenze di attivazioni multiple per estrarre solo gli stati vincenti persistenti, ovvero quelli che più probabilmente stanno descrivendo la sequenza.

Solamente quanto tutti gli stati persistenti in attivazione multipla scompaiono sarà possibile supporre di aver raggiunto il termine di un "evento" e perciò segnare una transizione nella *TAM*.

Questo comportamento può essere ottenuto durante la fase di addestramento del nodo mantenendo memoria dell'intersezione tra attivazioni multiple consecutive e segnalare un cambio di evento solamente quando l'intersezione risulti vuota.

La tabella 4.1 mostra un esempio di attivazioni multiple ricevute dal FF_{in} di un *Event Pooler node* durante la fase di addestramento. La prima colonna mostra il *time-*

time	FF activations	intersection	signaled event	n_{rep}	transitions to mark
32	...	...			
33	1 3 4 7	1 3 4 7	x	y	$z \rightarrow x$
34	1 3 5 7 8	1 3 7			
35	1 3 5 7	1 3 7			
36	1 3 4 9	1 3			
37	1 3	1 3			
38	1 3	1 3			
39	3 5	3			
40	3	3			
41	2 6 10	2 6 10	3	8	$x \rightarrow 3$
42	2 6 10	2 6 10			
43	2 10 11	2 10			
44	2 11	2			
45	2 11	2			
46	2	2			
47	2	2			
48	4	4	2	7	$3 \rightarrow 2$
...	...	...			

Tabella 4.1: Esempio di attivazioni multiple da parte di un nodo figlio e la corrispondente interpretazione durante la fase di addestramento

step corrente, la seconda mostra un insieme di gruppi vincitori segnalati dal livello di memoria sottostante tramite il suo FF_{out}. La colonna "intersection" mostra il contenuto corrente della memoria delle intersezioni, che è mantenuta aggiornata ad ogni passo. Ogni qualvolta tale memoria risulti vuota (ad esempio al passo 33,41 e 48) un evento di "cambio" è segnalato e l'intersezione stessa è reinizializzata all'ingresso corrente e il numero di ripetizioni azzerato. Al contrario, finché l'intersezione contiene almeno una coincidenza il contatore delle ripetizioni è incrementato (colonna n_{rep}). Infine l'ultima colonna ("transitions to mark") mostra solamente le transizioni che saranno inserite nella *TAM*; è possibile notare come esse siano in numero decisamente inferiore rispetto al numero di time-step considerati.

Al termine della fase di accumulazione delle statistiche è molto probabile che le catene formate siano più semplici rispetto a quelle che avrebbe formato il *Temporal-Pooler*; perciò altrimenti un algoritmo più semplice.

L'ultimo passaggio riguarda la finalizzazione della *State Permanence Matrix*, ovvero il calcolo della permanenza media di ogni stato e la relativa deviazione standard. Tali informazioni saranno impiegate nella fase di inferenza.

A questo punto ogni stato è codificato utilizzando una tripletta che definisce:

1. coincidenza di ingresso corrispondente
2. numero medio di ripetizioni consecutive (permanenza media)
3. deviazione standard della permanenza.

Si può notare come sia ancora possibile impiegare la strategia di addestramento tramite "marker" descritta nella sezione 4.2.3 anche per questo tipo di nodo.

Esempio di addestramento di un *Event Pooler* node

Prima di analizzare la fase di inferenza può essere utile approfondire come avviene la fase di apprendimento con un esempio che metta a confronto l'algoritmo di apprendimento di un normale *Temporal Pooler node* e quelli di un *Event Pooler node*.

Sia data la seguente sequenza di ingresso come unica sequenza all'interno dei dati di addestramento:

$$1\ 1\ 1\ 1\ 2\ 2\ 2\ 3\ 3\ 3\ 3\ 3\ 3\ 4\ 4$$

e sia presentata ad un normale *Temporal Pooler node*. Dopo la fase di "splitting" la sequenza preposta all'apprendimento diventa:

$$1 \to 1 \to 1 \to 1 \to 2 \to 2 \to 2 \to 3 \to 3 \to 3 \to 3 \to 3 \to 3 \to 4 \to 4$$

Al contrario, se la medesima sequenza viene presentata ad un *Event Pooler node*, non ci sarà bisogno di effettuare alcuna fase di "splitting" (in questo specifico caso) e la sequenza diventerà:

$$(1,\mathbf{4},0) \to (2,\mathbf{3},0) \to (3,\mathbf{5},0) \to (4,\mathbf{2},0)$$

Ogni coppia di parentesi rappresenta uno stato interno del nodo dove il primo numero rappresenta la coincidenza su quale lo stato è mappato, il secondo numero (in grassetto) rappresenta la media delle ripetizioni consecutive osservate durante la fase di addestramento per quella coincidenza e il terzo numero (in corsivo) specifica la deviazione standard (in questo caso specifico è sempre pari a 0 poiché vi è solo una sequenza nel set di addestramento).

Se al contrario forniamo come dati di addestramento:

$$\begin{gathered}\dots\\ 1\ 1\ 1\ 2\ 2\ 2\ 2\ 2\ 4\ 4\\ \dots\\ 1\ 1\ 1\ 1\ 1\ 1\ 2\ 2\ 4\ 4\\ \dots\end{gathered} \tag{4.16}$$

il nodo *Event Pooler* apprende:

$$(1, \mathbf{4.5}, \mathit{1.5}) \rightarrow (2, \mathbf{3.5}, \mathit{1.5}) \rightarrow (4, \mathbf{2}, \mathit{1})$$

dove si può notare che la varianza di alcuni stati è maggiore di uno.

Fase di Inferenza - Feedforward

L'uscita FF_{out} contiene la distribuzione di probabilità rispetto ai gruppi del nodo.

Facendo riferimento alla figura 4.15, ogni volta che il *Time Master* richiede al nodo l'aggiornamento del percorso di feedforward sono eseguiti i seguenti calcoli.

Due differenti comportamenti sono utilizzati per aggiornare lo stato di feedforward α a seconda dell'ingresso corrente:

1. quando viene rilevato un cambio negli ingressi (come presentato precedentemente), il nodo *Event Pooler* aggiorna il suo stato interno utilizzando le medesime equazioni utilizzate dal normale *Temporal Pooler node* descritte nel paragrafo 4.2.3, perciò muovendo di un passo in avanti tutte le catene interne.

$$\alpha_{tmp} = \text{TAM}\ \alpha \tag{4.17}$$

$$\alpha(i) = \alpha_{tmp}(i)\ \text{FF}_{in}(C(i)), \qquad i \in [1, n_s] \tag{4.18}$$

   dove n_s è il numero dello stato interno della *TAM* e C è una funzione che collega ogni stato sull'indice di coincidenza corrispondente

2. al contrario, quando vi è una persistenza dell'ingresso, vengono utilizzate le statistiche di permanenza raccolte durante la fase di addestramento per definire il comportamento degli stati correntemente attivi:

$$\delta_i = \begin{cases} 1 & \text{if } n_{rep} < \mu_i \\ e^{-\frac{1}{2}(\mu_i - n_{rep})^2} & \text{otherwise} \end{cases} \tag{4.19}$$

$$\tau_i = (1 - \rho_i) * \text{DecaySpeed} \tag{4.20}$$

$$\alpha_{tmp} = \alpha \tag{4.21}$$

$$\alpha(i) = \alpha_{tmp}(i)[\text{FF}_{in}(C(i)) + \delta_i] - \tau_i \tag{4.22}$$

dove $i \in A_s$, A_s contiene lo stato maggiormente attivo (*winner state* preso dai gruppi correntemente attivi, n_{rep} è il numero di passi trascorsi da quando questa coincidenza è comparsa, μ_i e σ_i sono le statistiche estratte da SPM_i ed associate allo stato s_i, $DecaySpeed$ è il fattore di decadimento e C è la funzione che collega ogni stato all'indice della coincidenza corrispondente.

Attualmente $DecaySpeed$ è un parametro globale determinato dalla durata degli eventi ricercati ad un determinato livello ma sarà possibile, in futuro, determinare un algoritmo per la sua individuazione automatica durante la fase di addestramento.

Al termine di questa procedura il valore di attivazione per ogni gruppo attivo viene calcolato come il massimo valore di attivazione tra gli stati appartenenti a tale gruppo come:

$$\lambda(i) = \max_{j \in g_i} \alpha(j), \qquad i \in [1, n_g] \tag{4.23}$$

dove n_g è il numero dei gruppi imparati durante la fase di addestramento. Infine viene aggiornata l'uscita di feedforward del nodo tramite:

$$\mathrm{FF}_{out} = \lambda \tag{4.24}$$

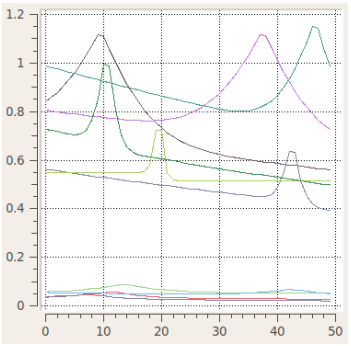


Figura 4.16: Esempio di un possibile trend dei valori di attivazioni per i gruppi che condividono la stessa coincidenza mentre essa è presente stabilmente in ingresso.

La figura 4.16 mostra un esempio di evoluzione nel tempo dei valori di attivazione dei gruppi quando una medesima coincidenza è persistente come input. Tale figura mostra l'effetto del parametro *DecaySpeed*.

Fase di inferenza - Feedback

L'uscita FB_{out} contiene la distribuzione di probabilità rispetto alle coincidenze in ingresso.

Facendo riferimento alla figura 4.15, ogni volta che il *Time Master* richiede al nodo l'aggiornamento del percorso di feedback, esso procede alle seguenti operazioni per aggiornare il proprio stato interno β.

$$\beta(i) \quad = \quad \alpha(i)\,\mathrm{FB}_{in}(G(i)), \qquad i \in [1, n_s] \tag{4.25}$$

dove n_s è il numero degli stati interni alla *TAM* e G è la funzione che mappa ogni stato all'indice del gruppo a cui appartiene.

A questo punto il valore di attivazione di ogni coincidenza è calcolato considerando il massimo valore di tutti gli stati interni della *TAM* che sono mappati su quella coincidenza:

$$Bel(i) = \max_{j \in [1,n_s] \wedge G(j)=i} \beta(j), \qquad i \in [1, n_c] \tag{4.26}$$

dove n_c è il numero di coincidenze in ingresso (pari alla dimensione del link di ingresso FF_{in}.

Infine viene aggiornata l'uscita di feedback del nodo:

$$\mathrm{FF}_{out} = Bel \tag{4.27}$$

4.2.6 Forced Event Pooler Node (FEP)

Il nodo *Forced Event Pooler* è una estensione del nodo *Event pooler*, descritto nel paragrafo precedente, in una maniera simile a come il nodo *Forced Temporal Pooler* estende il nodo *Temporal Pooler*.

Per maggiori dettagli è possibile fare riferimento al paragrafo 4.2.4.

4.2.7 Smoother Node (Sm)

Lo scopo di questo nodo (non presente nella implementazione Numenta) è quello di filtrare i segnali di uscita di feedback di un *Signal Quantizer node* o di uno *Spatial Pooler node* posti al livello più basso della rete. Tali segnali corrispondono a valori nello spazio degli ingressi del sistema. Questo nodo permette di ricostruire una uscita della rete meno rumorosa e non affetta dai classici artefatti dovuti al processo di quantizzazione; perciò permette di confrontare in maniera diretta i segnali ricostruiti o predetti dalla rete con gli effettivi valori provenienti dai sensori.

La figura 4.17 rappresenta un possibile esempio applicativo di tale nodo.

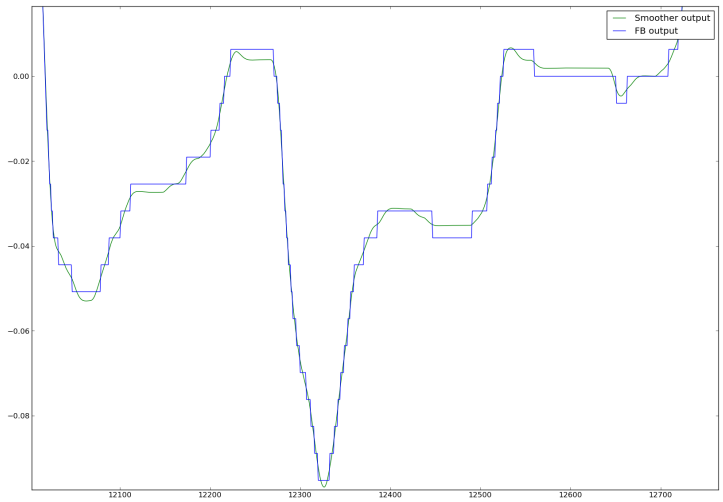


Figura 4.17: Esempio dell'uscita di un Smoother node che analizza un segnale quantizzato.

4.3 Procedimento di addestramento

In questa ultima parte verrà trattato la procedura per addestrare una rete costituita dai nuovi algoritmi, in particolare l'estrazione delle primitive dal dataset e la formazione delle memorie ai livelli più alti della rete.

La rete trattata si compone di un *Wired Memory level* a livello inferiore e da un *Memory level* a livello superiore. In particolare analizzeremo la formazione delle memorie nel *Wired Memory level* mentre valuteremo solamente i risultati della predizione effettuata tramite il *Memory level*.

4.3.1 Estrazione delle primitive

Ogni colonna del *Wired Memory level* corrisponde ad un ingresso monodimensionale del sistema. Per ogni ingresso è necessario estrarre le primitive da dataset; figura 4.18 rappresenta una procedura proposta per la loro estrazione.

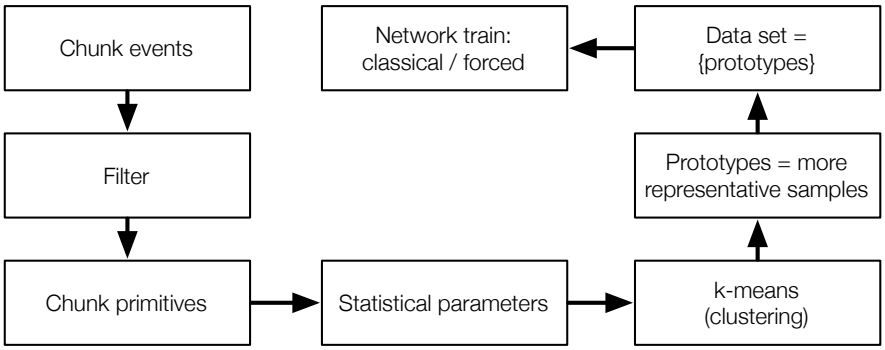


Figura 4.18: Procedura per l'estrazione di primitive da un dataset.

- *Chunk events*: segmenta il dataset nelle più piccole componenti che *Forced Temporal Pooler* dovrà imparare. Ad esempio è possibile selezionare i punti in cui la derivata prima si annulla come in figura 4.19.
- *Filter*: processo opzionale di filtraggio passa basso per eliminare il rumore dal segnale

- *Chunck Primitives*: creazione di un insieme di tutte le primitive
- *Statistical parameters*: per ogni primitiva nell'insieme vengono calcolati un insieme di descrittori (es: durata, valore massimo o minimo, pendenza, ...)
- *clustering*: vengono raggruppate le primitive a seconda dei descrittori grazie ad un algoritmo di clustering (K-Means, Hierarchical Clustering, ...)
- *Estrazione prototipi*: per ogni cluster viene estratta la primitiva più rappresentativa e viene eletta a "prototipo", come mostrato in figura 4.20
- *Dataset*: il dataset viene creano unendo tutti i prototipi precedentemente estratti. Se presente anche un *Event Pooler node* viene creato il file di addestramento contenente i dati originali eventualmente filtrati.
- *Addestramento*: il dataset creato viene utilizzato per addestrare i livelli della rete.

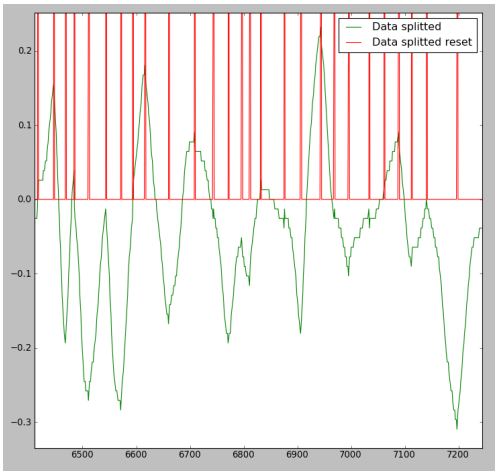


Figura 4.19: Primitive su un segnale monodimensionale

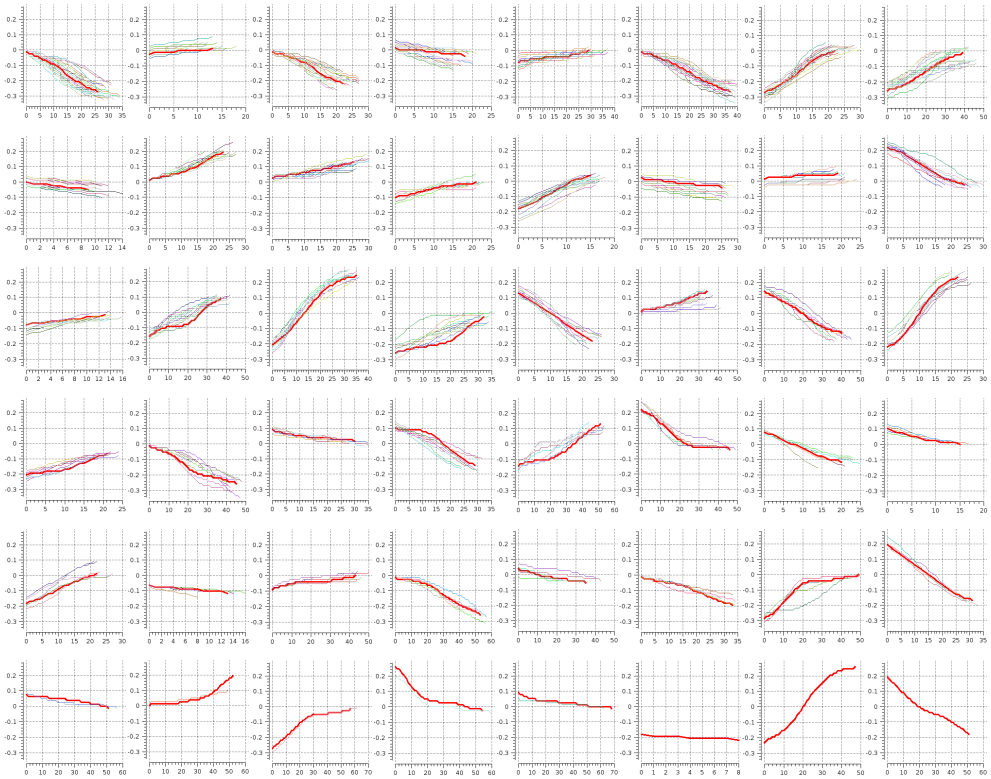


Figura 4.20: Elenco dei gruppi di primitive e relativi prototipi

4.3.2 Ispezione della memoria

Grazie all'apprendimento forzato i prototipi verranno memorizzati dal *Forced Temporal Pooler node* in maniera esatta. Al contrario l'addestramento del nodo *Event Pooler node* si basa sull'uscita di feedforward del *Forced Temporal Pooler node*, inoltre i dati utilizzati per addestrare il nodo *EP* non sono primitive o prototipi ma i dati originali o un sottoinsieme di essi perciò è necessario che la memoria del nodo *FTP* sia in grado di rappresentare il segnale.

Utilizzando una speciale modalità del *Time Master*, denominata *unfolding*, è possibile ispezionare la memoria di un nodo a qualsiasi livello della rete rappresentandola nello spazio degli ingressi. Tale modalità sfrutta i collegamenti di feedback presenti all'interno della rete. In particolare vengono tradotti i singoli stati del nodo sotto ispezione livello per livello discendendo lungo la gerarchia fino ad arrivare all'uscita del nodo al livello più basso. La figura 4.21 rappresenta quattro differenti gruppi memorizzati dal nodo *EP*. É possibile notare come siano composte da alcuni prototipi individuati precedentemente e mostrati in figura 4.20.

4.3.3 Predizione di segnali multidimensionali

L'ultimo test presentato riguarda la predizione dei valori futuri di un segnale sfruttando l'intera gerarchia di memoria.

Per poter predire con pochi campioni di anticipo i futuri ingressi al sistema potrebbe essere sufficiente l'approccio primitive/prototipi appena presentato poiché, una volta riconosciuto correttamente l'evento in corso il sistema è in grado di generare i futuri ingressi partendo dalle sequenze memorizzate.

Nonostante ciò alcuni problemi restano aperti:

1. al termine di una catena non si ha alcuna indicazione su quale sarà l'ingresso futuro

2. le catene memorizzate sono brevi poiché descrivono componenti statistiche del segnale

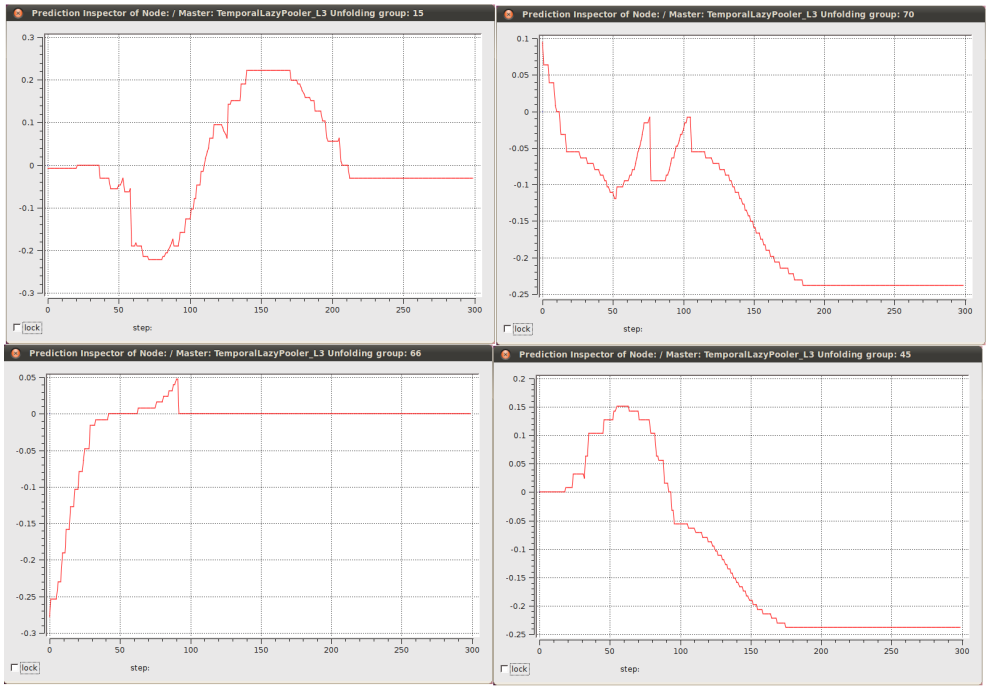


Figura 4.21: Rappresentazione della memoria di un *Event Pooler node*

3. ogni segnale è trattato indipendentemente dal *Wired Memory level* e non è possibile ricercare co-occorrenze tra di essi

Il primo problema è, probabilmente, un limite intrinseco di una architettura basata sul riconoscimento di sequenze. Una possibile soluzione risiede nell'aumentare la rete di un ulteriore *Memory Level* in maniera che possa imparare sequenze della lunghezza desiderata

Al contrario, il secondo ed il terzo problema possono essere risolti tramite l'introduzione di un *Memory Level* al di sopra del *Wired Memory Level*. Tale livello ha il compito di ricevere in ingresso le uscite dei nodi del primo livello e ricercare nelle sue uscite nuove sequenze. In particolare non è necessario che sia presente una colonna per ogni ingresso dal livello sottostante; al contrario, unendo più uscite del *Wired Memory level* in singole colonne del *Memory Level* sarà possibile riscontrare delle co-occorrenze tra differenti segnali. Ovviamente i differenti segnali forniti al sistema devono essere correlati temporalmente tra loro.

Tale strategia porta alla creazione di memorie non più direttamente rapportabili con un singolo segnale di ingresso ma piuttosto rappresentano l'evoluzione congiunta dei segnali. Sfruttando tale rappresentazione è possibile ottenere vari vantaggi:

- nel caso uno dei segnali di ingresso sia compromesso o rumoroso ma gli altri segnali siano sufficienti a creare il corretto stato all'interno della colonna, è possibile utilizzare le informazioni di feedback sul segnale affetto da problemi per ricostruirlo o filtrarlo dal rumore eccessivo;

- sfruttando la presenza di memorie a lungo termine su più canali è possibile prevedere quale sequenza di basso livello sarà attivata su uno specifico canale anche prima che su di esso si presenti l'inizio.

In figura 4.22 è mostrata l'uscita di predizione a 10 campioni in avanti rispetto ad un segnale monodimensionale facente parte di un insieme di più segnali analizzati da un sistema di memoria artificiale costituito da un *Wired Memory Level* e da un *Memory Level* normale. Per ogni istante di tempo vengono predetti i 10 campioni futuri e viene riportato in figura solamente l'ultimo dei 10 passi avanti. Al termine

della procedura è così possibile confrontare direttamente il valore predetto col valore successivamente rilevato. È possibile osservare come, per i due grafici in alto, il sistema sia stato in grado di predire correttamente l'inizio dell'evento senza che sul segnale stesso vi sia stata alcuna informazione, grazie alla struttura gerarchica e multidimensionale della memoria. Al contrario, nel grafico inferiore è possibile notare come la predizione non abbia avuto successo nell'individuare l'esatto momento di inizio dell'evento; ciò può essere dovuto all'incompletezza delle memorie formate o dalla insufficiente informazione negli altri canali percettivi per generare lo stato corretto ai livelli superiori.

Infine è possibile osservare come anche durante lo svolgimento dell'evento il segnale predetto ricalchi il segnale reale successivamente riscontrato.

Questi risultati non sono raggiungibili con l'implementazione delle Hiearchical Temporal Memory creata da Numenta poiché non sono presenti tutti quei meccanismi atti a gestire sequenze lunghe, temporizzare l'esecuzione dei nodi nella rete, trasferire informazione tramite collegamenti di feedback e gestire in maniera efficiente un insieme di segnali monodimensionali.

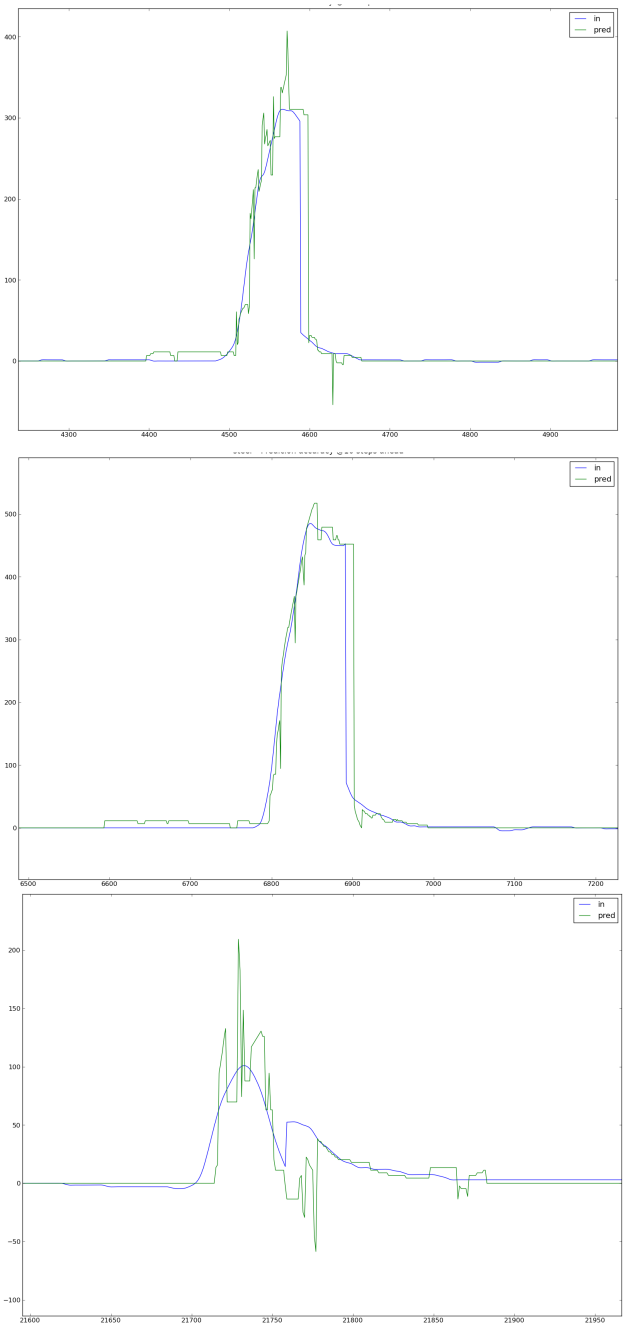


Figura 4.22: Predizione a 10 campioni in avanti su una memoria a più livelli

Conclusioni

In questo lavoro di tesi sono stati affrontati vari aspetti riguardanti l'acquisizione e l'analisi di insiemi di dati monodimensionali provenienti da vari sensori per individuare *pattern* spazio-temporali ricorrenti.

In particolare sono state presentate le Hierarchical Temporal Memories, un nuovo paradigma computazionale che si ispira al funzionamento della neocorteccia umana, il cui obiettivo è l'individuazione di sequenze spazio-temporali all'interno di un dataset. Tale paradigma è stato proposto solo di recente ma sta suscitando un grande interesse.

Questa architettura è stata successivamente impiegata per riconoscere movimenti ed azioni compiuti dalle persone. I sensori utilizzati in questo lavoro si compongono di un accelerometro triassale indossabile e da ambiente sorvegliato da varie telecamere. Dopo una fase di estrazione dei dati di movimento dai sensori, sono state impiegate le Hierarchical Temporal Memories per individuare dall'insieme dei dati raccolti le sequenze di valori rappresentanti le componenti *invarianti* dei vari movimenti compiuti. Data la duttilità delle HTM è stato possibile utilizzarle sia per analizzare i dati provenienti dalla telecamera sia per quelli provenienti dall'accelerometro.

Una ulteriore componente di questo lavoro di tesi ha riguardato la concezione di un protocollo di *routing* ottimizzato per Wireless Sensor Networks di monitoraggio. Successivamente è stata utilizzata una WSN per la raccolta di parametri ambientali (temperatura ed umidità) su un lungo periodo di tempo (circa un anno). Tale protocollo è in attesa di rilascio di un brevetto ed è attualmente utilizzato da Henesis s.r.l. per la realizzazione di reti di monitoraggio.

L'esperienza di utilizzo delle Hierarchical Temporal Memories ha portato alla luce alcuni limiti dell'implementazione utilizzata. Il lavoro di tesi si è quindi concentrato sulla progettazione ed implementazione di una nuova architettura che estende le HTM sia dal punto di vista algoritmico sia dal punto di vista delle funzionalità tramite l'aggiunta di componenti esterni. Questa nuova architettura si è dimostrata in grado di apprendere insiemi di segnali monodimensionali estraendone le componenti invarianti durante la fase di addestramento e di essere in grado di predire con buona accuratezza segnali multidimensionali. La realizzazione di questa nuova architettura ha portato alla sottomissione di brevetto internazionale.

Gli sviluppi futuri di questa tesi possono essere molteplici, in particolare ritengo particolarmente interessante continuare il lavoro svolto su questa nuova architettura di memoria artificiale. Nel prossimo futuro saranno eseguiti ulteriori test su differenti dataset e saranno ricercati nuovi algoritmi per migliorare ulteriormente le prestazioni in fase di predizione.

Bibliografia

[1] Jeff Hawkins and Sandra Blakeslee. *On Intelligence*. Times Books, October 2004.

[2] V. Mountcastle. An organizing principle for cerebral function: the unit model and the distributed system. In G. Edelman and V. Mountcastle, editors, *The Mindful Brain*. MIT Press, 1978.

[3] Jaros, B. and George, D. The HTM learning algorithms. Technical report, Numenta Inc., 2007.

[4] Dileep George and Jeff Hawkins. Towards a mathematical theory of cortical micro-circuits. *PLoS Comput Biol*, 5(10), 10 2009.

[5] A.B. Csapo, P. Baranyi, and D. Tikk. Object categorization using VFA-generated nodemaps and Hierarchical Temporal Memories. In *IEEE International Conference on Computational Cybernetics, ICCC 2007*, pages 257–262, oct. 2007.

[6] N. Farahmand, M.H. Dezfoulian, H. GhiasiRad, A. Mokhtari, and A. Nouri. Online temporal pattern learning. In *Proceedings of International Joint Conference on Neural Networks, IJCNN 2009*, pages 797–802, june 2009.

[7] Joost van Doremalen and Boven Lou. Spoken digit recognition using a hierarchical temporal memory. In *Proceedings of Interspeech 2008, Brisbane, Australia*, pages 2566–2569, September 2008.

[8] S. Zhang, M.H. Ang, W. Xiao, and C.K. Tham. Detection of activities for daily life surveillance: eating and drinking. In *10th International Conference on e-health Networking, Applications and Services, HealthCom 2008*, pages 171–176, july 2008.

[9] Y. C. Ho and Pepyne D. L. Explanation of the No-Free-Lunch Theorem and Its Implications. *Journal of Optimization Theory and Applications*, pages 549–570, 2002.

[10] Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988.

[11] Kevin Patrick Murphy. Dynamic bayesian networks: Representation, inference and learning, 2002.

[12] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[13] Federico Sassi, Luca Ascari, and Stefano Cagnoni. Classifying human body acceleration patterns using a hierarchical temporal memory. In *Proceedings of the XIth International Conference of the Italian Association for Artificial Intelligence Reggio Emilia on Emergent Perspectives in Artificial Intelligence*, AI*IA '09:, pages 496–505. Springer-Verlag, 2009.

[14] Brian D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[15] Maximilian Riesenhuber and Tomaso Poggio. Hierarchical models of object recognition in cortex. *Nature Neuroscience*, 2:1019–1025, 1999.

[16] Dorothy Monekosso, Paolo Remagnino, and Yoshinori Kuno, editors. *Intelligent Environments - Methods, Algorithms and Applications*. Springer, 2009.

[17] Xiaofei Ji and Honghai Liu. Advances in view-invariant human motion analysis: a review. *IEEE Transaction Systems, Man and Cybernetics Part C*, 40:13–24, January 2010.

[18] P. Turaga, R. Chellappa, V. S. Subrahmanian, and O. Udrea. Machine recognition of human activities: a survey. *IEEE Transactions on Circuits and Systems for Video Technology*, 18(11):1473–1488, September 2008.

[19] Nicky Kern, Bernt Schiele, and Albrecht Schmidt. Multi-sensor activity context detection for wearable computing. In *Proc. EUSAI*, pages 220–232, 2003.

[20] A.M. Khan, Young-Koo Lee, S.Y. Lee, and Tae-Seong Kim. A triaxial accelerometer-based physical-activity recognition via augmented-signal features and a hierarchical recognizer. *IEEE Transactions on Information Technology in Biomedicine*, 14(5):1166–1172, september 2010.

[21] Chun Zhu and Weihua Sheng. Multi-sensor fusion for human daily activity recognition in robot-assisted living. In *Proceedings of the 4th ACM/IEEE International Conference on Human Robot Interaction*, HRI '09, pages 303–304, New York, NY, USA, 2009. ACM.

[22] Jamie A. Ward, Paul Lukowicz, Gerhard Troster, and Thad E. Starner. Activity recognition of assembly tasks using body-worn microphones and accelerometers. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28:1553–1567, October 2006.

[23] U. Rutishauser, J. Joller, and R. Douglas. Control and learning of ambience by an intelligent building. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 35(1):121–132, jan. 2005.

[24] Jonathan Lester, Tanzeem Choudhury, Nicky Kern, Gaetano Borriello, and Blake Hannaford. A hybrid discriminative/generative approach for modeling human activities. In *Proc. International Joint Conference on Artificial Intelligence, IJCAI*, pages 766–772, 2005.

[25] A. Leone, G. Diraco, C. Distante, P. Siciliano, M. Malfatti, L. Gonzo, M. Grassi, A. Lombardi, G. Rescio, P. Malcovati, V. Libal, J. Huang, and G. Potamianos. A multi-sensor approach for people fall detection in home environment. In Andrea Cavallaro and Hamid Aghajan, editors, *Workshop on Multi-camera and Multi-modal Sensor Fusion Algorithms and Applications, M2SFA2 2008*, Marseille France, 2008.

[26] P. Rashidi, D.J. Cook, L.B. Holder, and M. Schmitter-Edgecombe. Discovering activities to recognize and track in a smart environment. *IEEE Transactions on Knowledge and Data Engineering*, 23(4):527–539, april 2011.

[27] C. W. Hsu, C. C. Chang, and C. J. Lin. A practical guide to support vector classification. Technical report, Department of Computer Science, National Taiwan University, Taipei, 2003.

[28] Vili Kellokumpu, Matti Pietikäinen, and Janne Heikkilä. Human activity recognition using sequences of postures. In *Proc. IAPR Conference on Machine Vision Applications, MVA 2005*, pages 570–573, 2005.

[29] Youngwook Kim and Hao Ling. Human activity classification based on micro-Doppler signatures using a support vector machine. *IEEE Transactions on Geoscience and Remote Sensing*, 47(5):1328–1337, May 2009.

[30] Juan Carlos Niebles and Li Fei-Fei. A hierarchical model of shape and appearance for human action classification. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1–8, Los Alamitos, CA, USA, 2007. IEEE Computer Society.

[31] C. Schuldt, I. Laptev, and B. Caputo. Recognizing human actions: a local SVM approach. In *Proceedings of the 17th International Conference on Pattern Recognition, ICPR 2004.*, volume 3, pages 32–36, August 2004.

[32] Xinyu Wu, Yongsheng Ou, Huihuan Qian, and Yangsheng Xu. A detection system for human abnormal behavior. In *IEEE/RSJInternational Conference on Intelligent Robots and Systems, IROS 2005*, pages 1204–1208, August 2005.

[33] A. Godfrey, R. Conway, D. Meagher, and G. ÓLaighin. Direct measurement of human movement by accelerometry. *Medical Engineering and Physics*, 30(10):1364 – 1386, 2008. Special issue to commemorate the 30th anniversary of Medical Engineering and Physics.

[34] Roberto Ugolotti, Federico Sassi, Monica Mordonini, and Stefano Cagnoni. Multi-sensor system for detection and classification of human activities. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–15, 2011.

[35] R. Cucchiara, C. Grana, M. Piccardi, A. Prati, and S. Sirotti. Improving shadow suppression in moving object detection with HSV color information. In *Proceedings of IEEE Intelligent Transportation Systems, 2001*, pages 334–339, 2001.

[36] Jianbo Shi and Carlo Tomasi. Good Features to Track. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR94)*, pages 593–600, 1994.

[37] Ralf Grossmann, Jan Blumenthal, Frank Golatowski, and Dirk Timmermann. Localization in Zigbee-based sensor networks, 2007.

[38] Marc Lihan, Takeshi Tsuchiya, and Keiichi Koyanagi. Orientation-aware indoor localization path loss prediction model for wireless sensor networks. In *Network-Based Information Systems*, pages 169–178. Springer, 2008.

[39] R.C. Luo, O. Chen, and Cheng Wei Lin. Indoor human monitoring system using wireless and pyroelectric sensory fusion system. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2010*, pages 1507–1512, oct. 2010.

[40] J.A. Gutierrez, M. Naeve, E. Callaway, M. Bourgeois, V. Mitter, and B. Heile. Ieee 802.15.4: a developing standard for low-power low-cost wireless personal area networks. *Network, IEEE*, 15(5):12 –19, sep/oct 2001.

[41] I. Howitt and J.A. Gutierrez. Ieee 802.15.4 low rate - wireless personal area network coexistence issues. In *Wireless Communications and Networking, 2003. WCNC 2003. 2003 IEEE*, volume 3, pages 1481 –1486 vol.3, march 2003.

[42] IEEE Std. 802.15.4. Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs), October 2003.

[43] Zigbee Alliance. Zigbee alliance [online]. 2008. Available from: `http://www.zigbee.org/` [cited 2011].

[44] Luca Ascari, Federico Sassi, Matteo Sacchi, Luca Mussi, Jonas Ambeck-Madsen, Ichiro Sakata and Hiromichi Yanagihara. Artificial memory system and method for use with a computational machine for interacting with dynamic behaviours - GB patent application No. 1111645.6 US patent application No. 61/505,278, July 2011.

[45] Mode, Charles J. *Semi-Markov Processes*. John Wiley and Sons, Ltd, 2005.

Ringraziamenti

Prima di tutto desidero ringraziare il mio tutor, il prof. Stefano Cagnoni. Ho avuto il piacere, e l'onore, di iniziare a conoscerlo fin dalla laurea "triennale" passando dalla laurea "magistrale" fino alla scuola di dottorato. Durante questi anni non ha mai smesso di essere uno stimolo alla scoperta di nuovi strumenti, punti di vista e possibilità senza far mai mancare il suo prezioso consiglio. Inoltre grazie alla sua infinita pazienza ed apertura alle idee più strane sono riuscito a vivere questa esperienza al meglio.

Ringrazio tutti i miei colleghi che sono transitati in questi anni dall'IBISlab, Monica, Luca, Luca, Roberto, Pablo, Youssef, per aver accettato le mie incursioni in laboratorio ed aver ascoltato divagazioni su strani progetti.

Inoltre vorrei ringraziare tutti i miei colleghi di Henesis per le tante discussioni e progetti realizzati insieme. In particolare vorrei ringraziare Luca Ascari per avermi fatto scoprire le Hierarchical Temporal Memories ed anche Luca Mussi e Matteo Sacchi per il lavoro svolto insieme su di esse che ha concorso alla realizzazione di questo lavoro di tesi. Ringrazio anche Jonas che ci ha saputo spronare come solo lui è in grado di fare. Inoltre vorrei ringraziare Lorenzo Chiesi per le lunghe discussioni e la fondamentale collaborazione nella definizione del protocollo di comunicazione per le reti di sensori e per poi averlo reso realtà.

Devo ringraziare anche tutti i co-autori degli articoli realizzati durante questi anni, poiché è anche merito loro se sono giunto a questa tesi. Sono inoltre grato agli studenti che ho avuto il piacere di accompagnare lungo il cammino di tesi, Roberto, Naldo, Giusy, Salvatore, Alessandro e Rita.

Ringrazio anche tutti i miei amici, vicini e lontani, nuovi e vecchi, che hanno sempre saputo quando era il momento di tirarmi fuori e farmi fare il pieno di nuova energia. Grazie infinite!

Infine ringrazio anche la mia "vecchia" famiglia, mia madre, mia sorella e mia zia, e anche chi non c'è più, per il solido supporto in tutti questi anni frenetici.

Dedico questa tesi ad Ilaria e alla nostra nuova avventura insieme. Lei merita senza ombra di dubbio questo riconoscimento per la comprensione, pazienza ed incoraggiamento mostrati in tutti questi anni, soprattutto quando la via era più stretta. Insieme a te la strada non è mai troppo lunga (e nemmeno noiosa, anzi)!