ALMA MATER STUDIORUM – UNIVERSITÀ DI BOLOGNA
CAMPUS DI CESENA

Scuola di Scienze
Corso di Laurea in Ingegneria e Scienze Informatiche

RICONOSCIMENTO REAL-TIME DI GESTURE TRAMITE TECNICHE DI MACHINE LEARNING

Relazione finale in
ARCHITETTURE DEGLI ELABORATORI

Relatore
Prof. DAVIDE MALTONI

Co-relatore
Prof. ALESSANDRO RICCI

Presentata da
DARIO PAVLLO

Prima Sessione di Laurea
Anno Accademico 2015 – 2016

Sommario

Il riconoscimento delle gesture è un tema di ricerca che sta acquisendo sempre più popolarità, specialmente negli ultimi anni, grazie ai progressi tecnologici dei dispositivi embedded e dei sensori. Lo scopo di questa tesi è quello di utilizzare alcune tecniche di machine learning per realizzare un sistema in grado di riconoscere e classificare in tempo reale i gesti delle mani, a partire dai segnali mioelettrici (EMG) prodotti dai muscoli. Inoltre, per consentire il riconoscimento di movimenti spaziali complessi, verranno elaborati anche segnali di tipo inerziale, provenienti da una Inertial Measurement Unit (IMU) provvista di accelerometro, giroscopio e magnetometro.

La prima parte della tesi, oltre ad offrire una panoramica sui dispositivi wearable e sui sensori, si occuperà di analizzare alcune tecniche per la classificazione di sequenze temporali, evidenziandone vantaggi e svantaggi. In particolare, verranno considerati approcci basati su Dynamic Time Warping (DTW), Hidden Markov Models (HMM), e reti neurali ricorrenti (RNN) di tipo Long Short-Term Memory (LSTM), che rappresentano una delle ultime evoluzioni nel campo del deep learning.

La seconda parte, invece, riguarderà il progetto vero e proprio. Verrà impiegato il dispositivo wearable Myo di Thalmic Labs come caso di studio, e saranno applicate nel dettaglio le tecniche basate su DTW e HMM per progettare e realizzare un framework in grado di eseguire il riconoscimento real-time di gesture. Il capitolo finale mostrerà i risultati ottenuti (fornendo anche un confronto tra le tecniche analizzate), sia per la classificazione di gesture isolate che per il riconoscimento in tempo reale.

Indice

Capitolo 1

Introduzione

1.1 Panoramica sui sistemi wearable

Negli ultimi anni si è assistito ad un consistente aumento dei dispositivi *wearable* (indossabili) disponibili sul mercato, nonché a numerosi sistemi innovativi ancora in sviluppo. Questa sezione fornirà una panoramica su alcuni di questi dispositivi, evidenziando i salti tecnologici che ne hanno permesso la creazione.

Prima di procedere, tuttavia, è necessario definire cos'è un dispositivo wearable: come suggerisce il nome, si tratta di un dispositivo che si indossa in una qualsiasi parte del corpo (solitamente sul braccio o sul polso), ed ha lo scopo di estendere o arricchire le funzionalità umane. Dal punto di vista ingegneristico, un dispositivo wearable è composto da un sistema embedded a basso consumo (solitamente alimentato a batteria), con limitate capacità di calcolo rispetto ad un computer *general-purpose*. In genere, questi dispositivi sono dotati di sensori che permettono loro di interagire con il corpo umano e/o l'ambiente esterno, e si possono interfacciare direttamente con l'utilizzatore (ad esempio tramite uno schermo), oppure con altri dispositivi attraverso un protocollo di comunicazione standard come *Bluetooth*. In quest'ultimo caso si parla di comunicazione *machine-to-machine* (M2M), che è uno dei punti cardine del cosiddetto *Internet of Things* (IoT).

1.1.1 Evoluzione del mercato dei sistemi wearable

Sebbene si tratti di un argomento di recente interesse, la tecnologia wearable ha radici che risalgono agli inizi dell'elettronica digitale di massa. Basti pensare agli orologi da polso digitali, introdotti da società come Casio e Seiko negli anni '70: oltre all'orologio, infatti, questi dispositivi integravano altre funzionalità come cronometri e sensori.

Tuttavia, è solo negli ultimi anni che questo mercato ha subito un consistente aumento, grazie ai progressi tecnologici in ambito informatico ed elettronico, uniti all'enorme richiesta da parte dei consumatori. Si stima, infatti, che questo mercato raggiungerà un valore di 19 miliardi di dollari entro il 2019 [1].

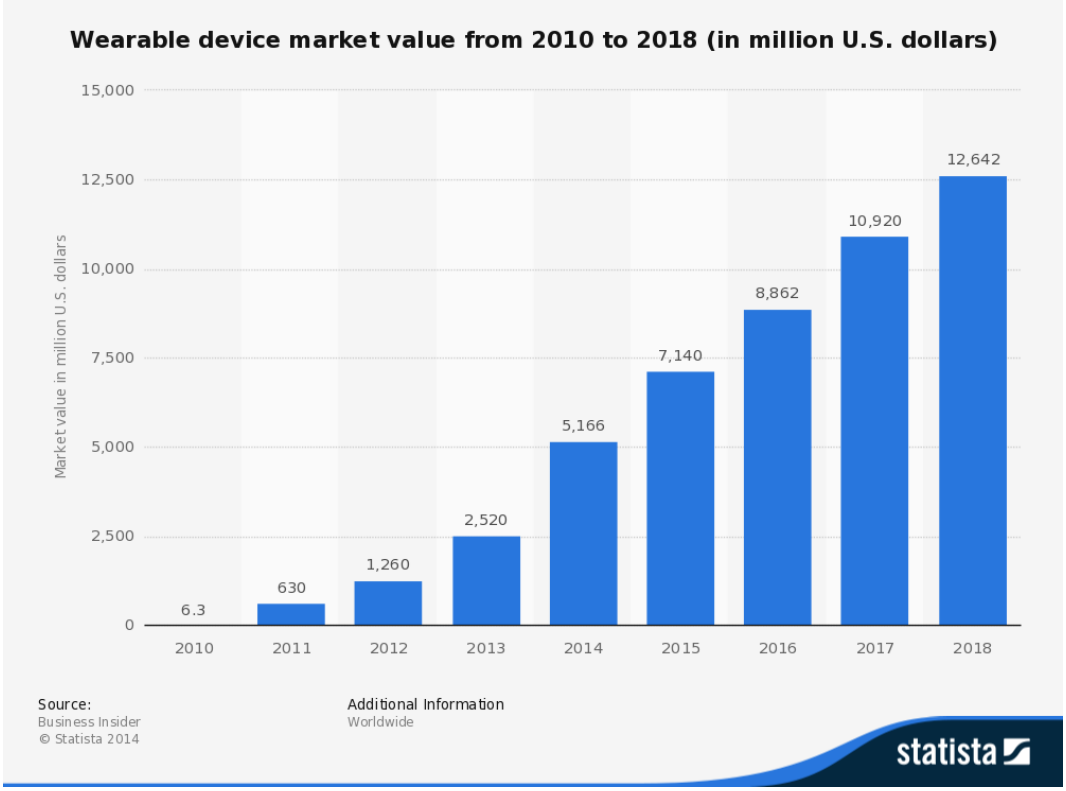


Figura 1.1: Andamento del mercato wearable e previsioni future. Si può facilmente notare l'impennata che il settore ha avuto a partire dal 2010.

Le applicazioni di questi dispositivi sono molteplici. Inizialmente, prima di approdare nel mercato di massa, venivano principalmente utilizzati in campo militare e *healthcare*, ad esempio come strumenti di diagnosi, o come protesi/interfacce per pazienti con perdite di funzionalità fisiche. Successivamente, è nato il mercato degli *smart watch*, di cui fanno parte Apple Watch, Pebble ed Android Wear, e dei *wrist band* come Fitbit, che monitorano i parametri vitali di chi li indossa. Alla classe dei dispositivi wearable appartengono anche i cosiddetti *smart glasses* come Google Glass, Epson Moverio e Microsoft HoloLens, che si indossano allo stesso modo degli occhiali normali e forniscono all'utente un ambiente in realtà aumentata.

(a) Fitbit (b) Apple Watch

Figura 1.2: Dispositivi wearable da polso

In generale, alcune delle applicazioni dei dispositivi wearable sono le seguenti:

- **Fitness/Sport:** sono in genere dotati di sensori in grado di monitorare il battito cardiaco (come Fitbit) e di tracciare i movimenti dell'utente, in modo da fornirgli statistiche sugli allenamenti. I dispositivi appartenenti a questa categoria sono spesso integrati con i *social network*.
- **Medicina:** permettono di monitorare i parametri vitali dei pazienti. Ad esempio, un recente campo di ricerca in questo ambito è rappresentato dai sensori cutanei per misurare il livello di glicemia nei diabetici, la quale unica alternativa sarebbe quella di prelevare un campione di sangue. Altri esempi, sono gli strumenti di diagnostica basati su elettrocardiogramma (ECG), elettroencefalogramma (EEG) ed elettromiografia (EMG), che però non sono adatti all'uso giornaliero. Tuttavia, alcune di queste tecnologie sono state riscoperte ed adattate al mondo consumer per scopi completamente diversi da quelli originali, con particolare riferimento all'elettromiografia per il riconoscimento di gesture, che è uno dei punti cardine di questa tesi.
- **Realtà aumentata (AR):** a questa categoria appartengono gli smart glasses citati precedentemente. Arricchiscono i sensi umani fornendo informazioni ricavate da fonti esterne, senza tuttavia isolare completamente l'utente dalla percezione del mondo reale. Solitamente, si proietta un *overlay* visivo nelle lenti degli occhiali.
- **Realtà virtuale (VR):** a differenza della realtà aumentata, i dispositivi VR immergono completamente l'utente in un ambiente virtuale, ren-

dendo questa tecnologia particolarmente adatta al settore gaming. Al momento, gli *headset* di realtà virtuale che stanno ricevendo i maggiori investimenti sono Oculus Rift, HTC Vive e PlayStation VR, ma sono presenti anche dei kit che permettono di trasformare il proprio smartphone in un visore di realtà virtuale, ad esempio Samsung Gear VR e Google Cardboard.

- **Interfacce di input:** concettualmente, i dispositivi appartenenti a questa categoria rappresentano una vera e propria estensione delle capacità umane. Infatti, permettono all'utente di interagire con le macchine in modo trasparente, usando ad esempio i gesti o la voce. A questa categoria appartengono i sistemi *Speech-to-text*, che trasformano la voce umana in testo e la utilizzano per impartire comandi ad uno smartphone (Google Now, Apple Siri, Microsoft Cortana), o il dispositivo Myo (che sarà oggetto di questa tesi), che misurando i segnali elettrici dei muscoli ed il movimento dell'avambraccio, è in grado di realizzare un'interfaccia di input configurabile.

- **General-purpose:** è il caso degli smart watch, che hanno lo scopo di fornire le stesse funzionalità degli smartphone, ma in modo più accessibile e semplificato.

1.1.2 Tecnologie per lo sviluppo dei dispositivi wearable

La possibilità di realizzare dispositivi wearable, come tutte le tecnologie presenti nel mercato IT, ha richiesto numerosi sforzi da parte della comunità scientifica. Nello specifico, dal punto di vista del software è stato necessario ripensare il modo di progettare i sistemi, mentre per quel che riguarda l'hardware è stato necessario sviluppare nuove tecnologie senza le quali non avremmo i dispositivi odierni. La spinta iniziale a questo settore è stata data dagli smartphone, che inizialmente si prefiggevano dei requisiti di difficile realizzazione. In altri termini, non esistevano ancora le piattaforme tecnologiche per realizzarli.

I primi due sistemi operativi moderni per smartphone sono stati iOS (di Apple) e Android (di Google), che hanno combinato lo stato dell'arte dei sistemi operativi e sistemi embedded per creare dei prodotti adatti alle risorse computazionali e requisiti degli smartphone, senza tuttavia perdere la flessibilità dei sistemi operativi desktop. Tra i requisiti, si riportano i consumi energetici limitati e la reattività, nonché la possibilità di funzionare con hardware diversi (specialmente per Android).

Come conseguenza di questa evoluzione, sono stati adottati diversi paradigmi di programmazione rispetto a quelli correntemente in uso. Dati i requisiti,

infatti, gli smartphone richiedono una programmazione fortemente orientata agli eventi ed al parallelismo.

Uno dei contributi fondamentali degli smartphone nello sviluppo dell'Internet of Things e dei dispositivi wearable è arrivato in seguito all'integrazione di vari sensori inerziali all'interno dei cellulari, come l'accelerometro, il giroscopio ed il magnetometro, in aggiunta ad altri sensori generici (ad esempio di temperatura, luminosità o pressione atmosferica). Ciò è stato reso possibile solamente grazie alla tecnologia *Microelectromechanical Systems* (MEMS), che nonostante esistesse già dagli anni '90, ha subito un enorme sviluppo ed abbattimento dei costi in seguito alla diffusione di massa degli smartphone. Come riferimento, al giorno d'oggi un accelerometro a 3 assi ha un costo minore di \$0.30 [2]. La tecnologia MEMS (Figura 1.3) permette di integrare le caratteristiche elettriche dei semiconduttori con quelle meccaniche dell'ambiente che ci circonda, permettendo di realizzare trasduttori di grandezze fisiche (come l'accelerazione o l'orientamento) su scala micrometrica.

Un altro contributo molto importante è stato dato dall'ampia diffusione delle CPU *RISC* (Reduced Instruction Set Computer) basate su architettura ARM (Advanced RISC Machine). Infatti, la quasi totalità delle CPU in ambito desktop è rappresentata dall'architettura x86 di Intel, che è molto inefficiente dal punto di vista energetico per via della sua architettura CISC (Complex Instruction Set Computer). Di conseguenza, non è utilizzabile su dispositivi embedded e smartphone.

Per concludere, le esigenze dei dispositivi wearable hanno portato anche allo sviluppo di nuovi protocolli di comunicazione fra macchine, il più importante dei quali è sicuramente *Bluetooth Low-Energy*: un'evoluzione del protocollo Bluetooth caratterizzata da bassissimi consumi energetici e da una breve latenza di connessione. Queste caratteristiche, lo rendono perfetto per quei dispositivi wearable che devono comunicare continuamente, e che possono entrare o uscire velocemente dallo *sleep-mode* (una modalità a consumo quasi nullo in cui il dispositivo è inattivo). Molti dispositivi embedded di recente concezione hanno adottato Bluetooth LE come protocollo di comunicazione, ed in genere si interfacciano con lo smartphone dell'utente o con il suo PC (attraverso un apposito dongle).

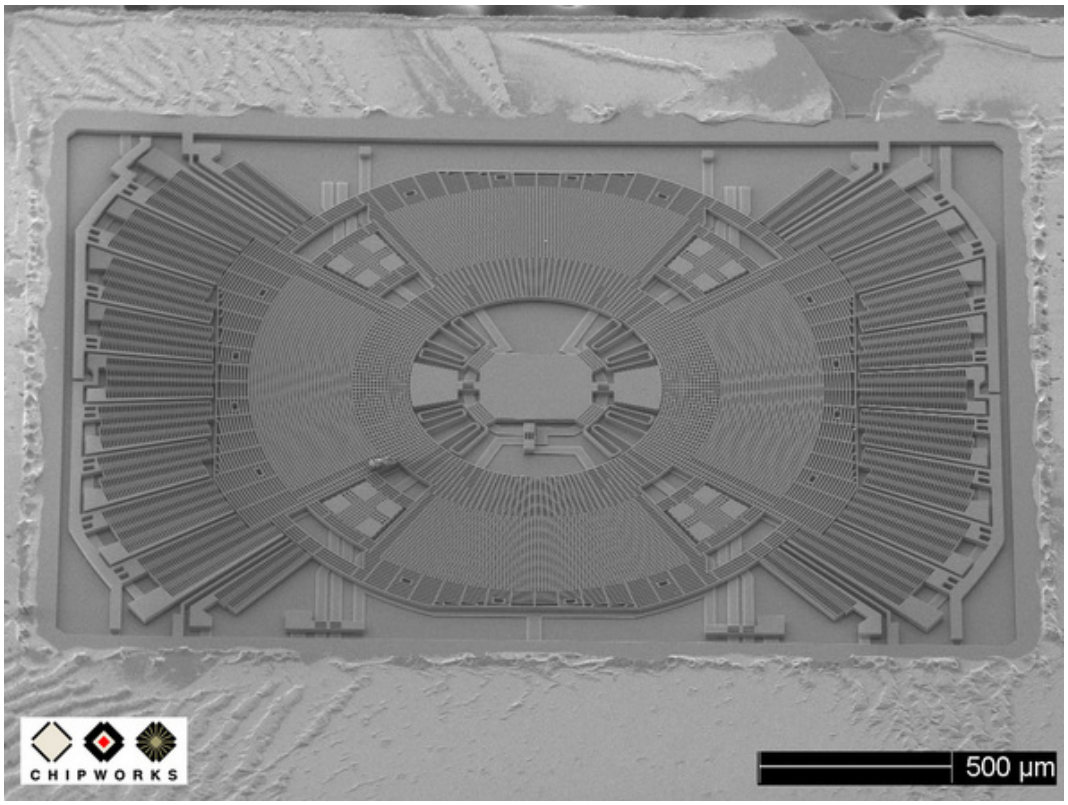


Figura 1.3: Giroscopio realizzato tramite tecnologia MEMS.

1.2 Importanza del riconoscimento delle gesture

Quello del riconoscimento dei gesti non è un ambito nuovo per l'informatica. Infatti, in letteratura scientifica questo tema è stato trattato a partire dagli anni '90, ed alcune delle tecniche utilizzate per risolvere il problema della classificazione di gesture erano già state sviluppate negli '80 per altri scopi, come il riconoscimento vocale o della scrittura, e sono state successivamente adattate. Tuttavia, negli ultimi anni, questo argomento sta catturando sempre di più l'attenzione del mercato IT, probabilmente anche grazie agli sviluppi tecnologici ottenuti in campo elettronico. Secondo un'indagine di Markets and Markets, il mercato dei sistemi basati sul riconoscimento di gesture raggiungerà un valore di 23.55 miliardi di dollari entro il 2020 (a differenza di un valore di 5.15 miliardi nel 2014) [3].

Inizialmente, il tema del riconoscimento delle gesture è stato portato alla massa grazie alla cinematografia (soprattutto tramite film di carattere fanta-

scientifico, tra cui i noti Minority Report e Iron Man), ma è stato il mercato del gaming a dare una vera spinta al settore. Tra i primi dispositivi adibiti al riconoscimento delle gesture si cita il Wii Remote (Figura 1.4a), il controller della console Wii di Nintendo. Successivamente sono arrivati anche i prodotti concorrenti: PlayStation Move di Sony e Kinect di Microsoft. Il principio di funzionamento di questi dispositivi verrà riassunto nella Sezione 1.3.

In un secondo momento si è capito che le tecnologie per il riconoscimento di gesture avevano un potenziale anche al di fuori del gaming, e di conseguenza sono nati nuovi dispositivi, mentre alcuni di quelli già esistenti sono stati adattati per applicazioni general-purpose (tra cui Microsoft Kinect, del quale è stato rilasciato l'SDK al pubblico nel 2011). Una lista non esaustiva delle possibili applicazioni di queste tecnologie è la seguente:

- **Gaming:** come periferica di input per controllare i videogiochi. L'utilizzo delle tecnologie basate su gesture è destinato ad aumentare ulteriormente in questo settore, specialmente con l'introduzione della realtà virtuale. Un dispositivo ideato per questo scopo è Leap Motion (Figura 1.4b).

- **Robotica:** in ambiti ad alto rischio (o dove è richiesta molta precisione), come nel settore chimico o farmaceutico, è necessario maneggiare materiali pericolosi che possono comportare un elevato rischio per l'uomo. Un robot industriale che riproduca gli stessi movimenti di un operatore umano permetterebbe di svolgere le operazioni in totale sicurezza. Inoltre, integrando la realtà aumentata in questo sistema, si otterrebbe una tecnologia molto efficace.

- **Periferica di input:** in sostituzione alle periferiche classiche come mouse e tastiera, specialmente in ambienti particolari. Ad esempio, ad una conferenza o presentazione è possibile cambiare le slide con il movimento della mano. In poche parole, si otterrebbe un'interfaccia tra uomo e macchina, denominata appunto *Human-Machine Interface* (HMI).

- **Intrattenimento:** alcuni televisori di recente concezione, appartenenti alla categoria delle *Smart TV*, possono essere controllati tramite gesture, senza la necessità di utilizzare il telecomando.

- **Medicina:** in questo caso sono presenti vantaggi sia per i medici che per i pazienti. I primi, possono interagire con un ambiente virtuale pur avendo le mani occupate (ad esempio per recuperare informazioni durante un'operazione chirurgica). I secondi, possono recuperare le funzionalità di una parte del corpo perduta.

- **Riconoscimento del linguaggio dei segni:** utile per trascrivere il linguaggio dei segni in testo.
- **Interazione con l'ambiente circostante:** in modo analogo alle periferiche di input, in un ambiente composto da oggetti intelligenti (*smart things*) sarà possibile interagire con essi selezionandoli (ad esempio indicandoli) e controllandoli. Con la diffusione della realtà aumentata e Internet of Things, questo è uno scenario molto realistico, specialmente nella domotica.

(a) Wii Remote (b) Leap Motion

Figura 1.4: Dispositivi per il riconoscimento di gesture

1.3 Sensori per il riconoscimento di gesture

Come accennato precedentemente, una delle possibili applicazioni dei dispositivi wearable è quella di riconoscere i gesti ed i movimenti delle mani, per poi interpretarli ed utilizzarli come input per un qualsiasi software. Nel caso dei dispositivi wearable, solitamente il riconoscimento avviene sulla base di alcuni sensori posti sul dispositivo stesso (ad un esempio un accelerometro). Elaborando e sfruttando le proprietà fisiche di questi segnali, è possibile tracciare il movimento effettuato dall'utente, oppure compararlo ad un modello già noto (*pattern*).

Tuttavia, non tutti i sistemi di riconoscimento di gesture si basano su dispositivi indossabili, in quanto sono state sviluppate anche tecniche per riconoscere i movimenti a partire da un flusso video (di conseguenza sarebbe sufficiente una semplice webcam). Esistono anche alcuni sistemi, soprattutto utilizzati in ambito scientifico, che a partire da più telecamere collocate appositamente, sono in grado di approssimare un modello 3D dell'utente, e di riconoscere i suoi movimenti con una maggiore precisione. Infine, sul mercato sono anche

presenti dispositivi che fondono più sensori visivi, ad esempio una telecamera unita ad un sensore di profondità. A questa categoria appartiene il noto Microsoft Kinect (Figura 1.5b), che con un sensore di profondità ad infrarossi è in grado di mappare in 3D l'ambiente circostante e di sovrapporlo al flusso video ottenuto dalla sua telecamera.

Come regola generale, se si hanno più dati a disposizione, sarà possibile effettuare un riconoscimento migliore, grazie alla capacità di discriminare più facilmente i segnali. A tale proposito, molti sistemi applicano la cosiddetta *sensor fusion*, che consiste nel combinare i dati di più sensori per aumentare la precisione dei segnali, o derivare nuovi segnali che hanno un contenuto informativo maggiore. Si riportano alcuni esempi degni di nota:

- **Accelerometro + GPS:** il primo ha un tempo di risposta rapido ma è impreciso, mentre il secondo è l'opposto. In linea teorica, integrando due volte il valore dell'accelerazione ottenuto dal primo è possibile ottenere lo spostamento (dato che $\mathbf{a} = \frac{\mathrm{d}^2\mathbf{x}}{\mathrm{d}t^2}$), ma nella realtà ciò non è praticabile, in quanto il rumore e l'errore del segnale si propagherebbero eccessivamente con il passare del tempo, dando luogo ad un risultato completamente errato nel giro di pochi decimi di secondo. Inoltre, si otterrebbe solo uno spostamento relativo, e non un posizionamento assoluto, dato che non si conosce la condizione iniziale. Tuttavia, è possibile sfruttare i vantaggi di entrambi i sistemi per ottenere una lettura più precisa rispetto a quella ottenibile con i due componenti presi singolarmente: si può utilizzare il posizionamento assoluto del GPS per ottenere una posizione corretta sul lungo termine, unendola all'accelerazione per calcolare un valore più preciso sul breve termine.

- **Giroscopio + Magnetometro:** è analogo all'esempio precedente. Il giroscopio fornisce solamente la differenza di orientamento in un dato istante (la derivata dell'orientamento assoluto), ed è molto veloce a rispondere. Il magnetometro, misurando il campo magnetico della Terra, è in grado di leggere l'orientamento assoluto del dispositivo, ma ha un tempo di risposta basso. Unendo questi due sensori si può ottenere una lettura corretta e veloce.

- **Accelerometro + Videocamera:** è equivalente al primo esempio, con l'unica differenza che al posto di utilizzare il GPS, si impiega una videocamera per tracciare il posizionamento assoluto del dispositivo. Questa tecnica è utile in ambienti chiusi, dove si rende necessaria una lettura precisissima (nell'ordine dei centimetri), e per questo motivo viene attualmente impiegata nei sistemi di realtà virtuale dedicati al gaming, come Oculus Rift e HTC Vive. In questi dispositivi, una telecamera ad infrarossi riprende alcuni *marker* posizionati sul caschetto.

(a) HTC Vive (b) Microsoft Kinect 2

Figura 1.5: Dispositivi per il tracking visivo

Nella pratica, i sopracitati sensori sono presenti tutti assieme sotto forma di moduli chiamati *Inertial Measurement Unit* (IMU), che vengono identificati dal numero di assi che rilevano (in genere, quelli a 6 assi contengono un accelerometro ed un giroscopio, mentre quelli a 9 assi sono dotati di accelerometro, giroscopio e magnetometro).

È doveroso comprendere le tecniche per trattare questi segnali a livello software, in modo da ottenere i risultati appena citati. Nel corso del tempo sono stati sviluppati vari approcci, ma i più efficaci sono di tipo statistico. Andando nel dettaglio, le letture dei vari sensori possono essere viste come osservazioni di un processo stocastico soggetto a rumore, e sfruttando le proprietà statistiche di questi segnali è possibile stimare dei valori più precisi. L'implementazione maggiormente nota di questo principio prende il nome di *Filtro di Kalman* [4], e consiste in un filtro ricorsivo che approssima lo stato di un sistema dinamico a partire da una serie di osservazioni. Nella pratica è estremamente efficace, e per questo motivo viene impiegato in numerosi settori, tra cui quello aeronautico e quello automobilistico, ma sta prendendo piede anche nel campo dell'Internet of Things (droni, modellismo, dispositivi intelligenti, automobili a guida autonoma, dispositivi wearable).

Sebbene, come si è appena visto, il mondo dei sensori sia molto eterogeneo, il risultato che si vuole ottenere è sempre lo stesso: un segnale finale che rappresenta una grandezza fisica (solitamente a bassa dimensionalità, come una posizione nello spazio 2D o 3D). Di conseguenza, le tecniche utilizzate per classificare questi segnali (o nel caso di questo progetto, classificare segnali che rappresentano gesture) sono sempre le stesse, e non dipendono dalla fonte da cui è stato ottenuto il segnale (se non marginalmente). Da un punto di vista concettuale, quasi tutti i sistemi di riconoscimento sono basati su una *pipeline* che riceve in ingresso il/i segnali originali (che possono essere immagini, audio, o dati provenienti da sensori), e li trasformano in altri segnali attraverso un processo noto come *feature extraction*. Nel caso di immagini o video, ad esempio, sarà necessario applicare opportune tecniche di *computer vision* per

estrarre le informazioni volute e ridurre la dimensionalità del segnale, mentre nel caso di segnali elettrici, nella maggior parte dei casi sarà sufficiente applicare alcuni filtri.

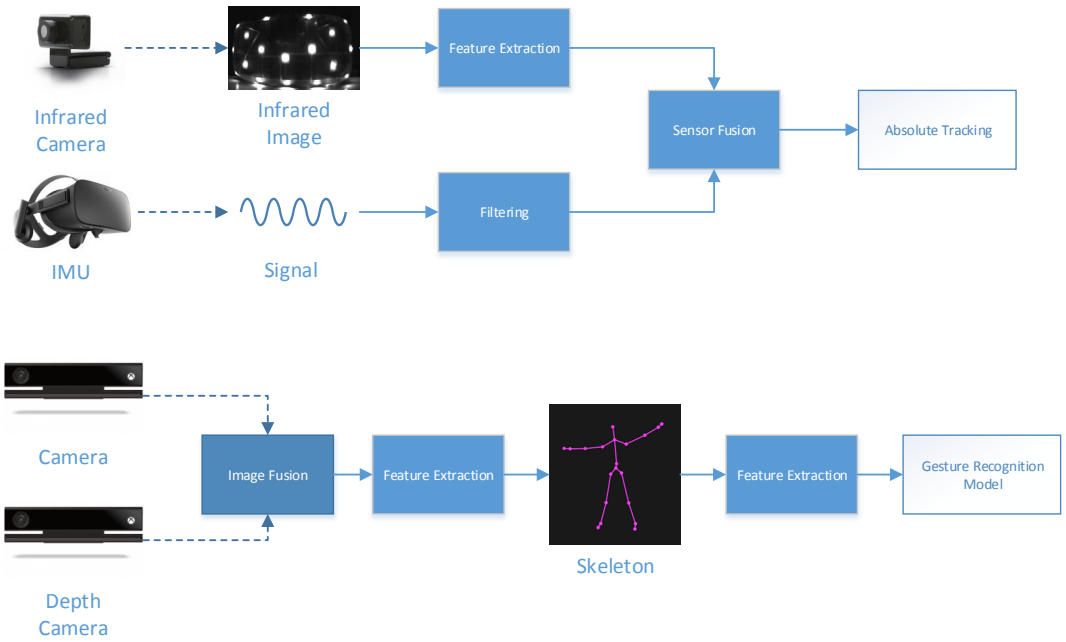


Figura 1.6: Pipeline per il tracking o il riconoscimento di gesture. Sopra si riporta il caso di studio di Oculus Rift, sotto quello di Kinect.

Un recente sviluppo degno di nota è rappresentato da *Project Soli* di Google [5], una nuova tecnologia di riconoscimento delle gesture basata sul concetto di *radar*. Un apposito dispositivo emette radiazioni elettromagnetiche ad alta frequenza in un raggio ristretto; eventuali oggetti posti davanti a questo raggio rifletteranno le radiazioni verso il ricevitore, che elaborerà le variazioni di energia e riconoscerà il movimento effettuato con un'elevata precisione.

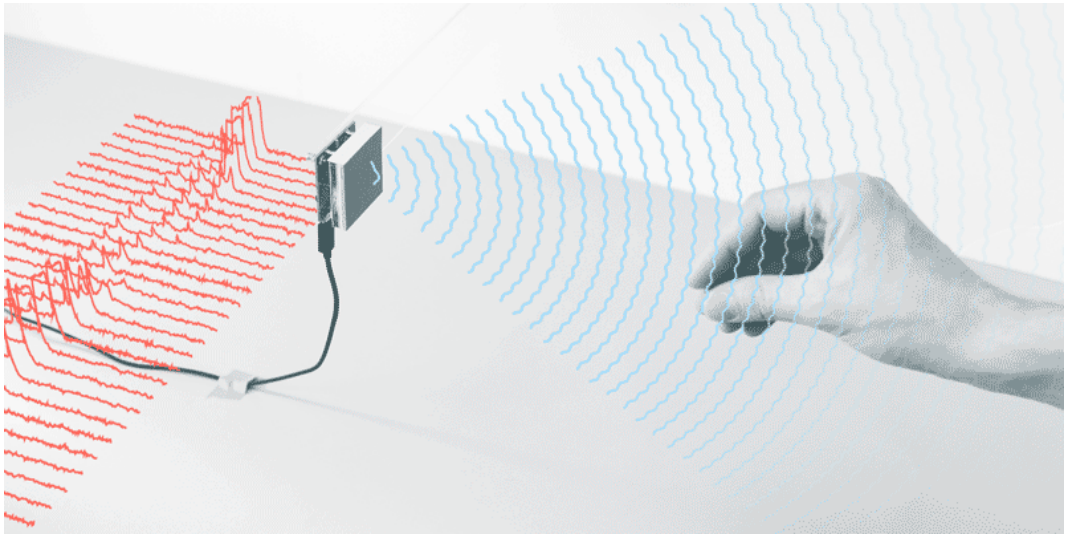


Figura 1.7: Project Soli, di Google

1.4 Il dispositivo Myo come caso di studio

Per la realizzazione di questo progetto è stato scelto il dispositivo *Myo* di Thalmic Labs, perché rappresenta una delle ultime evoluzioni nel campo delle interfacce di input per il riconoscimento di gesture.

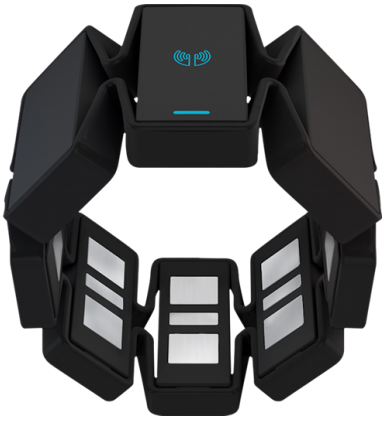


Figura 1.8: Myo Armband

Dal punto di vista tecnico, l'armband Myo è dotato di un'Inertial Measurement Unit da 9 assi (accelerometro, giroscopio e magnetometro), che gli permettono di rilevare l'orientamento assoluto del dispositivo e di tracciarne i movimenti. Tuttavia, la vera evoluzione del Myo sta nella possibilità di rilevare i gesti delle mani senza l'uso di videocamere o sistemi analoghi. A tale scopo, il

device è dotato di 8 sensori *sEMG* (*Surface EMG*), che, in modo identico agli stessi dispositivi per uso medico, misurano i segnali elettrici prodotti dall'attività muscolare; il Myo è una fascia elastica che si indossa nell'avambraccio, il quale viene avvolto circolarmente dagli 8 sensori. Durante l'esecuzione di una gesture (ad esempio un pugno), i muscoli dell'avambraccio si contraggono con una configurazione particolare, e misurando i segnali emessi delle varie fibre muscolari è possibile identificare la gesture effettuata. Informazioni più precise su questi segnali verranno fornite nell'apposita sezione 3.3.1 sull'analisi dei segnali.

Thalmic Labs fornisce un SDK (*Software Development Kit*) da utilizzare in abbinamento a Myo. Di norma, Myo è già in grado di riconoscere 5 gesture predefinite senza alcuno sforzo da parte dello sviluppatore, ma l'obiettivo di questa tesi è di poter essere in grado di definire nuove gesture a piacere. Chiaramente, gli scopi del produttore sono differenti da quelli di questo progetto. Dato che Myo è destinato al mercato consumer, il prodotto deve funzionare correttamente con qualsiasi utente (*user-independent*), ed il riconoscimento delle gesture dev'essere consistente. Per questo motivo, i produttori hanno deciso di concentrarsi su un numero basso di gesture in modo da rendere il prodotto affidabile dal punto di vista commerciale. Al contrario, questo progetto mira a creare un sistema personalizzabile, che però ha lo svantaggio di richiedere una messa a punto più complessa.

Nel prossimo capitolo, invece, verrà introdotto l'argomento del *machine learning* come possibile soluzione al problema dell'identificazione delle gesture a partire dai segnali misurati.

Capitolo 2

Classificazione di serie temporali

Come accennato nel capitolo precedente, i segnali ottenuti dalla sorgente (nel caso di questo progetto, il Myo) possono essere soggetti ad una fase di *preprocessing* per estrarne le caratteristiche importanti, e come risultato finale si ottiene un nuovo segnale. A questo punto, il problema sta nel trovare una tecnica che permetta di associare questo segnale ad un'etichetta, o in altri termini, identificare la classe a cui appartiene. Questo problema è noto in letteratura come *classificazione*, e nel caso di questa tesi, i pattern da classificare sono i segnali ottenuti dai sensori, mentre le classi sono l'insieme di gesture che si vogliono riconoscere. In ogni caso, le tecniche utilizzate per affrontare questi quesiti sono generiche ed adattabili a numerosi campi, e come conseguenza, in questo capitolo si farà riferimento a dei generici *pattern*, e non solo *gesture*.

Formalmente, il problema si può formulare come segue: si vuole effettuare la classificazione di una serie (o sequenza) temporale (*time series*)

$$\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_n}) = \begin{bmatrix} x_{1,1} & x_{1,2} & \ldots & x_{1,n} \\ x_{2,1} & x_{2,2} & \ldots & x_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m,1} & x_{m,2} & \ldots & x_{m,n} \end{bmatrix} \tag{2.1}$$

Dove $\mathbf{x}$ è un sequenza finita composta da n elementi di tipo vettore (n può variare da pattern a pattern), denominati *feature vectors*. Ognuno di questi vettori è composto da m elementi (m costante) reali. Il pattern, di conseguenza, si può rappresentare con una matrice $m \times n$, sebbene dal punto di vista semantico sia da considerarsi come una sequenza di vettori. Il significato di $\mathbf{x}$ può variare a seconda delle applicazioni, ma in questo caso è da interpretarsi come una serie di osservazioni di un sistema nel corso del tempo. Nel caso di un segnale proveniente da un sensore, ad esempio, ogni elemento del vettore corrisponde al valore del segnale campionato ad intervalli regolari. Inoltre, il

segnale può essere anche multivariato, cioè composto da un numero di variabili pari ad m. Ad esempio, un accelerometro a 3 assi sarà composto dalle tre componenti $\mathbf{x_i} = (x, y, z)$, quindi $m = 3$. Il campionamento di questo sensore ad una frequenza di 100 Hz per 2 secondi, produrrà un vettore $\mathbf{x}$ composto da $n = 200$ elementi, a loro volta composti da $m = 3$ componenti. Si ricorda nuovamente che i feature vectors potrebbero aver subito una fase di preprocessing, quindi non è garantito che rappresentino una grandezza fisica concreta (a differenza dell'esempio appena riportato).

Successivamente, la sequenza $\mathbf{x}$ verrà passata in input ad un *classificatore*, che restituirà in output l'etichetta della classe a cui $\mathbf{x}$ presumibilmente appartiene, considerando che l'insieme delle classi è finito e conosciuto in partenza.

A questo punto, si rende necessario stabilire la tecnica di funzionamento del classificatore. È possibile seguire due approcci generali:

- **Modello basato su regole (rule-based):** per ogni classe, è possibile codificare un insieme di regole che il pattern deve soddisfare per appartenere ad essa. Ad esempio, data una serie di punti nello spazio 2D, si può dire che essa rappresenta una circonferenza se i suoi punti giacciono sulla curva descritta dall'equazione $x^2 + y^2 = r^2$. Per aumentare la capacità di generalizzazione della regola, è possibile introdurre anche una tolleranza basata sulla distanza. Come si può intuire, questo approccio presenta varie problematiche: è necessario stabilire e programmare una regola per ogni classe, e la precisione di classificazione dipende fortemente dalla bontà di essa. Nella pratica, è difficile codificare regole che funzionino correttamente in uno scenario realistico.

- **Modello basato su esempi (pattern-based):** l'alternativa è quella di costruire un sistema che impari a classificare i pattern a partire da una serie di esempi. Collezionando un insieme di esempi significativi ed etichettati, sarà possibile fornirli in input al sistema, che produrrà un modello in grado di classificare pattern sconosciuti.

Il secondo approccio è parte di un'area nota come *machine learning*, e rappresenta una parte integrante di questa tesi. Infatti, di seguito verranno descritte varie tecniche di machine learning adatte a classificare serie temporali, e successivamente, verranno utilizzate in modo concreto per lo svolgimento del progetto.

Gli algoritmi di machine learning si suddividono in due tipi: *supervised* e *unsupervised*. I primi corrispondono al caso citato precedentemente, cioè l'insieme delle classi è noto a priori, e gli esempi forniti all'algoritmo di apprendimento sono già stati etichettati. Al contrario, i secondi ricevono un

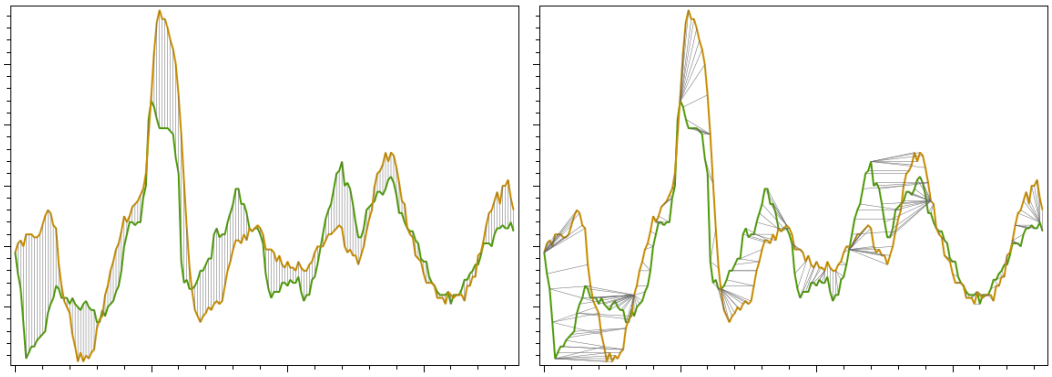
insieme di pattern eterogenei e non etichettati; successivamente, sarà l'algoritmo a creare le etichette e a raggrupparli secondo una caratteristica comune. Un tipico problema di natura unsupervised è il *clustering*. Ad esempio, si supponga di avere un insieme di punti nello spazio 3D che rappresentano stelle, che non sono distribuite uniformemente, ma tendono a raggrupparsi in conglomerati (galassie). Senza avere alcuna informazione aggiuntiva, è possibile sfruttare alcuni algoritmi come il *k-means clustering* per enumerare le galassie ed assegnare ad ogni stella la galassia di appartenenza.

Nel caso di questa tesi, si deve ricercare un algoritmo di *supervised learning*, dato che l'insieme di gesture viene stabilito in partenza, e gli esempi forniti all'algoritmo di apprendimento rientrano necessariamente nelle categorie previste.

Quello del machine learning è un campo molto vasto, ed esistono numerose tecniche per affrontare il problema. Quest'ultime non sono equivalenti tra loro, e devono essere scelte sulla base della natura dei pattern che si vogliono riconoscere. Nelle sezioni successive si riportano alcuni approcci idonei a classificare pattern temporali, utilizzati con successo in molti campi, ed adatti anche per classificare gesture.

2.1 Dynamic Time Warping

Il *Dynamic Time Warping* (DTW) è un algoritmo che permette di misurare la similarità fra due sequenze, che possono anche avere lunghezze differenti fra loro. In sintesi, l'algoritmo trova un allineamento ottimale non lineare fra le due sequenze, e ne calcola la distanza tramite una *funzione distanza* specifica dell'applicazione. Il risultato che si ottiene è l'allineamento di costo minimo fra tutti i possibili allineamenti. Dal punto di vista pratico, DTW permette di confrontare sequenze temporali che possono variare in velocità, ed inizialmente fu concepito proprio per poter essere utilizzato nel riconoscimento vocale [6]. Supponendo, ad esempio, di avere due registrazioni vocali di una certa frase, DTW è in grado di trovare l'allineamento ottimale fra le due sequenze audio, restituendo eventualmente un valore di distanza/similarità. Inoltre, l'allineamento trovato può anche non essere lineare, cioè può variare all'interno delle sequenze. Di conseguenza, facendo riferimento all'esempio precedente, l'algoritmo è in grado di trovare l'allineamento ottimale anche se le parole nella frase vengono pronunciate a velocità diverse tra loro. Naturalmente, DTW è anche in grado di allineare sequenze a velocità uguali, ma traslate nel tempo. Riassumendo, si può affermare che l'algoritmo è invariante alla traslazione ed alla velocità.



(a) Distanza euclidea (b) Distanza DTW

Figura 2.1: Confronto fra due approcci per misurare la distanza fra due sequenze temporali. A sinistra si ha una semplice distanza euclidea senza alcun tipo di allineamento, mentre a destra si ha l'allineamento ottimo trovato da DTW.

Formalmente, il Dynamic Time Warping si può descrivere come segue: siano date due sequenze di *feature vectors* (dello stesso tipo dell'eq. (2.1))

$$\begin{aligned} \mathbf{s} &= (\mathbf{s_1}, \mathbf{s_2}, \ldots, \mathbf{s_n}) \\ \mathbf{t} &= (\mathbf{t_1}, \mathbf{t_2}, \ldots, \mathbf{t_m}) \end{aligned} \tag{2.2}$$

Si vuole trovare un cammino $\mathbf{w}$ (*warping path*) composto da coppie di tipo $(\mathbf{s_i}, \mathbf{t_j})$ con $i \in [1..n]$ e $j \in [1..m]$, tale che il costo del cammino $\mathbf{w}$, definito come

$$\mathrm{c}_{\mathbf{w}}(\mathbf{s}, \mathbf{t}) = \sum_{k=1}^{L} \mathrm{dist}(\mathbf{s_i^{[k]}}, \mathbf{t_j^{[k]}}) \tag{2.3}$$

sia il minimo tra tutti i cammini possibili, dove L denota la lunghezza del cammino, mentre *dist* denota la funzione distanza (costo) tra i feature vectors, che verrà discussa in seguito. Identificando come $\mathbf{w}^*$ il cammino ottimo, si ha che

$$\mathrm{DTW}(\mathbf{s}, \mathbf{t}) = \mathrm{c}_{\mathbf{w}^*}(\mathbf{s}, \mathbf{t}) = \min\{\mathrm{c}_{\mathbf{w}}(\mathbf{s}, \mathbf{t})\} \tag{2.4}$$

I costi possono essere rappresentati su una matrice $n \times m$, ed il cammino deve necessariamente seguire i punti di quest'ultima. Come si può facilmente intuire, il problema non è ben definito allo stato attuale, ed è di difficile soluzione, in quanto esiste un numero esponenziale di possibili cammini. Per rendere il problema risolvibile, è necessario introdurre una serie di vincoli:

- **Condizione al contorno:** il cammino deve iniziare nel primo punto di entrambe le sequenze, e finire nell'ultimo punto di entrambe le sequenze. In altri termini: $\mathbf{w_0} = (1, 1)$ e $\mathbf{w_L} = (n, m)$

- **Condizione di monotonia:** dato un cammino $\mathbf{w}$ di lunghezza L, tale che $\mathbf{w_k} = (i_k, j_k)$, si deve avere che $i_1 \leq i_2 \leq \ldots \leq i_k \leq \ldots \leq i_L$ e $j_1 \leq j_2 \leq \ldots \leq j_k \leq \ldots \leq j_L$. Ciò equivale a dire che il cammino non può mai tornare indietro.

- **Condizione di continuità:** dato un cammino $\mathbf{w}$, tale che $\mathbf{w_k} = (i_k, j_k)$, deve essere soddisfatta la seguente condizione:

  $$\mathbf{w_k} - \mathbf{w_{k-1}} \in \{(0, 1), (1, 1), (1, 0)\}$$

  In breve, il cammino si può muovere solamente tra celle adiacenti nella matrice dei costi, e non può effettuare salti. Ciò garantisce che si ottenga un cammino continuo, e non una serie di cammini spezzati. Si noti anche che questo vincolo ingloba la condizione di monotonia descritta nel punto precedente.

Le suddette condizioni permettono di risolvere il problema tramite la seguente relazione di ricorrenza:

$$\begin{aligned} \mathrm{d}(i, j) = \mathrm{dist}(\mathbf{s_i}, \mathbf{t_j}) + \min\{ \quad & \mathrm{d}(i-1, j), \\ & \mathrm{d}(i-1, j-1), \\ & \mathrm{d}(i, j-1) \qquad \} \\ 1 \leq i \leq n & \\ 1 \leq j \leq m & \end{aligned} \tag{2.5}$$

È altresì necessario porre i seguenti casi base: $\mathrm{d}(0, 0) = 0$, $\mathrm{d}(i, 0) = +\infty$ e $\mathrm{d}(0, i) = +\infty$. In questo modo, si può calcolare la distanza DTW come $\textsc{DtwDistance}(\mathbf{s}, \mathbf{t}) = d(n, m)$ e procedere ricorsivamente.

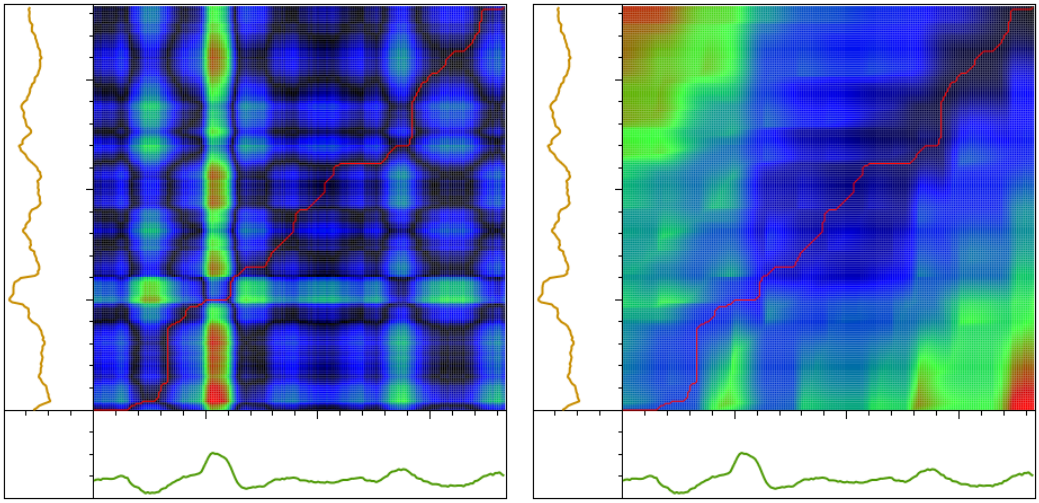
Valutando direttamente la relazione nell'eq. (2.5) si ottiene ancora un costo computazionale esponenziale, ma è possibile implementare l'algoritmo in maniera efficiente avvalendosi della cosiddetta tecnica della *programmazione dinamica*. In questo modo, è possibile calcolare la distanza DTW in tempo polinomiale, dato che le parti già valutate vengono memorizzate e non devono essere ricalcolate ad ogni iterazione dell'equazione di ricorrenza.
Di seguito, si riporta lo pseudocodice dell'algoritmo Dynamic Time Warping, implementato tramite programmazione dinamica.

Algoritmo 1 Dynamic Time Warping

```
1: function DtwDistance(s[1..n], t[1..m])
2:     d ← matrix[0..n, 0..m]
3:     for i ← 1 to n do
4:         d_{i,0} ← +∞
5:     end for
6:     for j ← 1 to m do
7:         d_{0,j} ← +∞
8:     end for
9:     for i ← 1 to n do
10:        for j ← 1 to m do
11:            cost ← dist(s_i, t_j)
12:            d_{i,j} ← cost + min{ d_{i−1,j} , d_{i,j−1} , d_{i−1,j−1} }
13:        end for
14:    end for
15:    return d_{n,m}
16: end function
```

Come si può intuire, questa implementazione ha un costo computazionale di $\Theta(n \cdot m)$, dove n ed m sono rispettivamente le lunghezze delle sequenze $\mathbf{s}$ e $\mathbf{t}$. Se le sequenze hanno lunghezza uguale, pari ad N, l'algoritmo ha un costo di $\Theta(N^2)$.

Qualora fosse necessario, per ottenere l'intero cammino (e non solo la distanza complessiva) si può eseguire una fase di *backtracking* partendo da $d[n, m]$ e procedendo al contrario fino ad arrivare a $d[0, 0]$, scegliendo sempre il minimo tra le 3 distanze ad ogni iterazione.



(a) Distanze dist($\mathbf{s_i}, \mathbf{t_j}$) (b) Matrice dei costi $d_{i,j}$

Figura 2.2: Funzionamento di DTW con le stesse sequenze della fig. 2.1. La linea rossa rappresenta il warping path ottimale $\mathbf{w}$.

2.1.1 Scelta della funzione distanza

Come accennato precedentemente, Dynamic Time Warping si serve di una funzione distanza definita dall'utente, che in questo contesto è stata denominata *dist*. Essa prende in ingresso due vettori $\mathbf{x}$ e $\mathbf{y}$ (che in questo caso assumono il valore dei feature vectors $\mathbf{s_i}$ e $\mathbf{t_j}$), e restituisce un valore scalare non negativo come risultato. Semanticamente, questo valore dovrebbe corrispondere ad una distanza fra i vettori, o comunque una misura di dissimilarità (dove un valore di 0 denota vettori identici).

La scelta di questa funzione dipende dall'applicazione e dai dati che vengono trattati, ma in genere si adottano le seguenti strategie:

- **Valori scalari:** se x e y sono valori scalari (cioè vettori ad una dimensione), l'algoritmo DTW si riduce al caso monodimensionale (cioè vengono confrontate due sequenze monodimensionali), ed è sufficiente calcolare la distanza come $|x-y|$, a meno che non si vogliano dare esplicitamente pesi diversi a distanze diverse (tramite l'uso di un esponente, ad esempio).

- **Vettori dotati di significato geometrico:** se i feature vectors rappresentano una grandezza fisica in uno spazio geometrico, la scelta più ragionevole è quella di utilizzare la *distanza euclidea* per confrontarli,

definita come:

$$\text{dist}(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\| = \sqrt{\sum_{i=1}^{k} (x_i - y_i)^2} \tag{2.6}$$

Esempi che rientrano in questo caso, sono tutti i vettori nello spazio bidimensionale o tridimensionale, come posizioni (punti), velocità o accelerazioni. Per confrontare i vettori derivanti da un accelerometro, quindi, la scelta più corretta è di usare la distanza euclidea.

- **Altri casi:** se i valori non hanno un significato ben preciso, oppure se non si è in grado di fare alcuna considerazione sulla loro natura, si può ricorrere alla distanza di Manhattan, definita come:

$$\text{dist}(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|_1 = \sum_{i=1}^{k} |x_i - y_i| \tag{2.7}$$

dove k è la dimensione dei feature vectors, che ovviamente dev'essere identica per $\mathbf{x}$ e $\mathbf{y}$.

In alternativa, se i vettori sono il risultato di un processo stocastico, è possibile sfruttare le correlazioni fra le variabili per ottenere un valore di distanza più attendibile [7]. A tale scopo, si utilizza la *distanza di Mahalanobis*, definita come:

$$\text{dist}(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^{\mathrm{T}} \, S^{-1} \, (\mathbf{x} - \mathbf{y})} \tag{2.8}$$

dove S denota la matrice di covarianza della distribuzione di probabilità che genera $\mathbf{x}$ e $\mathbf{y}$. Si noti, inoltre, che se S corrisponde alla matrice identità ($S = I$), ci si riduce al caso della distanza euclidea.

2.1.2 Classificazione

È doveroso specificare che Dynamic Time Warping non è un algoritmo di classificazione, bensì una funzione che misura la distanza fra due sequenze. Per eseguire la classificazione vera e propria, dato un insieme di classi con i template associati, è necessario decidere una logica di confronto con i suddetti template, ed un criterio che permetta di stabilire qual è la classe più probabile a cui appartiene una certa sequenza.

Una possibile idea potrebbe essere quella di eseguire l'algoritmo Dynamic Time Warping tra la sequenza da classificare e tutti i template appartenenti al training set, in modo da ottenere una lista di distanze. A questo punto,

per trovare la classe corrispondente alla sequenza in oggetto, sarà sufficiente scegliere quella associata al template con la distanza minima. Tuttavia, questo approccio è prono a dare risultati errati, dato che la classe con distanza minima potrebbe combaciare per errore.

Un approccio più robusto consiste nell'utilizzare l'algoritmo *k-Nearest Neighbors* (k-NN) per eseguire la classificazione. Fissato un k piccolo e costante, si selezionano i k template aventi distanza minima, e tra questi, si seleziona la classe avente la frequenza massima (cioè quella ripetuta più volte tra i k risultati). Si noti anche che il caso descritto precedentemente è equivalente ad un k-NN con $k = 1$ (*1-nearest neighbor*).

In alternativa, è possibile utilizzare altri classificatori, come le *Support Vector Machines* o le reti neurali.

2.1.3 Ottimizzazioni

Precedentemente è stato menzionato che DTW ha un costo computazionale quadratico rispetto alla dimensione dell'input. Ciò, a seconda delle applicazioni, potrebbe non essere desiderabile. Per aumentare le prestazioni del Dynamic Time Warping è possibile limitare ulteriormente lo spazio di ricerca ponendo un vincolo legato al *warping* massimo del cammino. In breve, si impone il passaggio del cammino per una banda (*Sakoe-Chiba band*) che scorre lungo la diagonale della matrice d. Ciò equivale a dire che per ogni punto del cammino $\mathbf{w_k} = (i_k, j_k)$ deve valere $|i_k - j_k| \leq T$, dove T è la larghezza della banda. In questo modo, il costo computazionale dell'algoritmo è pari a $\Theta(T \cdot N)$, dove N è la lunghezza delle due sequenze da confrontare (supponendo che siano di lunghezza uguale). Solitamente, come larghezza della banda T si sceglie una percentuale compresa fra l'1% ed il 10% della lunghezza della sequenza. Inoltre, limitando il warping massimo del cammino, si evita che l'algoritmo trovi allineamenti irrealistici.
Alternativamente, è possibile limitare la *pendenza* del warping path. Ciò equivale a imporre il passaggio per un parallelogramma, piuttosto che una banda.

Esistono anche algoritmi di approssimazione per DTW, come FastDTW. Questi, restituiscono un risultato diverso rispetto al DTW classico e talvolta possono essere imprecisi, ma hanno il vantaggio di avere spesso un costo di $\mathcal{O}(N)$, che li rende desiderabili dove siano richieste prestazioni elevate.

Infine, se le sequenze rappresentano segnali campionati, e questi ultimi cambiano lentamente nel tempo (rispetto alla frequenza di campionamento), è possibile eseguire un *downsampling* del segnale, cioè ricampionarlo con una frequenza più bassa. Questo approccio è teoricamente corretto solamente se la frequenza di campionamento è superiore al doppio della banda del segnale, cioè

$f_s > 2B$ (per il *teorema del campionamento di Nyquist-Shannon*). Se questa condizione non è soddisfatta, il campionamento produrrà una distorsione del segnale nota come *aliasing*. Per evitare questo problema, è possibile applicare un filtro passa basso prima di ricampionare il segnale, anche se ciò causerà inevitabilmente la perdita delle informazioni ad alta frequenza.

2.1.4 Vantaggi e svantaggi

In molte applicazioni, Dynamic Time Warping è un algoritmo da cui si ottengono ottimi risultati, ed è relativamente semplice da implementare. Dal punto di vista della sua applicazione nel campo del machine learning, ha l'indubbio vantaggio di non richiedere una fase di *learning*, dato che è sufficiente selezionare un pattern ed usarlo come modello.

Un problema non trascurabile di DTW è la sua sensibilità alle variazioni di ampiezza: infatti, dato che l'algoritmo trova un allineamento ottimale e calcola la distanza fra le varie coppie di punti delle sequenze, se si confrontano due segnali simili, ma diversi in ampiezza, DTW restituirà un elevato valore di distanza fra esse. Per mitigare questa problematica, è possibile normalizzare i segnali in modo che abbiano un'ampiezza media costante.

2.2 Hidden Markov Models

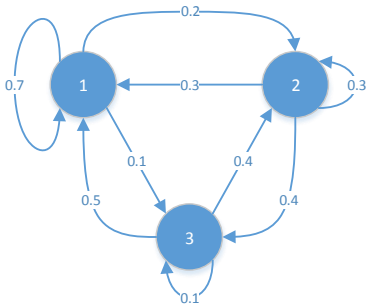
Gli Hidden Markov Models (HMM) sono dei modelli statistici che hanno numerose applicazioni nel riconoscimento di pattern temporali. Al momento, vengono impiegati con successo in bioinformatica, nel riconoscimento vocale, della scrittura e delle gesture, ed in generale in altri campi legati al machine learning.

Prima di introdurre il funzionamento di un Hidden Markov Model, è necessario definire il concetto di *catena di Markov*. Concettualmente, la si può vedere come una macchina a stati finiti in cui la transizione da uno stato all'altro avviene con una certa probabilità. In sintesi, una catena di Markov è processo stocastico che gode della *proprietà di Markov*, cioè, la probabilità di transizione da uno stato all'altro dipende solamente dallo stato immediatamente precedente, e non dall'intera storia degli stati passati. Questa proprietà viene anche chiamata *memorylessness*, per via del fatto che un processo markoviano non ha memoria.

Una catena di Markov a N stati può essere rappresentata tramite una matrice $A = \{a_{i,j}\}$ di dimensione $N \times N$. Ogni elemento della matrice $a_{i,j}$ corrisponde alla probabilità di transizione dallo stato i allo stato j. Dato che i valori rappresentano probabilità, la somma di ogni riga della matrice dev'essere

pari a 1, cioè $\sum_{j=1}^{N} a_{i,j} = 1$, e $0 \leq a_{i,j} \leq 1$. Ovviamente, uno stato può avere una certa probabilità di rimanere nello stesso stato in seguito alla valutazione delle transizioni.

Per completare il modello, è necessario definire anche la distribuzione di probabilità degli stati iniziali $\pi = (\pi_1, \pi_2, \ldots, \pi_N)$, dove π_i corrisponde alla probabilità di trovarsi nello stato i all'inizializzazione della macchina a stati. Logicamente, anche la somma delle probabilità π_i dev'essere pari a 1.



(a) Rappresentazione grafica

$$A = \begin{bmatrix} 0.7 & 0.2 & 0.1 \\ 0.3 & 0.3 & 0.4 \\ 0.5 & 0.4 & 0.1 \end{bmatrix}$$

$$\pi = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$$

(b) Parametri del modello

Figura 2.3: Markov chain composta da 3 stati.

Un Hidden Markov Model non è altro che una catena di Markov in cui non è possibile osservare direttamente lo stato corrente, per via del fatto che gli stati sono *nascosti*. Tuttavia, è possibile ottenere un'*osservazione* legata ad un certo stato. Ad ogni valutazione della macchina a stati, lo stato corrente produrrà un simbolo in uscita, che dipenderà da una distribuzione di probabilità (discreta o continua) caratteristica dello stato in oggetto. Ogni stato, quindi, può avere una distribuzione di probabilità diversa per l'emissione dei simboli.

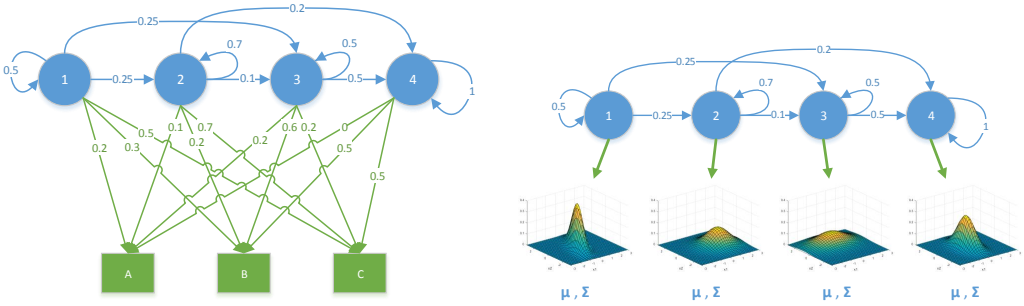
Di seguito verrà descritto il caso più semplice: gli Hidden Markov Models discreti. Questi ultimi possono produrre un insieme finito di simboli in uscita, e la probabilità di emettere un certo simbolo è descritta da una distribuzione di probabilità discreta. Un HMM discreto può essere descritto interamente con i seguenti parametri:

- Un insieme di N stati.
- Una matrice di transizione $A = \{a_{i,j}\}$.
- Una distribuzione di probabilità degli stati iniziali $\pi = (\pi_1, \pi_2, \ldots, \pi_N)$.
- Un insieme di M simboli distinti in uscita (osservazioni).

- Una matrice di emissione $B = \{b_{i,j}\}$ di dimensione $N \times M$, dove $b_{i,j}$ corrisponde alla probabilità di emettere il simbolo j trovandosi nello stato i. Anche qui, la somma delle probabilità di ogni riga dev'essere pari a 1.

In modo analogo agli HMM discreti, è possibile definire gli HMM continui, che sono basati su distribuzioni di probabilità continue per generare le osservazioni in uscita. In questo caso, sono assenti l'insieme degli M simboli e la matrice di emissione B, e vengono sostituiti dalla *funzione densità* della distribuzione continua che si vuole utilizzare. Solitamente, in assenza di assunzioni sui dati, si usa la distribuzione normale, quindi per ogni stato si definiscono i parametri μ e σ.

Gli HMM continui sono ideali per modellare quei processi che per loro natura sono continui. Inoltre, permettono di trattare anche distribuzioni multivariate, cioè, ad ogni transizione di stato viene emesso un vettore di osservazioni, e non un valore scalare. Nel caso delle distribuzioni normali multivariate, si rende necessario associare, ad ogni stato, un vettore $\boldsymbol{\mu}$ (le medie di ogni dimensione) ed una matrice di covarianza $\boldsymbol{\Sigma}$. Gli HMM continui multivariati sono ideali per trattare dati vettoriali, come le letture di accelerometri a 3 assi o giroscopi. Se si vogliono utilizzare questi dati su un HMM discreto, è necessario effettuare una conversione, quantizzando i dati in M simboli discreti con un processo noto come *Vector Quantization* (VQ). Esistono varie tecniche per ottenere questo risultato: è possibile farlo manualmente, ad esempio suddividendo il dominio dei dati in più intervalli ed assegnando un simbolo a ciascuno di essi, oppure si può utilizzare un algoritmo di clustering (come *k-means*) a partire da un dataset.



(a) HMM Discreto (b) HMM Continuo Multivariato

Figura 2.4: Comparazione visiva dei due tipi di Hidden Markov Models descritti. A sinistra, si ha un HMM discreto con $N = 4$ stati e $M = 3$ simboli (in verde). Le probabilità sugli archi verdi rappresentano la distribuzione delle emissioni. A destra, si ha un HMM continuo, in cui le emissioni di ogni stato sono rappresentate da una distribuzione normale multivariata a 2 dimensioni.

Gli usi degli Hidden Markov Models sono molteplici, ma nel contesto del machine learning vengono sfruttati per la classificazione di serie temporali. Data una sequenza di osservazioni $\mathbf{O} = (o_1, o_2, \ldots, o_n)$ ed un HMM λ noto, è possibile ottenere un valore che descrive la probabilità che tale sequenza sia stata generata dal modello in esame. Questo valore, denominato *likelihood* (verosomiglianza), è definito come $P(\mathbf{O}|\lambda)$ ed è possibile calcolarlo con un algoritmo chiamato *forward algorithm*. Questo algoritmo, inoltre, restituisce lo stato corrente più probabile.

In alternativa, se si desidera conoscere la sequenza di stati più probabile (*Viterbi path*), è possibile utilizzare l'*algoritmo di Viterbi*, che viene implementato tramite la tecnica della programmazione dinamica. Anche quest'algoritmo restituisce un valore di likelihood. La differenza sostanziale tra i due valori sta nel fatto che l'algoritmo forward restituisce la likelihood lungo tutte le possibili sequenze di stati, mentre l'algoritmo di Viterbi restituisce la likelihood lungo la sequenza di stati più probabile.

2.2.1 Costruzione di un Hidden Markov Model

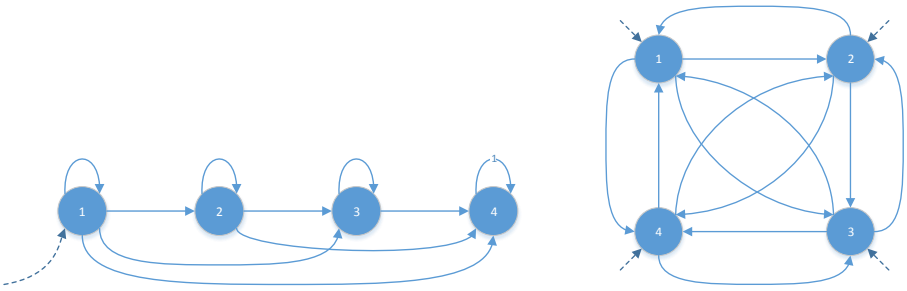
Nella panoramica precedente si è supposto di avere a disposizione un modello già costruito, ma nella pratica, nel contesto del machine learning, ciò che si vuole fare è costruire un modello a partire da un dataset, cioè da un insieme di sequenze di osservazioni. Per ottenere questo risultato, esistono vari algoritmi di apprendimento, che devono essere scelti in base all'applicazione e ai dati che si hanno a disposizione.

Come prima cosa, è necessario decidere il numero di stati del modello e la sua *topologia*. Per quel che riguarda il numero di stati, non esiste una regola precisa; si può scegliere un valore che si crede ragionevole per l'applicazione, oppure si possono fare più tentativi per trovare il numero di stati che massimizza la qualità del sistema. In genere, sono sufficienti pochi stati (5-10). Per topologia, invece, si intende l'insieme delle transizioni ammesse dal modello. Per inibire una transizione dallo stato i allo stato j è sufficiente porre il coefficiente $a_{i,j} = 0$ nella matrice di transizione A. Anche se è possibile costruire topologie personalizzate, nel campo degli HMM esistono due topologie standard:

- **Left-to-right (o Forward):** sono ammesse solamente le transizioni in avanti, e non indietro. In altri termini, dallo stato i si potrà passare allo stato j solo se $j \geq i$. Un modello left-to-right si può costruire ponendo $a_{i,j} = 0 \quad \forall i > j$, ottenendo così una matrice triangolare. Questa topologia viene utilizzata nelle sequenze temporali che rispettano un ordine ben preciso, come le gesture o la voce. Nel primo caso, si può segmentare la gesture in tante parti (una per ogni stato), e data la natura sequenziale

di quest'ultima, la sequenza deve seguire gli stati nell'ordine corretto. Nel caso del riconoscimento vocale, si può pensare ad esempio di creare un modello per una parola, creando uno stato per ogni fonema. Solitamente, si impone anche che $\pi_1 = 1$ e $\pi_i = 0 \quad \forall i > 1$, per modellare il fatto che la sequenza deve iniziare nel primo stato. Inoltre, esistono anche varianti di questa topologia, che limitano il numero di transizioni in avanti ad una certa profondità d (con il caso limite di $d = 1$, cioè permettendo solamente le transizioni da uno stato a quello immediatamente successivo). In generale, un modello left-to-right corrisponde ad un *grafo orientato aciclico* (DAG).

- **Ergodic:** tutte le transizioni sono ammesse; in questo modo, si ottiene un grafo totalmente connesso. Ciò significa che la sequenza di stati legata ad una serie di osservazioni può contenere dei cicli. Questa topologia è ideale per modellare processi ciclostazionari, come il battito cardiaco [8] o fenomeni stagionali.



(a) Topologia left-to-right (b) Topologia ergodic

Figura 2.5: Comparazione tra le due topologie appena descritte. Le frecce tratteggiate indicano gli stati iniziali.

Una volta stabilito il modello, si può procedere alla fase di apprendimento. L'obiettivo dell'algoritmo di learning è il seguente: dato un insieme di sequenze di osservazioni (il *training set*) indipendenti fra loro

$$\mathbf{S} = \{\mathbf{O^{[1]}}, \mathbf{O^{[2]}}, \ldots, \mathbf{O^{[n]}}\} \tag{2.9}$$

si vogliono trovare i parametri di un Hidden Markov Model $\lambda = (A, B, \pi)$ che massimizzino la seguente funzione

$$\prod_{i=1}^{n} P(\mathbf{O^{[i]}}|\lambda) \tag{2.10}$$

In parole semplici, si vuole massimizzare la likelihood di tutte le sequenze. Dal punto di vista funzionale, ciò equivale a trovare i parametri del modello che permettano di descrivere nel modo più fedele possibile le sequenze del training set. Purtroppo, non esistono algoritmi che siano in grado di trovare tale massimo analiticamente, ma sono state sviluppate alcune procedure iterative che permettono di approssimare il modello ottimo.

Gli algoritmi di learning possono essere di tipo *supervised* o *unsupervised*, e nel contesto degli Hidden Markov Models questi termini assumono un significato diverso dagli stessi riferiti al machine learning: nei primi, le varie sequenze devono già essere state etichettate con lo stato corrente ad ogni osservazione, e questo semplifica enormemente la fase di learning, ma richiede informazioni che non sono sempre disponibili. Nei secondi, non si conoscono gli stati associati alle osservazioni, e quindi si lascia che sia l'algoritmo a trovare una corrispondenza ragionevole.

- **Algoritmo di Baum-Welch:** questo algoritmo fa parte della categoria *unsupervised*. Partendo da un modello iniziale (una stima), l'algoritmo esegue iterativamente una procedura chiamata *forward-backward* e converge in modo garantito ad un *massimo locale* della likelihood. Dato che i parametri trovati corrispondono ad un massimo locale, e non globale, la stima iniziale del modello può influenzare fortemente il risultato. Tuttavia, nella pratica (e specialmente in assenza di dati aggiuntivi), i parametri del modello vengono inizializzati casualmente o uniformemente, in quanto l'algoritmo trova una soluzione soddisfacente nella maggior parte dei casi. Negli HMM discreti la scelta dei parametri iniziali è trascurabile, mentre in quelli continui (specialmente se multivariati) è fondamentale scegliere dei parametri ragionevoli per evitare che la procedura converga ad una soluzione inaccettabile [9].

- **Maximum-likelihood estimation (MLE):** algoritmo *supervised* che, come già accennato, considera sia la sequenza di osservazioni che la sequenza degli stati associati a tali osservazioni. MLE non soffre dei problemi di località dell'algoritmo di Baum-Welch, ma richiede informazioni aggiuntive (gli stati).

- **Viterbi training:** *(da non confondere con l'algoritmo di Viterbi per trovare la sequenza di stati più probabile)* questo algoritmo, noto anche come *Segmental K-Means* [10], rientra nella categoria *semi-supervised*, cioè si basa su un approccio ibrido tra i due descritti sopra. La procedura è la seguente: si utilizza l'algoritmo di Viterbi per trovare il Viterbi path (la sequenza di stati più probabile) basandosi sul modello corrente, e si applica MLE su questa sequenza di stati. Questo procedimento

viene ripetuto iterativamente per ottenere dei parametri sempre più rifiniti. Viterbi training è un algoritmo molto veloce dal punto di vista prestazionale, ma produce dei risultati molto approssimativi rispetto a Baum-Welch. Per questo motivo, viene spesso usato in un contesto semi-supervised: avendo a disposizione un ampio training set, si selezionano alcuni campioni di sequenze e se ne etichettano gli stati manualmente; successivamente, si usa un algoritmo di tipo supervised come MLE per stimare un modello iniziale, e si adotta Viterbi training per rifinire il modello tramite le sequenze rimanenti. Ad esempio, supponendo di avere a disposizione un enorme dataset di registrazioni vocali, se ne può scegliere un piccolo numero su cui etichettare i fonemi in ogni punto del segnale (ipotizzando che ad ogni stato corrisponda un fonema), per poi applicare la procedura appena descritta.

2.2.2 Costruzione di un classificatore

Per eseguire la classificazione di sequenze, non è sufficiente avere a disposizione un singolo Hidden Markov Model, bensì è necessario costruire un classificatore. Fortunatamente, il procedimento è molto semplice: date N classi, si creano N modelli (uno per ogni classe), e se ne calcolano i parametri con uno degli algoritmi di apprendimento descritti precedentemente. Ovviamente, il training set di ogni modello sarà composto dalle sequenze destinate alla rispettiva classe.

A questo punto il classificatore è pronto, e per classificare una sequenza sconosciuta $\mathbf{O}$ si applica l'algoritmo *forward* o *Viterbi* su ognuno degli N modelli, ottenendo la likelihood associata ad ogni modello $\mathcal{L}_i = P(\mathbf{O}|\lambda_i)$. Tra tutte, si seleziona la classe c^* corrispondente al valore di likelihood più alto.

$$c^* = \arg\max_i P(\mathbf{O}|\lambda_i) \tag{2.11}$$

2.2.3 Vantaggi e svantaggi

Gli Hidden Markov Models vengono sfruttati con successo in innumerevoli applicazioni, anche grazie all'enorme flessibilità di costruzione dei modelli. Le problematiche degli HMM riguardano soprattutto la difficoltà della loro messa a punto (scelta della topologia e dei parametri iniziali), ma una volta ottenuto un modello ottimale, le prestazioni risultano eccellenti. Gli Hidden Markov Models, inoltre, non possono essere utilizzati per modellare processi che per loro natura non sono o non possono essere approssimati come processi markoviani.

Come già accennato, se si utilizza l'algoritmo di Baum-Welch per l'apprendimento dei modelli, la condizione iniziale influisce sulla soluzione ottenuta.

Se si dispone di un dataset molto vasto, nella maggior parte dei casi ciò non rappresenta un problema, ma se il dataset ha un numero molto limitato di campioni, è importante trovare una stima iniziale che permetta una corretta convergenza dell'algoritmo. In generale, gli Hidden Markov Models non sono già pronti all'uso, bensì richiedono una personalizzazione in base ai segnali che si stanno trattando; di conseguenza, non è possibile identificare una regola generale che ne garantisca il corretto funzionamento in tutti i casi.

2.3 Reti neurali

In questa sezione verranno discussi gli approcci basati su reti neurali per la classificazione di sequenze temporali. Le reti neurali artificiali (ANN) si ispirano al funzionamento delle reti neurali biologiche (cioè del nostro cervello), e ne modellano il funzionamento in modo semplificato. Il loro scopo non è quello di simulare il comportamento del cervello, bensì di gestire in maniera efficace i pattern del dominio di interesse tramite l'uso di tecniche approssimative, al contrario degli approcci descritti nelle sezioni precedenti, che sono prevedibili e definiti in modo rigoroso dal punto di vista matematico. Ovviamente, anche i modelli basati su reti neurali sono definiti in modo rigoroso per quel che concerne la loro struttura e gli algoritmi, ma il loro funzionamento interno (dal punto di vista logico) è sconosciuto, e per questo motivo una rete neurale viene considerata una scatola nera (*black box*).

In termini semplici, uno dei casi d'uso delle reti neurali è il seguente: dato un *training set* composto da coppie di input e output, si vuole allenare una rete neurale in modo che "comprenda" le relazioni fra i dati ed astragga un modello che sia in grado di trattare nuovi dati.

2.3.1 Reti neurali feed-forward

Uno dei modelli più semplici di rete neurale è il *multilayer perceptron* (MLP), composto da una serie di strati di neuroni connessi tra loro con una topologia *feed-forward*, ottenendo così un grafo senza cicli.

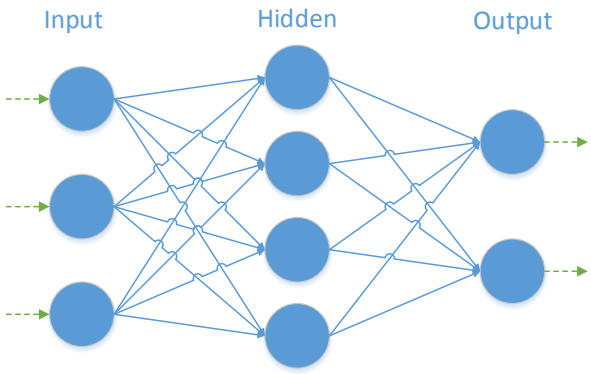


Figura 2.6: Multilayer perceptron a 3 strati

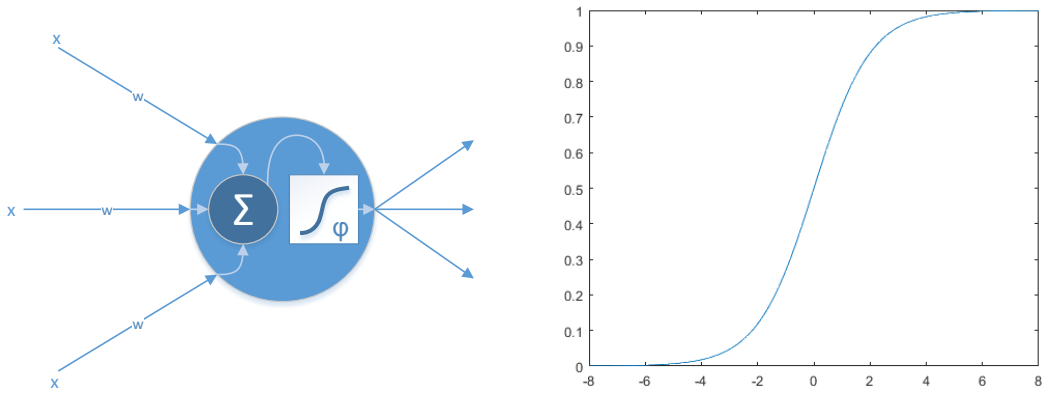
Tradizionalmente, il numero di strati è sempre stato basso. La suddivisione canonica è di 3 strati: input, nascosto (*hidden*) e output, ma è possibile sceglierne un numero arbitrario. In particolare, negli ultimi anni, le reti feed-forward stanno tornando ad essere un tema di ricerca promettente nell'ambito del machine learning, grazie al cosiddetto paradigma del *deep learning*. Esso consiste nel creare reti neurali molto profonde, cioè caratterizzate da un elevato numero di strati (superiore a 10), con lo scopo di far sì che ogni strato astragga una funzione diversa. Combinando queste funzioni assieme, si riescono a modellare sistemi molto complessi, che nella pratica non sarebbero realizzabili con reti di tipo *shallow* (cioè caratterizzate da pochi strati con molti neuroni). Infatti, nonostante in linea teorica le reti shallow siano in grado di approssimare qualsiasi funzione (per il teorema dell'approssimazione universale [11]), nella pratica ciò richiederebbe un numero proibitivo di neuroni. Al contrario, le *deep neural networks* (DNN) riescono a svolgere le stesse funzioni con una quantità minore di neuroni. Tuttavia, l'allenamento delle DNN presenta alcune problematiche di notevole importanza, che sono state superate solo negli ultimi anni, grazie allo sviluppo di nuove topologie di reti neurali e nuovi algoritmi di apprendimento. In ogni caso, questo argomento verrà descritto a breve.

Il modello di neurone artificiale è un'approssimazione grossolana di quello biologico, ed è definito in questo modo:

$$y = \varphi\left(\sum_{i=1}^{n} w_i x_i\right) \tag{2.12}$$

dove x_i sono i valori in uscita dai neuroni precedenti (e costituiscono gli input per il neurone corrente), w_i sono i pesi associati ad ogni arco (pesi sinaptici), y è il valore in uscita del neurone corrente, ed infine, φ è la funzione di attivazione del neurone. Quest'ultima, può essere specificata in modo differente per ogni

strato, e può essere lineare o meno. In genere, si utilizza la *funzione logistica* (o sigmoide), che descrive una funzione a gradino arrotondata, e si prefigge lo scopo di modellare le dipendenze input/output in maniera analoga ai neuroni biologici.



(a) Modello di neurone artificiale (b) Sigmoide

Figura 2.7: A sinistra, uno schema a blocchi di un neurone artificiale. A destra, la funzione di attivazione sigmoidale, definita come $1/(1+e^{-x})$

In un contesto *supervised*, l'algoritmo più utilizzato per allenare una rete feed-forward è *error backpropagation*. Si tratta di un algoritmo di ottimizzazione basato sulla *discesa del gradiente*, e si pone lo scopo di minimizzare una funzione obiettivo, che in genere è l'*errore quadratico medio* (MSE) degli output prodotti dalla rete neurale rispetto agli stessi del training set. In parole semplici, l'algoritmo fornisce i dati del training set in ingresso alla rete, ne misura l'errore in uscita, e modifica i pesi sfruttando il gradiente dell'errore, cioè l'insieme delle derivate parziali rispetto ad ogni peso. Ripetendo il procedimento iterativamente, si arriva ad un minimo locale. Come requisito, l'algoritmo richiede che la funzione di trasferimento sia derivabile.

Le reti neurali di tipo feed-forward hanno una natura fondamentalmente statica. Possono essere sfruttate per trattare problemi di classificazione di singoli *feature vectors* (nel caso del tema di questa tesi, *posture* al posto di *gesture*), ma non si prestano bene a gestire sequenze temporali. In passato sono state utilizzate per tale scopo (ad esempio il riconoscimento vocale), tramite complesse tecniche di preprocessing dell'input o l'uso di finestre temporali. Tuttavia, sono stati sviluppati degli approcci più moderni per trattare questi tipi di dati.

2.3.2 Reti neurali ricorrenti di Elman

Le reti neurali ricorrenti (RNN) sono un'estensione delle reti feed-forward, caratterizzate dalla presenza di cicli. Questa caratteristica permette loro di avere una sorta di memoria, e di poter esibire un comportamento dinamico, rendendole di fatto idonee a trattare sequenze temporali. Tra le loro applicazioni, c'è la possibilità di classificare queste ultime, oppure di eseguire predizioni di dati futuri (si pensi ad esempio ai suggerimenti di una tastiera per smartphone, durante la composizione di un messaggio).

Nel tempo sono state sviluppate numerose varianti di RNN. Quella descritta di seguito, nonché una delle più semplici, è quella di *Elman*. Concettualmente, la si può vedere come una rete feed-forward in cui lo strato nascosto, oltre ad essere connesso allo strato di output, si biforca in un altro strato identico, chiamato *strato di contesto*, a cui è connesso con pesi uguali a 1. Ad ogni istante di tempo (cioè ogni volta che vengono passati i dati ai neuroni dello strato di input), i neuroni dello strato di contesto mantengono i valori precedenti e li passano ai rispettivi neuroni dello strato nascosto.

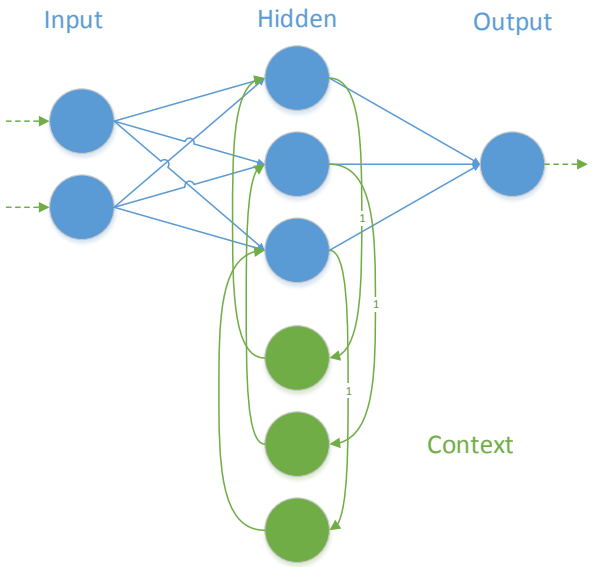


Figura 2.8: Rete neurale ricorrente di Elman a 3 strati

Analogamente alle reti feed-forward, le reti neurali ricorrenti di Elman possono essere allenate con un algoritmo chiamato *backpropagation through time* (BPTT), una variante del backpropagation creata appositamente per le RNN. Sostanzialmente, questo algoritmo "srotola" la rete neurale trasformandola in una rete feed-forward, dotata di un numero di strati pari alla lunghezza della sequenza da apprendere; successivamente, viene applicato l'algoritmo backpro-

pagation classico. Alternativamente, è possibile utilizzare metodi di ottimizzazione globale, come gli *algoritmi genetici*, specialmente con topologie di RNN su cui non è possibile applicare BPTT.

Le RNN classiche, tuttavia, soffrono di un problema di notevole importanza. Nella pratica, non sono in grado di gestire le dipendenze a lungo termine all'interno delle sequenze, bensì solamente quelle a breve termine. In altri termini, hanno un contesto molto limitato. Ciò non è dovuto alla topologia di rete in sè, bensì ad una limitazione degli algoritmi di apprendimento basati sulla discesa del gradiente (come backpropagation e BPTT). Questo fenomeno è strettamente connesso alle difficoltà di allenamento delle *deep neural networks*: infatti, concettualmente, una volta che una RNN è stata srotolata dall'algoritmo di apprendimento BPTT, può essere vista come una rete neurale molto profonda, che soffre quindi delle stesse problematiche legate al *deep learning*.

2.3.3 Deep learning: approcci e problematiche

Come già accennato, il *deep learning* è un approccio che consiste nell'utilizzare reti neurali molto profonde, cioè costituite da un numero elevato di strati. Ciò permette di creare sistemi più potenti, in quanto ogni strato può gestire un sottoproblema diverso dagli altri. Ad esempio, supponendo di voler allenare un rete per il riconoscimento delle immagini, essa potrebbe imparare ad estrarre i bordi nel primo strato, riconoscere forme geometriche semplici nel secondo, riconoscere oggetti complessi nel terzo, e così via.

Tuttavia, ciò che si osserva quando si allenano le reti feed-forward profonde con gli algoritmi classici (backpropagation), è che i primi strati (o gli ultimi) imparano velocemente, e quindi l'algoritmo ne modifica i pesi correttamente, mentre gli strati rimanenti rimangono pressoché invariati dopo un certo periodo di tempo. Maggiore è il numero degli strati, più evidente è il problema.

Questa instabilità degli algoritmi basati sulla discesa del gradiente è stata investigata per la prima volta da Hochreiter nel 1991 [12], ed in letteratura è conosciuta come *vanishing gradient problem*. Nei metodi basati su discesa del gradiente, come backpropagation, i pesi vengono aggiustati in modo proporzionale al gradiente dell'errore, e per via del modo con cui i suddetti gradienti sono calcolati, si ottiene l'effetto che il loro modulo diminuisce esponenzialmente procedendo verso gli strati più profondi.

Il problema del *vanishing gradient* affligge anche le reti neurali ricorrenti, che siano profonde o meno, proprio perché il *backpropagation through time* srotola la RNN, creando virtualmente una rete feed-forward molto profonda. L'impossibilità di avere un contesto a lungo termine da parte delle RNN è dovuto proprio a questo fenomeno: se il gradiente svanisce nell'arco di po-

chi strati, la rete non sarà in grado di imparare relazioni ad elevata distanza temporale tra i dati.

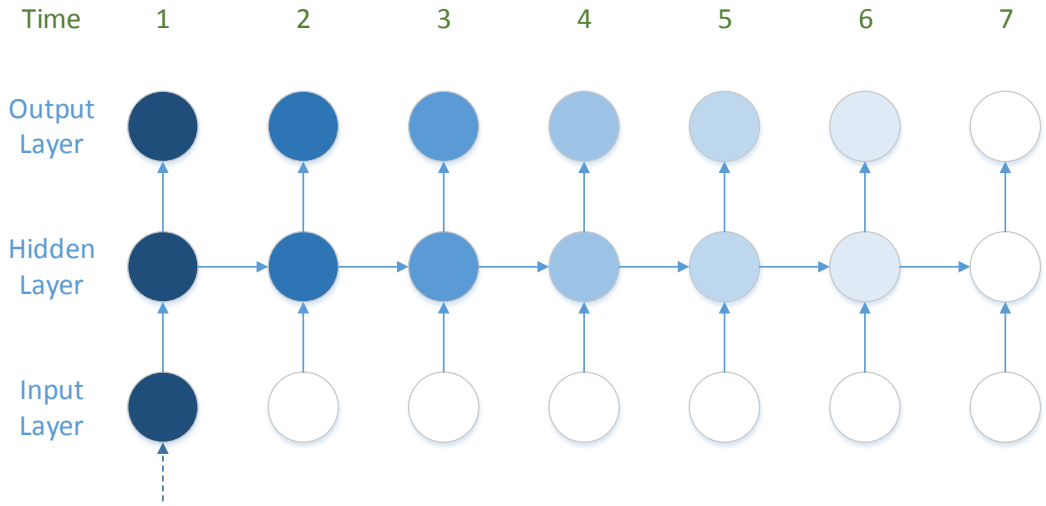


Figura 2.9: Problema del vanishing gradient.

Per superare questo problema, sono stati sviluppati nuovi approcci e nuove tipologie di reti neurali. Una possibilità consiste nella sostituzione della funzione di attivazione dei neuroni degli strati nascosti. Infatti, il problema del vanishing gradient è amplificato dall'uso di funzioni di tipo sigmoidale, che hanno un codominio molto ristretto (nello specifico caso della sigmoide, si ha $\mathbb{R} \to (0, 1)$). Una possibile scelta è la cosiddetta *Rectified Linear Unit* (ReLU) [13], la cui funzione di attivazione è definita come $y = \max(0, x)$. L'utilizzo di ReLU, unito all'aumento prestazionale dei calcolatori nel corso degli anni e alle implementazioni in ambito GPGPU degli algoritmi di learning, ha sicuramente mitigato il problema del *vanishing gradient*. ReLU viene adoperata con successo nelle *Convolutional Neural Networks* (CNN), che hanno applicazione nel riconoscimento delle immagini, e nella quasi totalità delle *Deep Neural Networks* [14].

Una tecnica per allenare le sopracitate *deep neural networks*, disponendo di un'enorme potenza di calcolo, consiste nell'utilizzare un *learning rate* molto ridotto per l'algoritmo di backpropagation, ed un numero di iterazioni molto elevato.

Come alternative ai suddetti approcci, esistono altre architetture di reti neurali, nonché altri algoritmi di apprendimento. In particolare, un'alternativa *deep* alle reti feed-forward classiche è rappresentata dalle *Deep Belief Network* (DBN), che consistono in una cascata di *restricted Boltzmann machines* (RBM) che possono essere pre-allenate singolarmente, per poi allenarle nel-

l'insieme. Un'alternativa all'algoritmo di backpropagation classico, inoltre, è *RProp* (Resilient backpropagation), che considera solo i segni delle derivate parziali, e non i moduli: per questo motivo, è meno suscettibile al problema del vanishing gradient [15].

In modo analogo, l'evoluzione delle reti neurali ricorrenti in ambito deep learning è rappresentata dall'architettura *long short-term memory*.

2.3.4 Long short-term memory

Long short-term memory (LSTM) è una particolare architettura di rete neurale ricorrente, originariamente ideata da Hochreiter e Schmidhuber nel 1997 [16]. Questa tipologia di rete neurale è stata riscoperta di recente nell'ambito del deep learning, perché è esente dal problema del *vanishing gradient*, e nella pratica offre risultati e prestazioni eccellenti.

Le reti basate su LSTM sono ideali per la predizione e classificazione di sequenze temporali, e stanno soppiantando molti approcci classici di machine learning. A sostegno di questa tesi, nel 2012 Google ha sostituito i suoi modelli di riconoscimento vocale, passando dagli Hidden Markov Models (che hanno rappresentato lo standard per oltre 30 anni) alle DNN, e nel 2015 è passata alle reti neurali ricorrenti LSTM abbinate a CTC [17]. La ragione di questa scelta è che le reti LSTM sono in grado di considerare le dipendenze a lungo termine fra i dati, e nel caso del riconoscimento vocale, ciò significa gestire il contesto all'interno di una frase per migliorare la capacità di riconoscimento.

Una rete LSTM è composta da celle (*LSTM blocks*) concatenate fra loro. Ogni cella è a sua volta composta da 3 tipi di porte: *input gate*, *output gate* e *forget gate*, che implementano rispettivamente le funzioni di scrittura, lettura e reset sulla memoria della cella. Le porte non sono binarie, bensì analogiche (generalmente gestite da una funzione di attivazione sigmoidale mappata in un intervallo $[0, 1]$, dove 0 indica la totale inibizione, e 1 indica la totale attivazione), e sono di tipo moltiplicativo. La presenza di queste porte, permette alle celle LSTM di ricordare le informazioni per un tempo indefinito: infatti, se l'*input gate* è sotto la soglia di attivazione, la cella manterrà lo stato precedente, mentre se è abilitato, lo stato corrente verrà combinato con il valore in ingresso. Come suggerisce il nome, il *forget gate* resetta lo stato corrente della cella (quando il suo valore viene portato a 0), e l'*output gate* decide se il valore all'interno della cella dev'essere portato in uscita o meno.

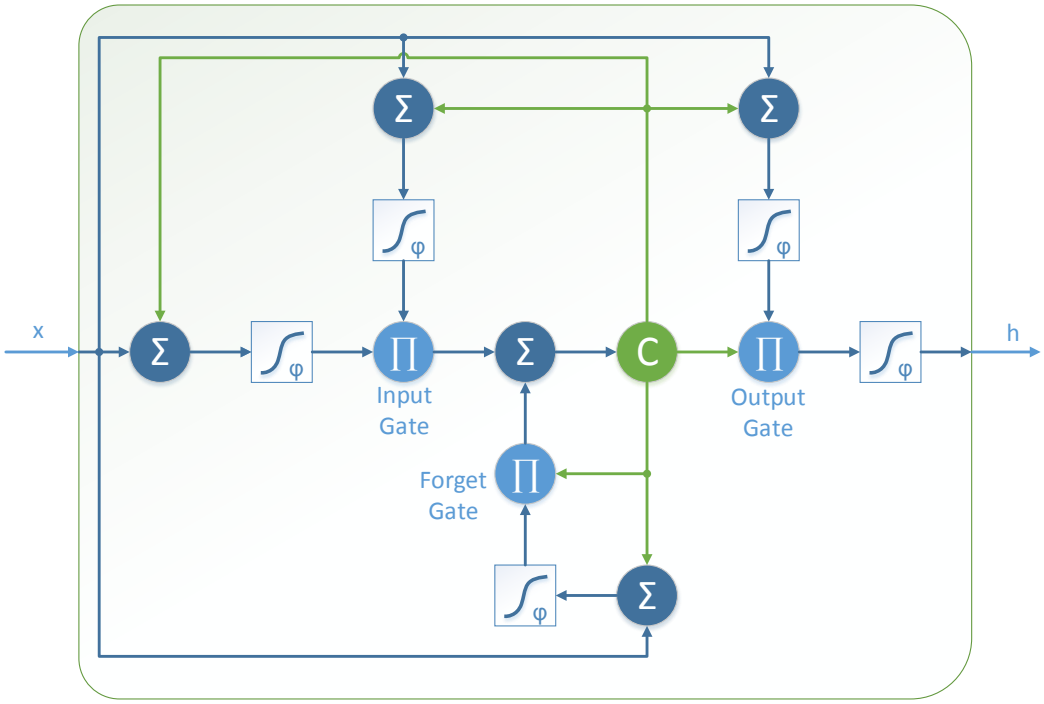


Figura 2.10: Schema a blocchi di una cella LSTM. I nodi di tipo Σ rappresentano una sommatoria pesata, mentre quelli di tipo Π rappresentano una moltiplicazione non pesata. C è la memoria corrente della cella, che verrà propagata all'istante di tempo successivo.

2.3.5 Connectionist Temporal Classification

Per eseguire la classificazione di sequenze temporali, non è sufficiente utilizzare le reti LSTM così come sono. Infatti, tutte le reti neurali ricorrenti producono un vettore in output ogni volta che ne ricevono uno in input, ma nel caso della classificazione, ciò che si vuole sapere è *quando* una classe valida viene trovata, e *quale*.

Nelle reti feed-forward classiche, per eseguire la classificazione di un feature vector, in genere si costruisce una rete dotata di N neuroni nello strato di output (uno per ognuna delle N classi), e si interpretano i valori di uscita come se fossero probabilità (assumendo che la funzione di trasferimento dello stato di uscita sia di tipo sigmoidale). Ciò significa che, nel caso di una classificazione perfetta, il neurone associato alla classe trovata produrrà un valore pari ad 1, mentre gli altri produrranno un output uguale a 0.

Nelle reti neurali ricorrenti, invece, il riconoscimento di una sequenza è un processo continuo che consiste nella raccolta di dati per un certo periodo di tempo, ed appena questi sono sufficienti, la rete può dare un responso.

Supponendo di avere a disposizione uno *stream* continuo di dati (ad esempio il segnale audio di un microfono nel caso del riconoscimento vocale, o i dati dei sensori nel caso del riconoscimento di gesture), la rete dovrebbe essere in grado di riconoscere un dato non appena questo è completo, e classificarlo. Nella pratica, ciò richiede complesse tecniche di *post-processing* dell'output della rete neurale; inoltre, è necessario allenare la rete con sequenze pre-segmentate.

Un approccio a questo problema è stato proposto da *Graves et al.* nel 2006, con una tecnica denominata *Connectionist Temporal Classification* (CTC) [18]. L'idea che sta alla base di CTC, è il fatto di interpretare gli output della rete neurale come una distribuzione di probabilità fra le varie classi. Di conseguenza, si può allenare la rete neurale con il classico *backpropagation through time*, impostando come funzione obiettivo la massimizzazione delle probabilità corrette. CTC riprende alcuni approcci ibridi basati su reti neurali ricorrenti e Hidden Markov Models, e li evolve, dando luogo ad un'interfaccia di input/output da applicare alle reti neurali esistenti, come LSTM. Inoltre, CTC permette di allenare le reti con sequenze non segmentate, e ciò rappresenta un vantaggio di notevole importanza per i campi come il riconoscimento vocale o il riconoscimento di gesture.

Il funzionamento di una rete CTC è molto intuitivo: ad ogni istante, essa produce in uscita un vettore di probabilità, con dimensione pari al numero di classi. Ogni volta che viene riconosciuta una classe, la rete produce un picco di probabilità nel rispettivo valore, che scompare poco dopo. Può anche essere prevista una classe aggiuntiva di default, che indica il *non riconoscimento*.

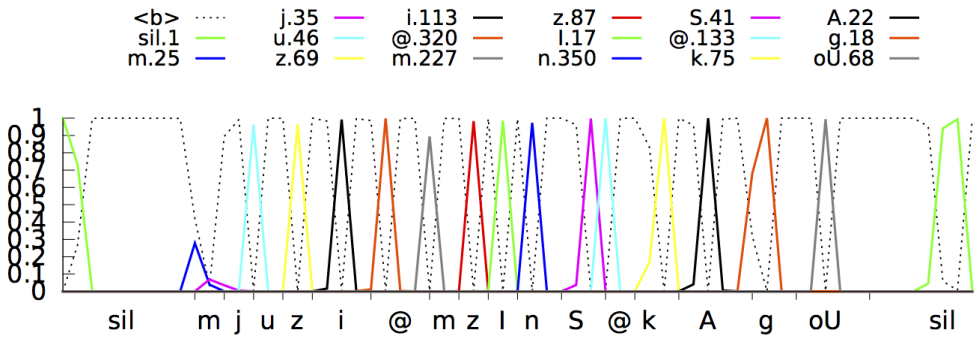


Figura 2.11: Funzionamento di CTC nel sistema di riconoscimento vocale di Google. Ad ogni istante di tempo la rete fornisce la probabilità di riconoscimento di ogni fonema.

2.3.6 Vantaggi e svantaggi

Gli approcci basati su reti neurali sono molto potenti, in quanto permettono di catturare le caratteristiche e le relazioni fra i dati. In particolare, si è anche visto che nella pratica le reti LSTM offrono prestazioni elevate ed ottimi tassi di riconoscimento. Uno svantaggio già descritto, è che le reti neurali sono dei modelli *black box*, quindi il loro comportamento non è predicibile, e non è possibile risalire alla logica con cui elaborano i dati. Inoltre, le reti neurali hanno bisogno di un training set mediamente più corposo rispetto agli approcci descritti precedentemente (DTW e HMM). Se il training set è insufficiente, può insorgere un problema noto come *overfitting*, che è tipico delle reti neurali (ma può interessare anche altre tecniche di machine learning). Ciò che avviene in questi casi, è che la rete "impara a memoria" i dati del training set, senza riuscire a generalizzarne le relazioni, per via dell'assenza di sufficienti informazioni. Di conseguenza, le prestazioni del modello diventano pessime, dato che si comporta correttamente sui dati del training set, ma non è in grado di gestire nuovi dati. Un altro problema che può causare l'overfitting, è la sovrastima della complessità della rete neurale: infatti, creando una rete composta da un numero eccessivo di neuroni, saranno presenti troppe variabili libere (pesi) rispetto ai dati del training set. In generale, bisognerebbe cercare di utilizzare il minor numero di neuroni che permetta di modellare con successo l'applicazione.

Un altro modo per evitare il problema dell'overfitting è di adottare un approccio di *cross validation*. Esso consiste nel dividere il dataset in un *training set* ed un *validation set*. Nel caso delle reti neurali, il primo verrà utilizzato esclusivamente per l'allenamento della rete, mentre il secondo verrà utilizzato per verificare l'errore durante la fase di allenamento, ma non prenderà parte ad esso in modo diretto. Se ad una certa iterazione l'errore sul validation set inizia ad aumentare, significa che sta insorgendo un caso di overfitting, ed è opportuno terminare la fase di training. La tecnica appena descritta prende il nome di *early stopping*.

Capitolo 3

Analisi e progettazione

3.1 Requisiti

Lo scopo di questo progetto è la creazione di un *framework* per riconoscimento in tempo reale dei gesti delle mani (*gesture*). Il framework non dovrà essere destinato ad un'applicazione in particolare, bensì sarà concepito per essere *general-purpose* ed utilizzabile come interfaccia di input per qualsiasi sistema, tramite una libreria.

Il sistema di riconoscimento sarà di tipo *user-dependent*, cioè concepito per poter essere utilizzato da una persona sola. Ciò non significa che il sistema non possa essere utilizzato da altre persone, bensì indica che, prima dell'utilizzo vero è proprio, sarà necessario eseguire una fase di calibrazione (raccolta dati) per creare un profilo su misura per l'utente. Questo approccio è il contrario del concetto di *user-independent*, che consiste in un sistema che sia in grado di funzionare correttamente con qualsiasi persona, senza alcun tipo di calibrazione. Nella pratica, creare sistemi *user-independent* è molto difficile perché richiede la raccolta di un'enorme quantità di dati, e spesso è necessario trovare un compromesso tra il tasso di riconoscimento e l'adattabilità a persone diverse.

Inoltre, è già stato anticipato che il dispositivo wearable utilizzato come fonte per il riconoscimento è Myo di Thalmic Labs. Ciò significa che questo progetto, e la fase di studio associata, si concentreranno sull'analisi ed elaborazione di segnali inerziali (provenienti dall'IMU del dispositivo) e mioelettrici (provenienti dai sensori EMG). Il riconoscimento delle gesture, quindi, dovrà avvenire esclusivamente sulla base dei suddetti segnali, e non su altri dati (ad esempio, flussi video).

Un buon sistema di riconoscimento di gesture, dovrebbe soddisfare i seguenti requisiti:

- **Riusabilità:** sebbene il sistema sia stato concepito per essere utilizzato in abbinamento a Myo, le tecniche utilizzate sono valide anche per altri dispositivi analoghi (cioè dotati di IMU e/o sensori EMG), quindi il sistema sarà progettato in modo da essere il più generico possibile. Ovviamente, dato che le caratteristiche elettriche dei sensori variano tra dispositivi di costruttori diversi (e probabilmente anche la parte del corpo su cui essi vengono indossati), potrebbe essere necessario ritoccare i parametri del sistema per farlo funzionare al meglio.

- **Reattività:** dato che il framework sarà utilizzato in un contesto *real-time*, il sistema di riconoscimento dovrà essere il più reattivo possibile. In altri termini, il tempo di latenza tra il momento in cui viene eseguita una gesture, ed il momento in cui questa viene riconosciuta, dovrà essere minimo ed impercettibile da parte dell'utente.

- **Prestazioni:** il sistema dovrà essere sufficientemente performante da funzionare in tempo reale anche su hardware con limitate capacità di calcolo, richiedendo il minimo indispensabile di risorse. Idealmente, il framework di riconoscimento potrebbe essere integrato su un dispositivo embedded (ad esempio sullo stesso dispositivo wearable che contiene i sensori), ed in questo contesto, prestazioni maggiori si traducono direttamente in un minore consumo di energia (dato che diminuisce il tempo di lavoro della CPU).

- **Affidabilità:** logicamente, il framework dovrà esibire un ottimo tasso di riconoscimento delle gesture, idealmente prossimo al 100%. Tuttavia, ciò dipende fortemente dalla bontà dei segnali iniziali ottenuti dai sensori, nonché dalla qualità dei dati raccolti durante la fase di calibrazione.

- **Facilità d'uso:** il sistema dovrebbe essere congegnato in modo da risultare relativamente semplice da utilizzare da parte dell'utente. In particolare, la fase di calibrazione dovrebbe richiedere il minimo tempo indispensabile.

Per quel che concerne l'utilizzo vero e proprio del framework da parte dell'utente, si può riassumere il procedimento nei seguenti passaggi:

1. *(solo la prima volta)* Il framework richiede che venga effettuata la calibrazione.

2. *(solo la prima volta)* L'utente effettua la calibrazione, eseguendo ogni gesture prevista dal sistema un certo numero di volte.

3. Il framework legge i dati in tempo reale dai sensori, in attesa che venga eseguita una gesture.

4. Ogni volta che viene riconosciuta una gesture, il sistema produce un evento che verrà gestito dall'applicazione che sfrutta il framework.

3.2 Analisi preliminare

Come già accennato più volte, per effettuare il riconoscimento delle gesture, il framework si servirà di tecniche di *machine learning*. Gli approcci candidati al riconoscimento sono del medesimo tipo di quelli riportati nel capitolo 2, cioè idonei a classificare sequenze temporali. Tuttavia, data la vastità dei possibili approcci, è impossibile sapere a priori quale sarà quello che darà i risultati migliori. Per questo motivo, è necessario eseguire una fase di *analisi* in cui si considera un insieme di possibili tecniche di machine learning. Successivamente, sulla base di motivazioni teoriche fondate, se ne seleziona un piccolo insieme, e si esegue una fase di *sperimentazione* per trovare la tecnica ottimale fra i possibili candidati.

3.2.1 Piano strategico

Per una corretta scelta dell'approccio definitivo, e la valutazione effettiva della sua qualità, è necessario elaborare un piano strategico generale per l'analisi e la progettazione del sistema. Di seguito si riporta la strategia che verrà utilizzata per lo svolgimento di questo progetto, che si può riassumere sinteticamente nei seguenti passi:

1. **Studio di fattibilità:** si deve valutare se il sistema è realizzabile o meno.

   (a) **Raccolta dati:** si raccoglie un piccolo campione di dati su cui effettuare un'analisi preliminare.

   (b) **Analisi dei dati:** si analizza la natura dei segnali appena raccolti, valutando se hanno un sufficiente potere discriminante per effettuare il riconoscimento di gesture. Ciò verrà deciso sulla base delle caratteristiche dei segnali (come il rapporto segnale/rumore, o la risposta ad eventuali stimoli).

2. **Definizione delle gesture:** dovrà essere definito l'insieme delle gesture che il sistema sarà in grado di riconoscere.

3. **Definizione della modalità di raccolta dei dati (dataset):** dato che verrà raccolto un insieme di campioni (esempi) per ogni gesture, sarà necessario definire la modalità con cui questi verranno collezionati ed utilizzati.

4. **Raccolta dei dati**

5. **Scelta delle tecniche candidate**

6. **Sperimentazione sulle tecniche candidate:** per ogni tecnica, si cerca di sviluppare un modello regolato al meglio per il riconoscimento delle gesture.

7. **Valutazione delle tecniche:** si testano i vari modelli sviluppati nel punto precedente con una parte del dataset, e si sceglie quello che offre il tasso di riconoscimento migliore.

8. **Testing finale:** una volta scelto il modello finale, si effettua una verifica definitiva per confermare il corretto funzionamento del sistema.

3.3 Caratteristiche dei segnali

3.3.1 Natura dei segnali

Segnali mioelettrici (EMG)

Una gesture non è altro che un movimento eseguito con le mani, e dal punto di vista fisiologico corrisponde ad una serie di contrazioni di alcuni gruppi muscolari. Utilizzando una tecnica nota come *elettromiografia* (EMG), è possibile misurare indirettamente l'attività elettrica prodotta dalle nostre fibre muscolari, mediante dei sensori posti sui muscoli in oggetto.

Nello specifico, una contrazione muscolare ha origine in cellule nervose chiamate *motoneuroni*, che generano impulsi elettrici (potenziali d'azione) e li propagano verso le singole fibre muscolari. Quando l'intensità dello stimolo verso una determinata fibra muscolare raggiunge una certa soglia (*soglia di depolarizzazione*), essa si contrae producendo a sua volta un potenziale d'azione muscolare, ed emettendo un debole campo elettromagnetico, misurabile sotto forma di tensione elettrica. Di conseguenza, il segnale EMG misurato dal sensore corrisponde alla somma algebrica dei potenziali d'azione nel raggio rilevabile [19].

Esistono due tipi di elettromiografia: *fine-wire* e *surface*. Nel primo caso, un ago viene inserito direttamente nella fibra muscolare, permettendo di ottenere un elevato isolamento fra le varie fibre. In questo modo, è possibile

osservare un segnale molto pulito e di tipo impulsivo, che corrisponde ai singoli potenziali d'azione. Maggiore è la frequenza degli impulsi, maggiore è l'intensità della contrazione. Il secondo tipo, invece, utilizza elettrodi collocati sulla pelle. Sebbene non sia una tecnica invasiva, è molto meno precisa rispetto alla precedente, in quanto non permette di isolare le singole fibre muscolari. Realisticamente, ciò che si ottiene è un segnale che corrisponde alla somma dei contributi di tutte le fibre muscolari nelle vicinanze del sensore, assumendo uno spettro simile a quello del rumore bianco, e variando in ampiezza in relazione all'intensità dello stimolo.

Dato che i sensori utilizzati da Myo sono di tipo *Surface EMG*, ci si aspetta di rientrare nel secondo caso. A maggior ragione, la prima tecnica è molto utile in medicina per l'esecuzione di esami diagnostici, ma è impraticabile per il riconoscimento di gesture.

È molto importante considerare il fatto che i segnali EMG possono variare enormemente da persona a persona, per via di inevitabili differenze fisiologiche. Inoltre, all'interno della stessa persona, i segnali ottenuti dipendono fortemente dal collocamento dei sensori, e da altre variabili come la sudorazione (che altera la conduttività elettrica della pelle), l'affaticamento, e la forza muscolare.

Segnali inerziali

I segnali inerziali provengono da dispositivi denominati Inertial Measurement Unit (IMU), che sono dotati di un accelerometro, un giroscopio, e talvolta un magnetometro. In questa sezione ci si concentrerà sulle IMU a 9 assi, dotate cioè di tutti i componenti sopracitati, producendo per ognuno di essi una lettura nello spazio 3D euclideo (composta quindi dalle tre componenti x, y e z).

Accelerometro Fornisce un vettore 3D che rappresenta l'accelerazione istantanea su ognuno dei 3 assi. In genere, l'unità di misura è quella del Sistema Internazionale ($\mathrm{m/s^2}$), oppure in g (dove $1\,g = 9.81\,\mathrm{m/s^2}$, l'accelerazione di gravità sulla Terra). È importante notare che, se l'oggetto è fermo, il sensore riporterà un'accelerazione pari a $1\,g$ verso l'*alto*. Questo *offset* apparente è sempre presente, ed è dovuto al *principio di equivalenza* di Einstein. A seconda delle applicazioni, questo effetto potrebbe essere desiderabile o meno: nel caso del riconoscimento delle gesture, ad esempio, l'inclinazione del dispositivo potrebbe alterarne il riconoscimento. Un approccio *naïve* per rimuovere l'offset dell'accelerazione di gravità consiste nell'applicare un filtro passa alto sul segnale, con una frequenza di taglio molto bassa. Tuttavia, ruotando il dispositivo, l'accelerazione di gravità tornerà ad essere registrata e dovrà passare un certo periodo di tempo prima che venga nuovamente filtrata. Ciò significa che è

necessario scegliere la frequenza di taglio in modo da trovare un compromesso tra il filtraggio della gravità e l'eliminazione del segnale utile. Se si ha la certezza che il dispositivo non subirà rotazioni consistenti, questo approccio è praticabile. Un'alternativa migliore, se si dispone dell'orientamento assoluto del dispositivo, è di sottrarre $1\,g$ dalla direzione in cui esso è orientato, compensando così l'accelerazione di gravità.
Supponendo, ad esempio, di avere a disposizione un quaternione[1] unitario $\mathbf{q} = (q_w, q_x, q_y, q_z)$ che rappresenta l'orientamento del dispositivo, ed un vettore $\mathbf{a} = (a_x, a_y, a_z)^{\mathrm{T}}$ che rappresenta l'accelerazione ottenuta dal sensore, si può calcolare un vettore $\mathbf{a}' = (a'_x, a'_y, a'_z)^{\mathrm{T}}$ privo di gravità nel seguente modo:

$$R = \begin{bmatrix} 1 - 2{q_y}^2 - 2{q_z}^2 & 2(q_x q_y + q_w q_z) & 2(q_x q_z - q_w q_y) \\ 2(q_x q_y - q_w q_z) & 1 - 2{q_x}^2 - 2{q_z}^2 & 2(q_y q_z + q_w q_x) \\ 2(q_x q_z + q_w q_y) & 2(q_y q_z - q_w q_x) & 1 - 2{q_x}^2 - 2{q_y}^2 \end{bmatrix} \tag{3.1}$$

$$\hat{\mathbf{g}} = R\,\hat{\mathbf{z}} = \begin{bmatrix} 2(q_x q_z - q_w q_y) \\ 2(q_y q_z + q_w q_x) \\ 1 - 2{q_x}^2 - 2{q_y}^2 \end{bmatrix} \tag{3.2}$$

$$\mathbf{a}' = \mathbf{a} - g\,\hat{\mathbf{g}} \tag{3.3}$$

dove R è la matrice di rotazione corrispondente al quaternione $\mathbf{q}$ [20], $\hat{\mathbf{g}}$ è il *versore* dell'offset gravitazionale, $\hat{\mathbf{z}} = (0, 0, 1)^{\mathrm{T}}$ è l'asse verticale di riferimento, e g è uno scalare che rappresenta l'accelerazione gravitazionale di riferimento ($1g$ o $9.81 m/s^2$).

Giroscopio I giroscopi MEMS montati sulle IMU non sono dei veri e propri giroscopi (in senso stretto), bensì assumono il nome di *giroscopi a struttura vibrante*, e sono basati sulla forza di Coriolis. Questo tipo di giroscopio fornisce una lettura della velocità angolare (generalmente espressa in rad/s) del dispositivo, negli assi x, y e z. Di conseguenza, non è possibile risalire all'orientamento assoluto del dispositivo, ed integrando la velocità angolare si otterrebbe un valore troppo impreciso. È possibile fondere questo sensore con l'accelerometro, sfruttando l'offset descritto precedentemente per ottenere l'orientamento assoluto (supponendo che il dispositivo non si muova), oppure, utilizzando il magnetometro (se presente).

[1] Un quaternione è un'estensione dei numeri complessi rappresentabile come $q + xi + yj + zk$, dove vale la proprietà $i^2 = j^2 = k^2 = ijk = -1$. Tra le varie applicazioni, i quaternioni possono essere impiegati per rappresentare rotazioni nello spazio 3D, ed oltre ad essere molto performanti, non sono suscettibili al problema del *gimbal lock* (blocco cardanico) tipico degli angoli di Eulero.

Magnetometro Fornisce la lettura dell'intensità del campo magnetico nei 3 assi dello spazio 3D (convenzionalmente in μT). Solitamente, il valore che si vuole ottenere è quello del campo magnetico terrestre, che, in modo analogo ad una bussola, permette di ricostruire l'orientamento assoluto del dispositivo. Per quel che riguarda il riconoscimento delle gesture, i dati provenienti da questo sensore dovrebbero essere ignorati ai fini della classificazione, dato che una gesture dovrebbe poter essere riconosciuta in qualsiasi direzione (sia puntando verso nord che verso sud, per esempio). Tuttavia, è lecito utilizzare questo segnale per altri scopi, come quello di sottrarre l'accelerazione gravitazionale (se lo si ritiene necessario).

3.3.2 Analisi dei segnali campionati da Myo

Segnali EMG

Myo è dotato di 8 sensori *Surface EMG* (a secco) che avvolgono circolarmente l'avambraccio. Quest'ultimo è il gruppo muscolare ideale per il riconoscimento delle gesture, dato che è suddiviso in numerosi fasci muscolari che controllano il movimento delle dita (Figura 3.2).

Figura 3.1: Allocazione dei canali EMG di Myo.

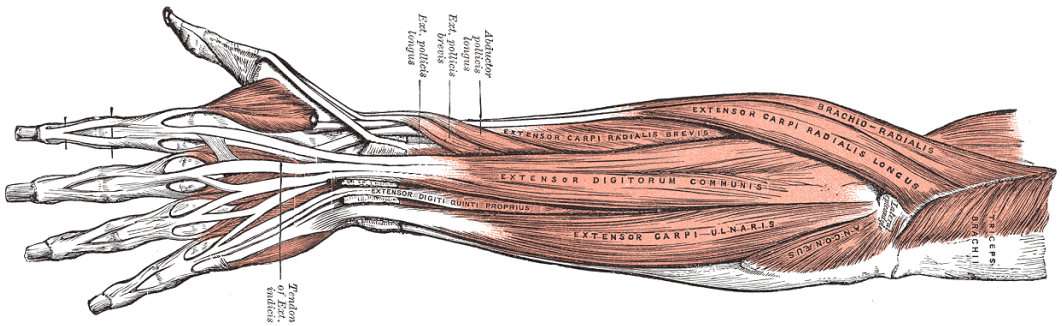
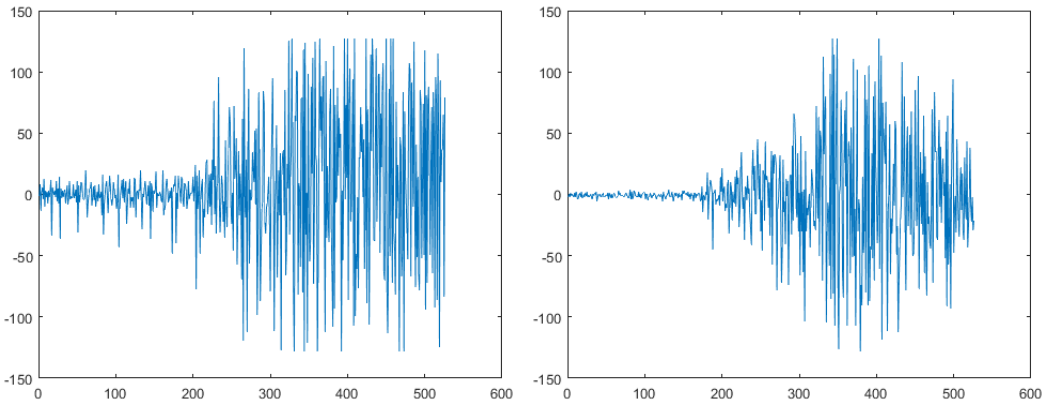


Figura 3.2: Muscoli dell'avambraccio.

I sensori EMG di Myo hanno una frequenza di campionamento di 200 Hz, e ciò significa che, secondo il teorema del campionamento di Nyquist-Shannon, la massima frequenza estraibile in modo affidabile dal segnale è pari a 100 Hz. Le letture vengono fornite come vettori di 8 numeri interi *signed* a 8 bit ciascuno, risultando in una *gamma dinamica* (la differenza tra la massima e la minima ampiezza rappresentabile) di circa 48 dB (cioè $20 \log_{10}(2^8)$). L'unità di misura rappresenta probabilmente una tensione elettrica, ma non conoscendo il guadagno degli amplificatori, non si può fare alcuna considerazione in merito alla scala. Tuttavia, ciò non rappresenta un problema per gli scopi di questo progetto.

I segnali provenienti dall'IMU, invece, sono campionati a 50 Hz (la massima frequenza rappresentabile è pari a 25 Hz) e vengono restituiti come vettori floating-point, dei quali non si conosce la reale risoluzione di quantizzazione da parte dell'ADC (convertitore analogico-digitale) interno al dispositivo. L'accelerazione è fornita in g (in ognuno dei 3 assi), e la rotazione in rad/s (in ognuno dei 3 assi).

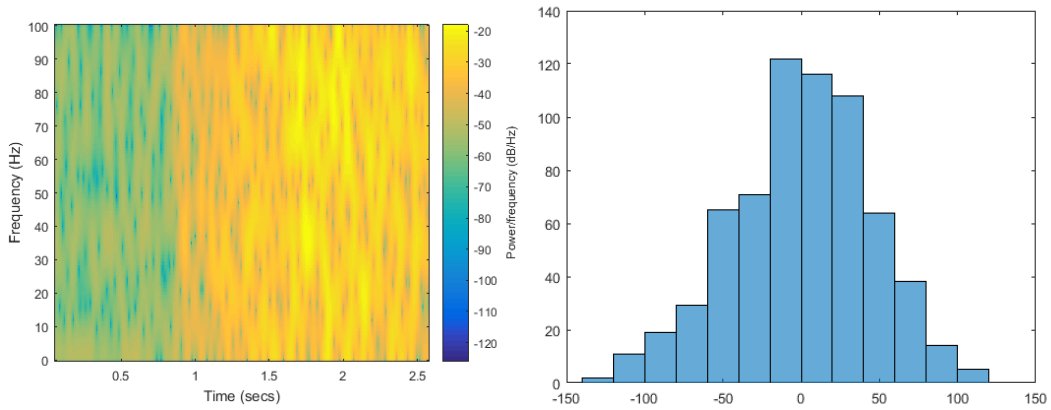
Un'analisi qualitativa dei segnali EMG, ha mostrato che le loro caratteristiche non sono uniformi tra gli 8 sensori, bensì variano leggermente, probabilmente per differenze di natura muscolare (e non elettriche). In relazione a varie misure effettuate su alcuni campioni (con il dispositivo indossato), si è visto che il rumore di fondo si assesta tra i $-42\,\mathrm{dB_{RMS}}$ ed i $-45\,\mathrm{dB_{RMS}}$. Si è anche valutato il rapporto segnale/rumore (SNR) sui singoli sensori, calcolato come il rapporto tra il valore efficace nel momento di massima contrazione muscolare, ed il valore efficace del rumore di fondo. In questo caso la variazione è consistente, spaziando da 12 dB a 23 dB. Si tratta di margini molto bassi per rilevare se un muscolo è contratto o meno, ma comunque sufficienti.



(a) Sensore 1 (b) Sensore 8

Figura 3.3: Differenza di rapporto segnale/rumore tra due sensori diversi. Inizialmente il braccio è a riposo, ed intorno al campione 200 inizia l'esecuzione di una gesture (pugno). Si può chiaramente notare che il sensore 8 presenta un segnale più pulito, nonostante sia adiacente al sensore 1.

Dato che molti sistemi di riconoscimento, come quello vocale, si basano sul contenuto del segnale nel dominio delle frequenze, si è deciso di effettuare un'analisi spettrale dei segnali EMG. Lo studio non ha fatto altro che confermare l'ipotesi avanzata nella sezione sezione 3.3.1, cioè che la distribuzione delle ampiezze di ciascuna frequenza nello spettro del segnale rimane pressoché invariata quando un muscolo si contrae, e nella fattispecie, sembra corrispondere ad un rumore bianco gaussiano a valore medio nullo.



(a) Spettrogramma (b) Distribuzione delle ampiezze

Figura 3.4: A sinistra, lo spettrogramma corrispondente al segnale mostrato precedentemente. Come si può notare, sebbene siano presenti delle variazioni, non sembra possibile estrarre alcuna informazione utile da esso. Il grafico è stato ottenuto tramite una *Short-Time Fourier Transform* (STFT), con una finestra di 20 campioni (100 ms) e *window function* di Hamming. A destra: un istogramma che mostra la distribuzione delle ampiezze di un segnale EMG con il muscolo sempre contratto, approssimabile ad una distribuzione normale.

Come conclusione, appare evidente come l'unica caratteristica del segnale utilizzabile per il riconoscimento delle gesture sia rappresentata dall'ampiezza.

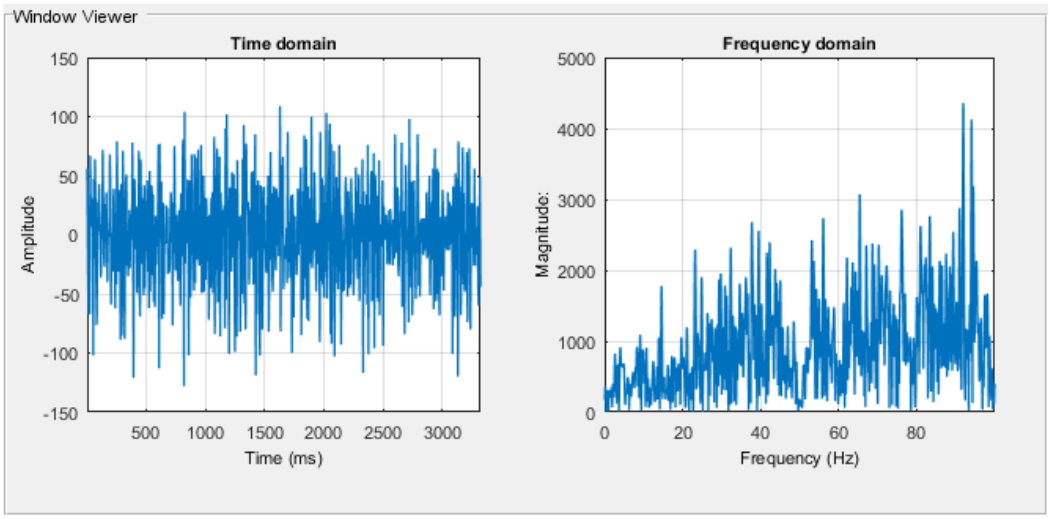
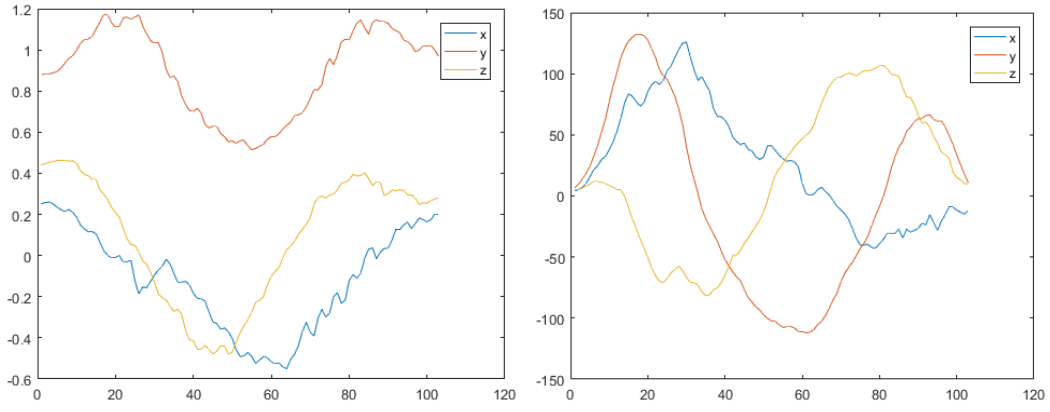


Figura 3.5: Analisi di un segnale EMG nel dominio del tempo e nel dominio delle frequenze. La densità spettrale è riconducibile a quella del rumore bianco. Si noti anche il picco negativo a 50 Hz, che è probabilmente dovuto ad un filtro elimina banda (*notch filter*) utilizzato per rimuovere la frequenza della rete elettrica, dato che gli amplificatori per questi tipi di segnali sono molto sensibili.

Segnali IMU

Fortunatamente, i segnali provenienti dall'IMU sono più puliti, e potrebbero essere anche utilizzati senza alcun tipo di filtraggio.



(a) Accelerometro (b) Giroscopio

Figura 3.6: Segnali provenienti dall'accelerometro e dal giroscopio, durante l'esecuzione della stessa gesture (un cerchio disegnato nell'aria).

3.4 Acquisizione del dataset

3.4.1 Tipi di gesture da riconoscere

Data la natura completamente differente tra i segnali EMG e quelli inerziali, è stato deciso di suddividere le gesture in due gruppi: *gesture statiche* (abbinate ai sensori EMG) e *gesture dinamiche* (abbinate all'IMU). Le prime sono correlate ai movimenti del polso e delle dita (cioè dalle parti controllate dalla muscolatura dell'avambraccio), e vengono denominate *statiche* perché sono indipendenti dal movimento del braccio (come l'esecuzione di un pugno o di una mano aperta, ad esempio). Il secondo gruppo, invece, contiene le gesture che rappresentano movimenti nello spazio (ad esempio disegni di figure geometriche nell'aria), e sono indipendenti dalla postura assunta dalla mano. L'aggettivo *statiche* non deve trarre in inganno: le gesture statiche, ovviamente, possono contenere movimenti che si evolvono nel tempo.

I due gruppi dovranno essere trattati con tecniche diverse fra loro: le gesture statiche, infatti, coinvolgeranno esclusivamente i dati dei sensori EMG, mentre le gesture dinamiche verranno riconosciute solamente sulla base dei segnali dell'accelerometro e del giroscopio. Combinando le due tecniche assieme (pur mantenendole indipendenti fra loro), sarà possibile riconoscere un numero ancora superiore di gesture, a patto che le gesture di un gruppo non influenzino i sensori dell'altro.

Gesture statiche

(a) Pugno (b) Carta (c) Forbici

(d) Destra (e) Sinistra (f) Destra + Sinistra

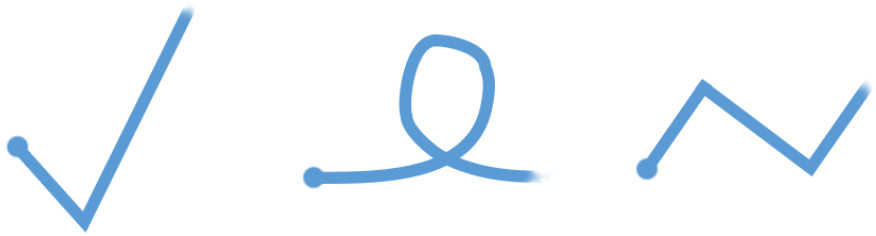
(g) Indice (h) Pollice in su (i) Morsa

(j) OK

Figura 3.7: Lista delle 10 gesture statiche. La gesture *Destra + Sinistra* è una combinazione tra *Destra* e *Sinistra* (eseguita con un movimento lineare), ed ha lo scopo di valutare il comportamento del classificatore con gesture che contengono segmenti di altre gesture.

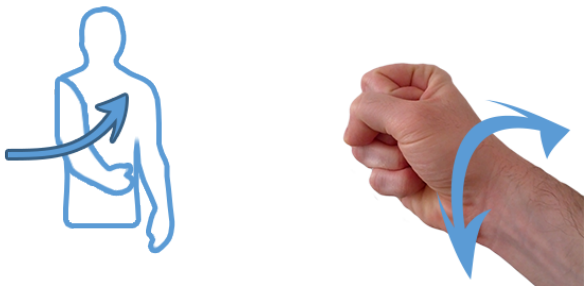
Gesture dinamiche

(a) Cerchio (b) Quadrato (c) Triangolo



(d) Check (e) Pigtail (f) Zig-zag

(g) Linea obliqua (h) Pugno sferrato (i) Rotazione esterna



(j) Rotazione interna (k) Rotazione polso

Figura 3.8: Lista delle 11 gesture dinamiche. Le gesture dalla (a) alla (g) rappresentano disegni di figure geometriche nello spazio, mentre le successive corrispondono a movimenti generici.

3.4.2 Collezione dei dati

Per garantire una corretta analisi delle tecniche di machine learning, nonché l'allenamento del modello che verrà adoperato, è necessario raccogliere un ampio dataset su cui eseguire lo studio. Inoltre, la fase di collezione dei dati dovrebbe avvenire con un piano preciso, e con modalità consistenti e riproducibili.

Per lo svolgimento di questo progetto, sono stati raccolti 120 campioni per ogni gesture statica e dinamica, per un totale di 2520 campioni. Ogni campione corrisponde ad una singola esecuzione di una gesture, eseguita a velocità naturale (nè lenta, nè veloce), e cercando di riprodurre il movimento nel modo più fedele possibile, pur mantenendo un minimo fisiologico di variabilità.

Le gesture statiche sono state eseguite partendo da una posizione di riposo, e tornando nuovamente alla posizione di riposo una volta completate, con un movimento lineare, e senza lasciare spazi vuoti non necessari (in modo da ottenere una segmentazione ottimale). Anche le gesture dinamiche sono state effettuate partendo da una posizione di riposo, però sono state interrotte subito dopo il completamento, in quanto il ritorno alla posizione iniziale avrebbe potuto essere fonte di potenziali problemi. Inoltre, le gesture dinamiche che rappresentano movimenti nello spazio sono state eseguite attenendosi alle illustrazioni, cioè partendo dall'angolo in basso a sinistra (se applicabile) e procedendo in senso orario.

Al fine di aumentare la variabilità del dataset, ed anche per permettergli di rappresentare casi più realistici (*session-independent*), i dati sono stati raccolti in sessioni diverse. Nello specifico, la fase di collezione è stata suddivisa in 4 giorni (distanziati di una settimana uno dall'altro), ottenendo così 30 campioni per gesture al giorno. All'interno di una stessa sessione, inoltre, il dispositivo è stato indossato e sfilato più volte. Si ricorda nuovamente che, essendo in un contesto *user-dependent*, i dati sono stati raccolti da una persona sola.

Per valutare il funzionamento del modello nel riconoscimento *real-time* di gesture, cioè partendo da un flusso continuo di segnali, sono stati catturati degli *stream* di dati, all'interno dei quali sono state eseguite varie gesture in uno specifico ordine. In modo analogo, sono stati raccolti anche degli stream *non-gesture*, che contengono movimenti casuali. Il loro scopo è di valutare il tasso di falsi positivi, cioè gesture riconosciute erroneamente (quando in realtà non sono mai state eseguite). In totale, la lunghezza delle registrazioni ammonta a circa 10 minuti, ed al loro interno sono state eseguite 210 gesture valide (10 per tipo).

Collocamento dispositivo

Un'altra variabile molto importante riguarda il collocamento del dispositivo, cioè il modo in cui viene indossato. In questo caso, si è cercato di mantenere la massima consistenza, indossando il dispositivo sempre nel braccio destro, e nella stessa posizione (a meno di una leggera variabilità). Inoltre, il dispositivo è stato orientato in maniera identica per tutte le sessioni, puntando il sensore 4 verso l'interno del gomito. Ciò permette di semplificare la creazione del modello, ma ha lo svantaggio di non funzionare correttamente nel caso il dispositivo venisse indossato in modo differente da quello originariamente previsto. Per superare questo problema, può essere prevista una brevissima fase di calibrazione ogniqualvolta il dispositivo viene indossato, indicando all'utente di eseguire una certa gesture direzionale che abbia un profilo distintivo (ad esempio *Destra*). In questo modo, trovando il sensore da cui si ottiene l'ampiezza massima del segnale, si può effettuare un allineamento *virtuale* dei sensori (facendo ruotare gli elementi del vettore). A maggior ragione, questo approccio viene utilizzato anche dal software ufficiale di Myo, che richiede di eseguire il *Sync* ogni volta che il dispositivo viene indossato. In ogni caso, l'argomento non rientra negli obiettivi di questo progetto, e pertanto non verrà considerato.

3.4.3 Suddivisione del dataset

Per garantire la massima affidabilità della fase sperimentale, si è deciso di adoperare la tecnica del *cross-validation*. Ne esistono molte varianti, ma quella utilizzata in questo progetto è la seguente: l'intero dataset è stato suddiviso in 3 gruppi: *training set*, *validation set* e *test set*, con delle proporzioni rispettivamente corrispondenti al 50%, 25% e 25% del totale. Ogni campione è stato assegnato casualmente ad uno dei gruppi sopracitati.

Ognuna delle tre categorie ha uno scopo preciso:

Training set È l'insieme su cui si effettua l'allenamento del modello (cioè la fase di training).

Validation set Una volta allenato il modello, se ne esegue la verifica con i dati di questo insieme. Ciò costituisce un ottimo metodo per validare il modello, dato che ne misura il comportamento con dati non ancora visti. Infatti, in generale non bisognerebbe mai eseguire la verifica con i parametri del *training set*, in quanto il risultato sarebbe quasi sempre positivo. La verifica con il *validation set*, inoltre, permette di rendersi conto della presenza o meno di un caso di *overfitting* del *training set*. Questo insieme può anche essere utilizzato per trovare la messa a punto ottimale degli *iperparametri* del modello.

Test set Dopo aver scelto il modello definitivo (ed i parametri associati), lo si congela e se ne effettua una verifica finale con i dati di questo insieme, le cui statistiche saranno quelle che faranno fede per valutare la qualità del sistema. Ciò ha lo scopo di evitare l'overfitting dei dati del *validation set*, in seguito alla messa a punto ottimale dei parametri del modello.

3.5 Tecniche candidate

Nel capitolo 2 sono state menzionate alcune tecniche di machine learning per la classificazione di sequenze temporali: *Dynamic Time Warping*, *Hidden Markov Models*, e reti neurali ricorrenti di tipo *Long Short-Term Memory* abbinate a CTC. Esistono anche altre tecniche, come le *Support Vector Machines* (SVM) ed i *(Hidden) Conditional Random Fields* ((H)CRF), che sono anch'essi approcci molto validi, ma che non verranno trattati in questa tesi.

Per le prime tre, verrà svolta un'analisi preliminare atta a valutare se sono utilizzabili o meno nel contesto di questo progetto:

- **Dynamic Time Warping**: data la sua semplicità di implementazione, la richiesta di una piccola quantità di template, ed il fatto che è già stato impiegato con successo nel riconoscimento delle gesture [21][22][23], si è deciso di includere DTW nella sperimentazione delle possibili tecniche.

- **Hidden Markov Models**: anche gli HMM sono già stati impiegati con successo nel riconoscimento delle gesture [24][25][26]. Questa tecnica, sebbene sia più complessa da gestire, e richieda una maggiore quantità di dati, è stata inclusa nella sperimentazione perché è considerata molto potente.

- **Long short-term memory**: le reti LSTM hanno lo svantaggio di richiedere un'enorme quantità di dati per evitare il problema dell'*overfitting*, ma probabilmente, se usate nel modo corretto, sono quelle che darebbero i migliori risultati tra le tecniche riportate. Tuttavia, per il motivo appena citato, è stato scelto di non utilizzarle in questo progetto. Inoltre, allo stato dell'arte attuale, le implementazioni di LSTM sono poco pratiche da utilizzare e richiedono un'enorme potenza di calcolo per la fase di *training*. Probabilmente, LSTM rappresenterebbe la scelta migliore nel caso di un contesto *user-independent*, in cui si potrebbe raccogliere un vasto dataset su più persone, ed allenare la rete in modo *offline*, per poi distribuire agli utenti il modello finale. In un contesto *user-dependent*, invece, è necessario adottare approcci più flessibili.

3.6 Estrazione features

3.6.1 Segnali EMG

Come si può facilmente intuire, i segnali mioelettrici *raw* (cioè ottenuti direttamente dalla fonte) non sono utilizzabili allo stato attuale, quindi è necessario effettuare un passaggio di *feature extraction* per esaltarne le caratteristiche e permetterne il confronto con altri segnali simili.

In questo contesto, gli approcci convenzionalmente utilizzati consistono nell'applicare una serie di filtri e trasformazioni sul segnale, che hanno lo scopo di estrarne l'inviluppo (*envelope*), cioè una curva che ne approssimi la forma. Il principio è del tutto equivalente alla demodulazione AM (cioè di segnali modulati in ampiezza) utilizzata in elettronica e telecomunicazioni.

In generale, per trovare l'inviluppo dei segnali EMG si può scegliere uno dei seguenti approcci [27]:

- Si raddrizza il segnale calcolandone il valore assoluto (*full-wave rectification*), in modo da ottenere un nuovo segnale $\mathbf{y} = |\mathbf{x}|$. Successivamente, si applica un filtro passa basso digitale su quest'ultimo, per livellarne i picchi. La scelta del filtro è molto importante, dato che stabilirà la forma finale dell'inviluppo.

- Si applica un filtro RMS (*root mean square*), che consiste nel calcolare il valore efficace (RMS) del segnale all'interno di una finestra temporale scorrevole, costruendo un nuovo segnale composto dai suddetti valori (centrati all'interno della finestra). Riassumendo, il segnale filtrato si può calcolare come segue:

$$y_i = \sqrt{\frac{1}{w} \sum_{k=i-\frac{w}{2}}^{i+\frac{w}{2}} {x_k}^2} \tag{3.4}$$

  dove w rappresenta la larghezza della finestra, $\mathbf{y}$ il segnale filtrato, e $\mathbf{x}$ il segnale originale. Si noti che la formula non è applicabile nei punti estremi del segnale, perciò è necessario gestire questo caso separatamente. Il filtro RMS ha due significati ben precisi:

  – Dal punto di vista statistico, se il segnale ha valore medio nullo (vero nei segnali EMG), il valore efficace corrisponde alla deviazione

standard dell'insieme dei campioni: infatti, si ha che

$$\mu_i = \frac{1}{w} \sum_{k=i-\frac{w}{2}}^{i+\frac{w}{2}} x_k \approx 0 \tag{3.5}$$

$$\sigma_i = \sqrt{\frac{1}{w} \sum_{k=i-\frac{w}{2}}^{i+\frac{w}{2}} (x_k - \mu_i)^2} \approx \sqrt{\frac{1}{w} \sum_{k=i-\frac{w}{2}}^{i+\frac{w}{2}} {x_k}^2} \tag{3.6}$$

che coincide con l'eq. (3.4). μ_i e σ_i sono rispettivamente la media e la deviazione standard della finestra centrata sul campione i-esimo. Ciò significa che il filtro RMS può essere utilizzato per stimare l'andamento della varianza all'interno del segnale, e dato che si è visto che quest'ultimo è distribuito normalmente (rumore bianco gaussiano), il filtro fornisce un'eccellente approssimazione del suo inviluppo.

– Dal punto di vista fisico/elettrico, il valore efficace corrisponde alla potenza media dissipata da una resistenza elettrica ai capi della quale viene applicato il segnale originale. Ciò si traduce in una diretta misura della sua intensità.

Invece, se si decide di applicare la prima tecnica (valore assoluto e filtro passa basso) è fondamentale scegliere un filtro passa basso adeguato allo scopo. Alcune ipotetiche scelte sono le seguenti:

- **Filtro Butterworth di primo ordine:** rappresenta uno dei filtri più semplici, conosciuto in elettronica come *filtro RC*. Si può dimostrare che, nel contesto dei filtri digitali, applicare un filtro passa basso Butterworth di primo ordine è equivalente ad applicare una media mobile esponenziale pesata (EWMA) [28], definita dalla seguente relazione di ricorrenza:

$$y_i = \alpha\, y_{i-1} + (1 - \alpha)\, x_i \tag{3.7}$$

dove α è una costante di smorzamento che determina la frequenza di taglio. In particolare, dato un segnale campionato ad una frequenza f_s, si può determinare la costante α in modo da ottenere la frequenza di taglio f_t nel seguente modo:

$$\alpha = e^{-\frac{1}{f_s \cdot \tau}} = e^{-2\pi \frac{f_t}{f_s}} \tag{3.8}$$

Nella teoria dei segnali, questo filtro appartiene alla categoria IIR (*Infinite Impulse Response*) per via della sua natura ricorsiva, e ciò significa

che il valore corrente dipende dall'intera storia dei valori passati. La sua implementazione è estremamente semplice e performante, ma i filtri di questo tipo hanno lo svantaggio di causare uno sfasamento del segnale con il variare della frequenza (cioè, certe frequenze subiscono un ritardo maggiore di altre). Questa caratteristica è nota come *distorsione di fase*, e dal punto di vista qualitativo causa un'alterazione della morfologia del segnale. Per questo motivo, il filtro non rappresenta la scelta migliore per estrarre le features dai segnali EMG.

- **Media mobile:** è uno strumento molto utilizzato in statistica ed economia, e consiste nel calcolare la media aritmetica all'interno di una finestra che scorre lungo il segnale, in modo analogo al filtro RMS. Si può calcolare nel seguente modo:

$$y_i = \frac{1}{w} \sum_{k=i-\frac{w}{2}}^{i+\frac{w}{2}} x_k \tag{3.9}$$

  Dal punto di vista della teoria dei segnali, la media mobile non è altro che una *convoluzione discreta* del segnale originale con un vettore di dimensione w e coefficienti tutti pari a $1/w$. In particolare, rappresenta uno dei casi più semplici di filtri passa basso di tipo FIR (*Finite Impulse Response*). La risposta in frequenza della media mobile è tutt'altro che ideale, ma ha la qualità di ridurre il rumore del segnale senza alterarne eccessivamente la forma. Oltre a ciò, se la finestra ha dimensione dispari ed è perfettamente centrata sul campione in oggetto, il filtro ha la proprietà di essere a *fase nulla*, cioè è esente dal problema della distorsione di fase tipica dei filtri IIR.

- **Filtro di Savitzky-Golay:** si tratta di un filtro digitale che ha lo scopo di pulire un segnale ed estrarne la morfologia. Concettualmente, il suo funzionamento è il seguente: si fa scorrere una finestra di lunghezza w lungo il segnale, centrata sul campione in oggetto; si calcola un polinomio interpolatore (di ordine a piacere, ma generalmente basso) sui w campioni all'interno della finestra, tramite il *metodo dei minimi quadrati*; se ne calcola il valore al centro della finestra, che diventerà il valore del nuovo segnale filtrato. Il filtro di Savitzky-Golay può essere implementato efficientemente come un filtro FIR, i cui coefficienti di convoluzione possono essere precalcolati analiticamente, fissato w e l'ordine del polinomio k (che dev'essere dispari). La media mobile può essere vista come un caso speciale del filtro di Savitzky-Golay, con $k = 1$, ed in generale, il secondo è meno adatto a rimuovere il rumore del primo, ma permette di seguire in modo più fedele la morfologia del segnale, preservandone i picchi.

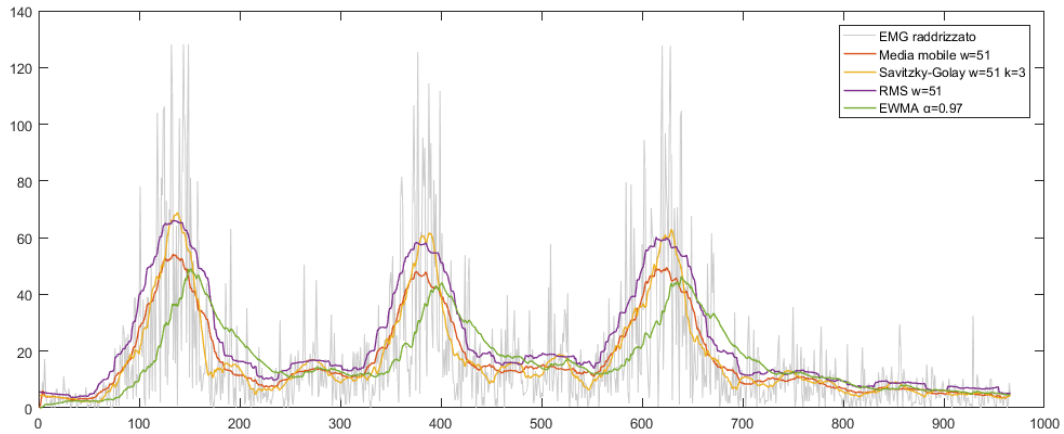


Figura 3.9: Confronto tra le varie tecniche appena descritte su un segnale EMG. Dove applicabile, tutte le finestre w sono state poste pari a 51 punti (255 ms). Come previsto, il filtro EWMA tende a causare un ritardo nel segnale, mentre il filtro Savitzky-Golay è quello che riproduce più fedelmente i picchi, al costo di un maggiore rumore. La media mobile ed il filtro RMS producono una forma praticamente identica, con una lieve differenza di scala.

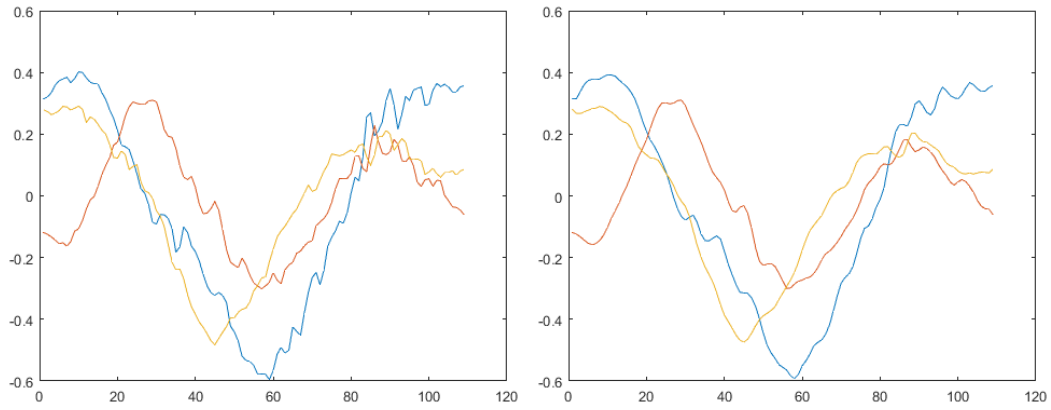
Data la sua semplicità di implementazione, ed un'approssimazione più che soddisfacente dell'inviluppo del segnale, per l'estrazione delle features si è deciso di utilizzare l'approccio basato sul raddrizzamento del segnale, unito ad un filtro basato su *media mobile*. Empiricamente, si è determinato che la dimensione della finestra che dà i migliori risultati è $w = 31$, corrispondente ad una durata di 155 ms (considerando la frequenza di campionamento di 200 Hz).

Inoltre, dato che a questo punto il segnale non contiene componenti ad alta frequenza, è stato effettuato un *downsampling* di un fattore 5x su di esso, riducendo la frequenza di campionamento da 200 Hz a 40 Hz. Ciò ha lo scopo di ridurre il volume dei dati (e quindi il tempo di calcolo per la classificazione) senza alterare le capacità di riconoscimento.

Ovviamente, le precedenti tecniche possono essere applicate solamente su un segnale dotato di una singola variabile. Dato che per ogni traccia di gesture sono presenti 8 segnali EMG, è necessario applicare il suddetto filtro su ognuno di essi (in modo indipendente).

3.6.2 Segnali inerziali

Come già visto, i segnali provenienti dall'IMU sono già puliti, e non richiedono un procedimento di estrazione delle feature. Se lo si ritiene necessario, è possibile applicare un filtro di Savitzky-Golay per esaltarne la curvatura, ma nella pratica i suddetti segnali possono già essere utilizzati senza filtraggio.



(a) Originale (b) Filtrato

Figura 3.10: Filtraggio del segnale proveniente dall'accelerometro, utilizzando il filtro di Savitzky-Golay, con grandezza della finestra $w = 11$ (220 ms) e ordine del polinomio interpolante $k = 3$.

In tale senso, sono stati effettuati i seguenti esperimenti:

- Denoising del segnale con un filtro passa basso.
- Rimozione dell'offset gravitazionale con la tecnica spiegata nell'eq. (3.3).

Entrambi i test hanno avuto effetti trascurabili durante le sperimentazioni preliminari sul sistema di riconoscimento, pertanto non sono stati utilizzati nel sistema finale. Infatti, dato che in un contesto real-time tutti i filtri causano inevitabilmente un ritardo nel segnale, è buona pratica applicarli nella misura minima necessaria.

3.7 Riconoscimento di gesture isolate

Prima di trattare l'argomento del riconoscimento in tempo reale delle gesture, è necessario sviluppare una tecnica che permetta di classificare correttamente i segnali già segmentati (*gesture isolate*), cioè quelli del *validation set* e del *test set*. Una volta trovato un modello che permetta di classificare correttamente le gesture isolate, sarà possibile utilizzarlo come pilastro per costruire il sistema di riconoscimento real-time (si veda la sezione 3.4.3).

3.7.1 Classificazione tramite Dynamic Time Warping

In questa sezione verrà trattato il tema della classificazione dei segnali EMG ed inerziali tramite l'algoritmo Dynamic Time Warping, per riconoscere la gesture corrispondente ad un certo segnale. Si presuppone di avere a disposizione

un *training set* composto da segnali presegmentati ed etichettati con le gesture corrispondenti.

Costruzione del modello

Per l'uso di Dynamic Time Warping non è necessaria alcuna fase di learning, quindi è possibile utilizzare direttamente le sequenze del *training set* come template per la classificazione.

Nel caso dei segnali inerziali, tuttavia, è stata eseguita una normalizzazione dei dati, conosciuta come *standardizzazione* (z-score). Essa consiste nell'applicare una trasformazione lineare sulle sequenze, in modo da ottenere una nuova sequenza con media $\mu = 0$ e deviazione standard $\sigma = 1$. Il processo deve essere eseguito sia sui segnali dell'accelerometro che su quelli del giroscopio, ed in modo indipendente per ogni asse x, y e z. Il procedimento è molto semplice: per ognuna delle 6 variabili X (3 dell'accelerometro e 3 del giroscopio) si calcola la media μ e la deviazione standard σ, e si ricava una nuova variabile Z attraverso la seguente trasformazione:

$$Z = \frac{X - \mu}{\sigma} \tag{3.10}$$

Ciò è stato fatto per due motivi:

1. Dato che i segnali dei due sensori utilizzano unità di misura diverse (g e rad/s), nell'eventuale costruzione di un feature vector composto dalla loro coppia, i valori di un sensore predominerebbero l'altro. Infatti, si è visto che i valori dei segnali del giroscopio sono almeno un ordine di grandezza superiori a quelli dell'accelerometro, quindi questi ultimi diventerebbero trascurabili durante il calcolo della distanza tra i feature vectors. Normalizzando i dati, ci si assicura che tutti i sensori abbiano le stesso peso.

2. Dynamic Time Warping è molto sensibile alle variazioni di ampiezza, anche se i segnali sono allineati correttamente. La normalizzazione rende l'algoritmo invariante all'ampiezza dei segnali da confrontare.

Successivamente, si costruisce la sequenza di *feature vectors* concatenando i dati dei due sensori. Ogni feature vector, quindi, dovrà essere composto dai seguenti valori:

$$\mathbf{x_i} = (a_{ix}, a_{iy}, a_{iz}, r_{ix}, r_{iy}, r_{iz}) \tag{3.11}$$

dove $\mathbf{a_i}$ è il vettore dell'accelerazione, e $\mathbf{r_i}$ è il vettore del giroscopio, entrambi opportunamente normalizzati.

Il suddetto procedimento viene applicato per ogni campione appartenente al training set, ed ovviamente dovrà essere applicato anche sui nuovi segnali da classificare.

Per quel che riguarda i segnali EMG, da varie prove effettuate è emerso che normalizzare i segnali è deleterio, in quanto l'informazione sull'ampiezza è una caratteristica molto importante per discriminare le varie gesture. Di conseguenza, i feature vector di questi segnali sono stati composti direttamente dai valori originali (opportunamente filtrati secondo le tecniche precedentemente descritte):

$$\mathbf{y_i} = (e_{i1}, e_{i2}, e_{i3}, e_{i4}, e_{i5}, e_{i6}, e_{i7}, e_{i8}) \tag{3.12}$$

Classificazione

Per eseguire la classificazione di una nuova sequenza, si è deciso di adottare l'algoritmo *k-Nearest Neighbors* (k-NN), già descritto nella sezione 2.1.2. Empiricamente, si è determinato che i migliori risultati si ottengono con $k = 3$ (per il riconoscimento di gesture isolate).

L'uso di k-NN permette anche di ottenere una stima approssimativa della probabilità che la sequenza appartenga alla classe con la frequenza maggiore: infatti, chiamando f_{max} la frequenza più alta ($1 \leq f_{max} \leq k$), si ha che $P = f_{max}/k$.

Per quel che riguarda la funzione distanza da associare a DTW, è stato deciso di usare la *distanza di Manhattan* sia per i segnali mioelettrici che per quelli inerziali (che, si ricorda nuovamente, sono gestiti separatamente fra loro). Nel caso dei segnali inerziali, infatti, le informazioni geometriche legate ai sensori sono andate perdute durante le fasi di fusione dei feature vectors e normalizzazione, rendendo inadatto l'uso della distanza euclidea.

3.7.2 Classificazione tramite Hidden Markov Models

Dato che nel contesto di questo progetto si ha a che fare con segnali continui, si è scelto di utilizzare gli Hidden Markov Models a distribuzione di probabilità continua (cHMM). Come spiegato nella sezione 2.2.2, è stato creato un HMM per ogni gesture da riconoscere, utilizzando una distribuzione normale multivariata per le osservazioni. Il numero di variabili di quest'ultima, è ovviamente pari alla dimensione dei feature vectors (8 per i segnali EMG, e 6 per i segnali inerziali).

L'allenamento degli HMM è stato effettuato in un contesto *unsupervised*, cioè utilizzando l'algoritmo di Baum-Welch con le sequenze aventi le etichette delle classi (gesture), ma non quelle degli stati. Dato che l'algoritmo di Baum-Welch richiede una stima iniziale dei parametri del modello (specialmente nel

caso delle HMM continue), sono state adottate due strategie diverse tra le gesture statiche (che usano i segnali EMG) e quelle dinamiche (che si basano invece sui segnali inerziali).

Segnali EMG

In base a varie prove effettuate, si è visto che inizializzando i parametri del modello in modo uniforme o casuale, l'algoritmo di Baum-Welch converge ad una soluzione che dà pessimi risultati. Per questo motivo, è stata sviluppata una procedura euristica di inizializzazione del modello, che permette di trovare dei parametri iniziali non eccessivamente lontani da quelli ottimali, per poi rifinirli con l'algoritmo di Baum-Welch. Detta procedura, che deve essere ripetuta per ogni Hidden Markov Model (e quindi per ogni gesture), si può riassumere nei seguenti passaggi:

1. Tra le sequenze del *training set* corrispondenti alla gesture in esame, se ne seleziona una casualmente, che diventerà il template per l'inizializzazione del modello.

2. Si effettua il *clustering* dell'insieme dei feature vectors della sequenza, utilizzando l'algoritmo *k-means clustering*. Concettualmente, ciò ha lo scopo di suddividere il segnale in più soglie di attivazione, basate sull'ampiezza del segnale. Una suddivisione semplificata (ad esempio prendendo il valore minimo e massimo del segnale, e suddividendolo in k sottointervalli) non funzionerebbe, perché il segnale è multivariato (composto dalle letture degli 8 sensori). In sintesi, come euristica, si lascia che sia l'algoritmo k-means a trovare delle soglie adeguate. Sempre come euristica, si è scelto $k = 3$, corrispondente a 3 soglie di attivazione (muscolo a riposo, in parziale contrazione, in totale contrazione). Questo passaggio si può considerare come una forma di *Vector Quantization* (VQ), sebbene il suo scopo sia differente. Si noti anche che i cluster solitamente coincidono con soglie di attivazione diverse (prendendo in considerazione l'ampiezza media tra i vari sensori), ma possono anche rappresentare attivazioni con ampiezza uguale di muscoli differenti.

3. Si assegna ogni feature vector della sequenza al rispettivo *cluster* ottenuto nel passaggio precedente. In questo modo, si ottiene un'effettiva segmentazione in stati (contratto/non contratto).

4. Partendo dall'inizio della sequenza, si avanza finché non si rileva una variazione del cluster associato all'indice corrente. A questo punto, viene aggiunto uno stato nell'HMM, e se ne stima la distribuzione di probabilità (media e matrice di covarianza) utilizzando i dati contenuti nel

segmento di segnale in esame. Partendo dal punto corrente, si ripete il procedimento finché non si arriva alla fine della sequenza. In parole semplici, si crea uno stato per ogni sottosequenza contigua di cluster (cioè per ogni stato di contrazione del muscolo).

5. Gli stati vengono collegati fra loro con una topologia *left-to-right*, e le probabilità di transizione vengono definite come segue: dati N stati (corrispondenti a N sottosequenze di lunghezza n_i, $i \in [1..N]$), si deve ottenere

$$a_{i,j} = \begin{cases} 0 & \text{se } j < i \\ 1 - \frac{1}{n_i} & \text{se } j = i \\ \frac{1-a_{i,i}}{N-i} & \text{se } j \geq i \end{cases} \tag{3.13}$$

   Ovviamente, per l'ultimo stato vale che $a_{N,N} = 1$. Inoltre, come avviene solitamente nei modelli left-to-right, si pone $\pi_1 = 1$ e $\pi_i = 0 \ \ \forall\, i > 1$.

6. A questo punto il modello iniziale è pronto, e si può applicare l'algoritmo di Baum-Welch utilizzando le sequenze del *training set*.

Dato che il suddetto procedimento potrebbe non essere di facile comprensione, di seguito se ne riporta un'illustrazione grafica.

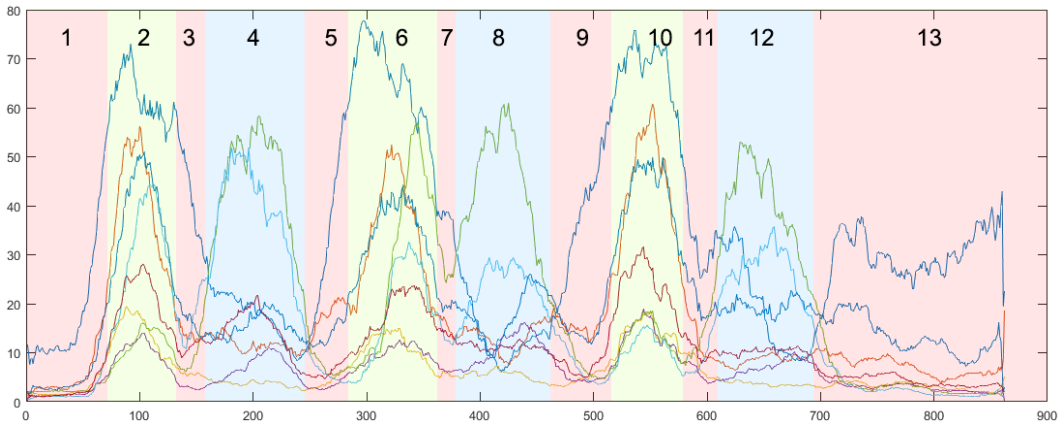


Figura 3.11: Segmentazione di una gesture in stati. I 3 cluster prodotti da *k-means* corrispondono ai colori di sfondo (rosso, verde e blu). La gesture consisteva nel movimento della mano a destra e poi a sinistra, ripetuto per 3 volte. Come si può osservare, l'algoritmo proposto ha prodotto una segmentazione ottimale del segnale, riuscendo anche ad estrapolare le caratteristiche del movimento eseguito: infatti, gli stati rossi corrispondono alla posizione di riposo/ritorno al centro, gli stati verdi corrispondono al movimento a destra, mentre quelli blu al movimento a sinistra. In questo caso, l'algoritmo avrebbe prodotto un Hidden Markov Model da 13 stati, calcolando per ognuno di essi una stima iniziale del vettore media e matrice di covarianza.

Come ulteriore delucidazione, lo stato 1 della figura ha una lunghezza di 71 campioni. Ciò significa che la probabilità di transizione $a_{1,1}$ sarà pari a $1 - 1/71 \approx 0.9859$, mentre $a_{1,2}..a_{1,13} = 1/(71 \cdot 12) \approx 0.0012$, in modo che la somma delle probabilità della riga valga 1.

Segnali inerziali

Questo caso si è rivelato più semplice del precedente. Infatti, si è constatato che un'inizializzazione uniforme dell'Hidden Markov Model è più che sufficiente per permettere all'algoritmo di Baum-Welch di convergere ad una soluzione soddisfacente.

L'unica stima necessaria ha riguardato il numero di stati dei modelli. Empiricamente, si è deciso di assegnare 5 stati per ogni modello (quindi per ogni gesture), in quanto questo valore ha fornito i migliori tassi di riconoscimento. Anche in questo caso, la topologia è di tipo *left-to-right*; tuttavia, sono stati ammesse solamente le transizioni con profondità $d = 1$, cioè, da uno stato a quello immediatamente successivo. I parametri del modello sono stati

inizializzati in modo uniforme, come segue:

$$A = \begin{bmatrix} 0.5 & 0.5 & 0 & 0 & 0 \\ 0 & 0.5 & 0.5 & 0 & 0 \\ 0 & 0 & 0.5 & 0.5 & 0 \\ 0 & 0 & 0 & 0.5 & 0.5 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.14}$$

$$\pi = (1, 0, 0, 0, 0) \tag{3.15}$$

$$\boldsymbol{\mu} = \mathbf{0} \tag{3.16}$$

$$\boldsymbol{\Sigma} = I \tag{3.17}$$

Si noti anche che, per tutti gli stati, la distribuzione di probabilità viene inizializzata in modo da avere media nulla e matrice di covarianza pari all'identità.

Note finali

A differenza di come è stato eseguito nell'approccio basato su Dynamic Time Warping, qui non viene applicata alcuna forma di normalizzazione del segnale, perché l'algoritmo di Baum-Welch è già in grado di stimare la distribuzione di probabilità associata ai campioni dei segnali. In altre parole, una normalizzazione del segnale basata su tecniche di tipo statistico (come *z-score*) sarebbe del tutto ridondante.

La classificazione viene effettuata con la metodologia descritta nella sezione 2.2.2.

3.8 Riconoscimento real-time di gesture

Precedentemente è stato trattato il caso del riconoscimento *isolato* di gesture, cioè partendo da segnali già segmentati. In vari contesti, le tecniche considerate sarebbero sufficienti per eseguire un riconoscimento completo: si pensi, ad esempio, alle *touch gestures* effettuate sullo smartphone, dove la pressione ed il sollevamento del dito sullo schermo effettuano una segmentazione naturale del movimento. In questo contesto, tuttavia, ciò che si vuole ottenere è il riconoscimento *real-time* di gesture a partire da un flusso continuo di dati (i segnali provenienti dai sensori). Il problema può essere fondamentalmente scomposto in due sottoproblemi:

1. **Costruzione di un modello di classificazione "non-gesture":** le tecniche per la classificazione descritte precedentemente trovano la classe più probabile (gesture) associata ad una certa sequenza temporale. Tuttavia, nel caso si decidesse di classificare una sequenza che non rappresenta una gesture, il sistema la classificherebbe comunque come la

gesture più somigliante, producendo un falso positivo. Per questo motivo, è necessario sviluppare un modello che permetta di rilevare se una gesture è valida o meno, classificandola come *non-gesture* nel secondo caso.

2. **Costruzione di un modello di segmentazione:** dato che i segnali provengono da uno *stream* continuo, non si è in grado di conoscere a priori il punto iniziale e finale di una gesture. Il classificatore, d'altronde, ha bisogno di ricevere in input un segnale segmentato per effettuarne il riconoscimento. Di conseguenza, è necessario sviluppare un modello che sia in grado di riconoscere le gesture all'interno di uno stream di dati (*gesture spotting*), o in alternativa, fornire una segmentazione approssimativa che fornisca i risultati sperati.

3.8.1 Modello non-gesture

Per la costruzione di un modello che permetta di scartare le sequenze che non rappresentano gesture, sono stati individuati i seguenti approcci:

Classe dedicata

Una tecnica comunemente utilizzata in questi casi consiste nel creare una nuova classe (*filler/garbage model*), come se fosse un'altra gesture, che verrà allenata con un grande insieme di campioni che non rappresentano gesture. Ciò significa che sarà necessario raccogliere dei dati aggiuntivi da includere nel *training set*. I dati raccolti, dovrebbero essere il più possibile rappresentativi di movimenti casuali e non voluti.

Come si può intuire, questa tecnica presenta alcune problematiche: i dati raccolti dovrebbero coprire l'intero insieme di movimenti *non-gesture*, e ciò è chiaramente impossibile. Inoltre, i movimenti dovrebbero essere eseguiti con attenzione, avendo cura di non sovrapporli a quelli delle gesture effettive.

Modello basato su soglie

I modelli di classificazione trattati in questo capitolo (DTW e HMM), oltre alla classe più probabile a cui appartiene una certa sequenza, riportano una misura di similarità rispetto al modello. Nel caso di DTW, si ottiene una misura di distanza tra la sequenza ed il template, mentre nel caso degli HMM si ottiene la probabilità che la sequenza sia stata generata dal modello (*likelihood*). Attraverso opportune analisi statistiche, è possibile fissare delle soglie (*threshold*) nei suddetti parametri, sopra le quali il riconoscimento viene marcato come fallito.

Nel caso degli Hidden Markov Models, è necessario precisare un aspetto importante: una semplice soglia sulla likelihood non è sufficiente, perché questo valore diminuisce esponenzialmente con l'aumentare della lunghezza della sequenza. Di conseguenza, per confrontare sequenze di lunghezze diverse, un approccio valido consiste nel normalizzare il *log-likelihood* (il logaritmo della likelihood) con la lunghezza della sequenza, cioè:

$$\log \hat{P}(\mathbf{O}|\lambda) = \frac{\log P(\mathbf{O}|\lambda)}{n} \tag{3.18}$$

dove n è la lunghezza della sequenza $\mathbf{O}$.

In maniera analoga, talvolta viene introdotto un fattore di normalizzazione anche nella distanza ottenuta da Dynamic Time Warping, in modo da confrontare i risultati di sequenze con lunghezza differente. Uno degli approcci più utilizzati consiste nel dividere la distanza per la lunghezza del warping path $\mathbf{w}$. Tuttavia, la reale utilità della normalizzazione applicata a DTW è oggetto di pareri contrastanti [29].

Modello specifico

Le due idee descritte sopra sono generiche ed applicabili a qualsiasi sistema di riconoscimento. D'altra parte, in certi casi è possibile individuare alcuni approcci *custom* che sfruttano determinate proprietà del modello in uso.

Lee et al. [25] hanno proposto un metodo basato su Hidden Markov Models per distinguere le gesture significative da quelle sconosciute, tramite la costruzione di un apposito modello adattativo (*threshold model*) che fornisce un valore di soglia della likelihood in grado di evolversi nel tempo. Questo approccio non richiede alcuna informazione aggiuntiva rispetto a quelle che si hanno già.

Il metodo consiste nella costruzione di un HMM con topologia *ergodic* (in cui ogni stato è raggiungibile da un altro), i cui stati sono l'insieme di *tutti* gli stati dei modelli delle gesture. In altri termini, si copiano gli stati degli HMM corrispondenti alle gesture, mantenendo le rispettive distribuzioni di probabilità per le emissioni dei simboli. Successivamente, gli stati vengono connessi fra loro creando un grafo totalmente connesso. Le probabilità di transizione vengono definite in modo da mantenere intatte le probabilità di auto-transizione (cioè da uno stato a sè stesso). Le probabilità di transizione da uno stato a tutti gli altri, invece, vengono distribuite uniformemente in modo che la somma delle probabilità in uscita equivalga ad 1.

Questo modello viene successivamente aggiunto tra le classi del classificatore (come *non-gesture*), e rappresenta una soglia di likelihood adattativa, in grado di aumentare o diminuire nel tempo (grazie all'uso della topologia

ergodic). Una gesture, quindi, verrà riconosciuta solo se la sua likelihood è superiore a quella del *threshold model* appena descritto.

Questo modello funziona perché l'insieme di tutti gli stati crea effettivamente una sorgente di *entropia*, permettendo di rappresentare qualsiasi movimento. Il fatto che siano mantenute solamente le probabilità di auto-transizione, e non quelle in uscita, fa sì che durante l'esecuzione di una gesture reale, il modello corrispondente a tale gesture domini il *threshold model* generando una probabilità più alta. Al contrario, nel caso dell'esecuzione di una gesture non riconosciuta, il *threshold model* produrrà la likelihood più elevata, dominando tutti gli altri modelli.

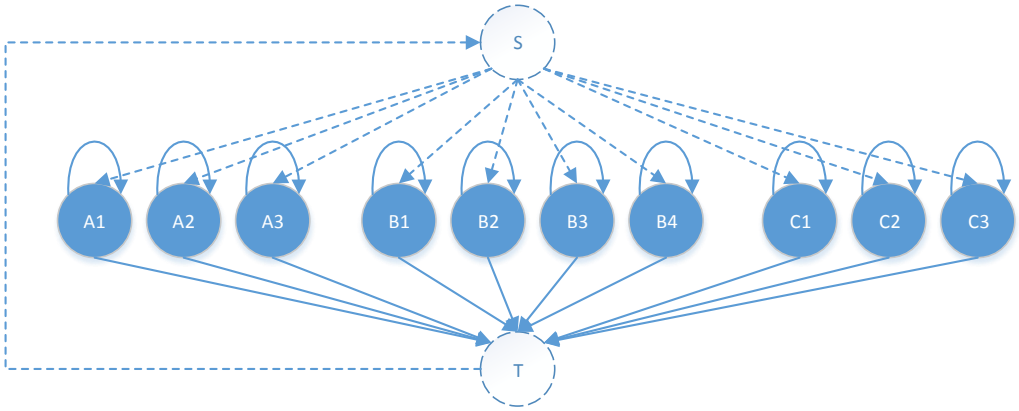


Figura 3.12: Threshold model per un classificatore composto da 3 gesture, che consistono rispettivamente di 3, 4 e 3 stati. Gli stati S e T sono *null states*, dei particolari tipi di stato che non producono osservazioni.

3.8.2 Modello di segmentazione

L'approccio più semplice consiste nell'utilizzare una finestra temporale scorrevole (*sliding window*) di larghezza w (corrispondente cioè a w campioni), che ad ogni iterazione scorre avanti di $p < w$ campioni, in modo da ottenere una sovrapposizione parziale tra le finestre. Ad ogni istante di tempo, il contenuto della finestra verrà passato in ingresso al classificatore, che restituirà un responso. In questo modo, si otterrà una sequenza contigua di classi in corrispondenza di una gesture, e l'evento di gesture riconosciuta potrà essere lanciato ad ogni cambiamento di contiguità. Ovviamente, per funzionare correttamente, questo sistema dovrà essere integrato con un valido modello *non-gesture* tra quelli descritti precedentemente.

Questa tecnica può essere implementata attraverso una coda di lunghezza w, in cui ad ogni istante di tempo viene accodato un nuovo campione, e viene

rimosso quello più vecchio. Se il tempo corrente è un multiplo del passo p, viene effettuata la classificazione del contenuto corrente della finestra.

L'approccio appena descritto è generico ed applicabile sia nel caso del Dynamic Time Warping che nel caso degli Hidden Markov Models. Tuttavia, per questi ultimi, sono state sviluppate delle tecniche speciali.

Lee et al. [25], nello stesso articolo in cui introducono il *threshold model*, hanno descritto un metodo per effettuare il riconoscimento di gesture all'interno uno stream di dati, ottenendo così una segmentazione automatica. Esso consiste nella costruzione di una particolare topologia di Hidden Markov Model, chiamata *Gesture Spotting Network* (GSN), che contiene tutti i modelli *left-to-right* delle gesture, ed il *threshold model*, collegati fra loro in modo ciclico: partendo da uno stato iniziale, ci si può immettere in qualsiasi modello tra quelli presenti, per poi ritornare allo stato iniziale.

Durante la fase di riconoscimento, si confrontano tutte le likelihood prodotte dai vari modelli della rete, e non appena la likelihood di uno dei modelli relativi alle gesture scende sotto quella del *threshold model*, significa che una gesture è stata completata. A questo punto, viene applicato l'algoritmo di Viterbi per risalire alla sequenza più probabile di stati, e si effettua un *backtracking* fino ad arrivare allo stato iniziale della gesture. In questo modo, sono stati trovati i due punti estremi della gesture nel segnale, ottenendo effettivamente una sequenza segmentata da poter classificare.

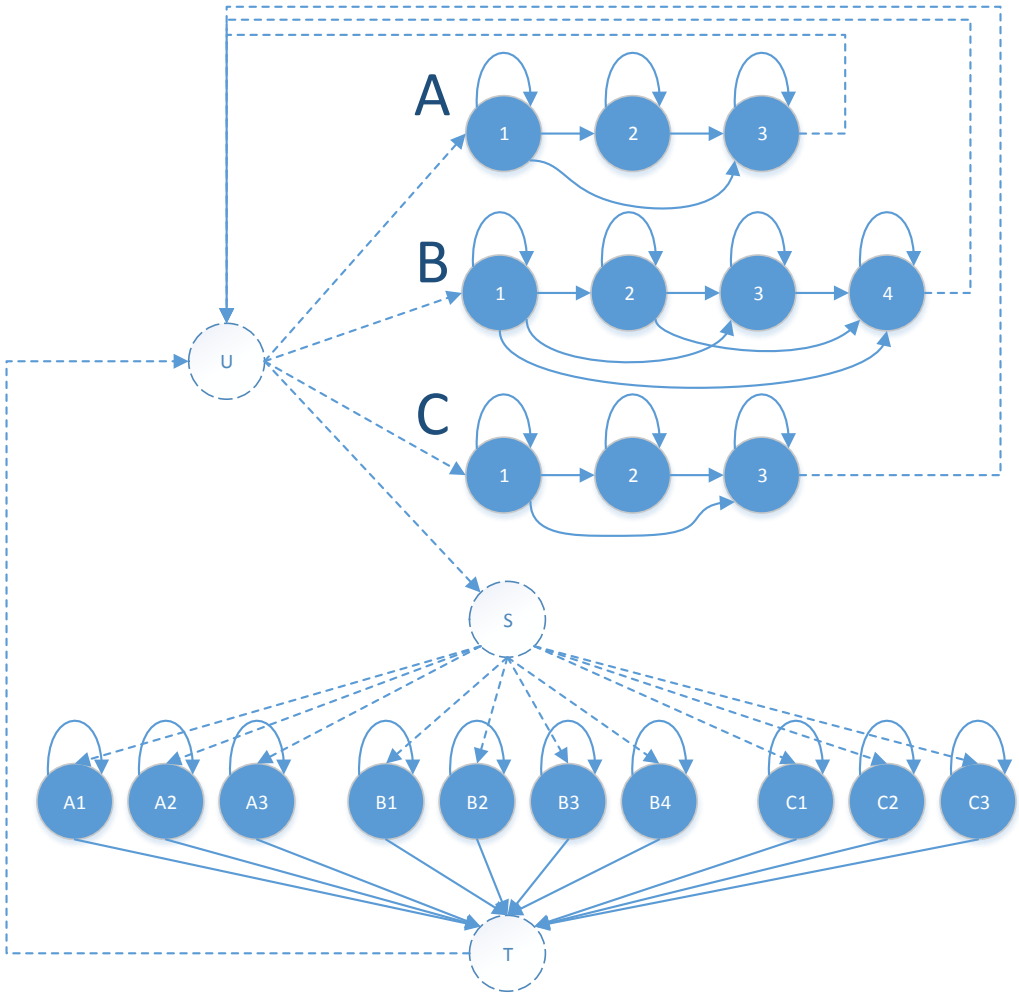


Figura 3.13: Gesture Spotting Network.

L'approccio sopracitato ha lo svantaggio di produrre una certa latenza nel riconoscimento delle gesture, dato che prima di effettuare la classificazione è necessario attendere che il *threshold model* ne decreti il completamento.

Alternativamente, è stata sviluppata una tecnica esente dal suddetto problema [26], che consiste nell'utilizzare una finestra temporale molto ristretta (pochi campioni) su cui eseguire il confronto della likelihood tra i modelli delle gesture ed il *threshold model*. Una volta che la likelihood di uno dei primi supera quella del secondo, la gesture può considerarsi iniziata, ed è già possibile avviare la classificazione in modo sequenziale (attraverso un'implementazione incrementale dell'algoritmo di Viterbi). Quando il threshold model torna nuovamente a dominare le gesture all'interno della finestra, il movimento può ritenersi completato, e si ha già il risultato della classificazione.

Questo approccio è più reattivo, perché il riconoscimento avviene in avanti, e non indietro (backtracking), ed il ritardo massimo è stabilito dalla dimensione della finestra.

3.8.3 Metodo proposto

Tutte le tecniche sopracitate sono state valutate per lo svolgimento di questo progetto. Per quel che riguarda gli approcci basati su Hidden Markov Models, si è constatato che l'implementazione di una Gesture Spotting Network richiede una particolare implementazione degli HMM e dell'algoritmo di Viterbi, in grado di gestire i cosiddetti *null states*, che consistono in stati che non emettono simboli. L'uso di queste funzionalità avrebbe aumentato enormemente la complessità del progetto, senza tuttavia avere la garanzia del suo corretto funzionamento.

Sono stati effettuati anche diversi esperimenti con il *threshold model* proposto, ottenendo purtroppo un successo molto limitato. Probabilmente, i segnali coinvolti in questo progetto hanno una variabilità intrinseca troppo elevata, venendo così classificati come *non-gesture*.

L'approccio che si è deciso di adottare, invece, è composto da un classificatore basato su Dynamic Time Warping e k-Nearest Neighbors, un modello *non-gesture* basato su soglie, ed un modello di segmentazione basato su finestre temporali scorrevoli.

Prima di procedere, è necessario fare una premessa: dato che il modello di riconoscimento è basato su soglie, saranno sicuramente presenti dei parametri che ne regoleranno la sensibilità. Sarà quindi necessario trovare un compromesso tra *falsi negativi* e *falsi positivi*: i primi avvengono quando si esegue una gesture corretta, ma questa non viene riconosciuta, mentre i secondi avvengono quando il sistema segnala l'esecuzione di una gesture, sebbene in realtà non sia mai stata eseguita. In un contesto realistico, i falsi positivi rappresentano un errore più grave dei falsi positivi. Infatti, se una gesture non viene riconosciuta, l'utente la può svolgere nuovamente senza conseguenze negative, mentre se il sistema rileva una gesture che non è stata eseguita, ciò potrebbe innescare un evento/input non voluto. In sintesi, il sistema di riconoscimento verrà regolato in modo da essere conservativo piuttosto che aggressivo.

Modello non-gesture proposto

Di seguito si propone un approccio che permette di scartare le gesture non valide, basandosi unicamente sulla distanza DTW e sulle frequenze del classificatore k-Nearest Neighbors.

Il metodo consiste in una fase di *learning* iniziale, che ha lo scopo di calcolare una soglia sulla distanza DTW per ogni template di gesture. Per ogni classe (cioè per ogni gesture) appartenente al *training set*, si esegue il seguente procedimento:

1. Dati n template per una certa gesture, si calcola una matrice quadrata D dei costi fra tutte le n^2 coppie:

$$D = \begin{bmatrix} d_{1,1} & d_{1,2} & \dots & d_{1,n} \\ d_{2,1} & d_{2,2} & \dots & d_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{n,1} & d_{n,2} & \dots & d_{n,n} \end{bmatrix} = \begin{bmatrix} 0 & d_{1,2} & \dots & d_{1,n} \\ d_{1,2} & 0 & \dots & d_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ d_{1,n} & d_{2,n} & \dots & 0 \end{bmatrix} \tag{3.19}$$

   dove $d_{i,j}$ rappresenta la distanza DTW fra la sequenza i-esima e la sequenza j-esima. Dato che $d_{i,i} = 0$ e $d_{i,j} = d_{j,i}$, è sufficiente riempire la matrice triangolare superiore (o inferiore) e specchiarla, calcolando così solamente $n(n-1)/2$ coppie.

2. Per ogni riga i della matrice D, si calcola la media μ_i corrispondente, escludendo l'elemento $d_{i,i}$ (che vale sempre 0):

$$\mu_i = \frac{1}{n-1} \sum_{j \in [1..n] \setminus \{i\}} d_{i,j} \tag{3.20}$$

3. Per ogni riga i della matrice D, si calcola la deviazione standard σ_i corrispondente, escludendo l'elemento $d_{i,i}$:

$$\sigma_i = \sqrt{\frac{1}{n-1} \sum_{j \in [1..n] \setminus \{i\}} (d_{i,j} - \mu_i)^2} \tag{3.21}$$

4. Si stabilisce la soglia θ_i del template i-esimo come la sua media più un certo numero di deviazioni standard. Se la distanza DTW tra questo template e la sequenza da classificare è superiore a θ_i, la gesture verrà scartata.

$$\theta_i = \mu_i + k\,\sigma_i \tag{3.22}$$

   Empiricamente, si è determinato $k = 2$ (due deviazioni standard) come compromesso tra il tasso di riconoscimento ed i falsi positivi. Inoltre, tramite una rapida analisi, è emerso che la distribuzione delle distanze del *training set* è approssimabile ad una distribuzione normale, ed in linea teorica la scelta di $k = 2$ limiterebbe il tasso di riconoscimento ad un massimo del 97.72% circa, che rappresenta un valore più che accettabile.

Per effettuare la classificazione di una gesture, tenendo conto delle soglie calcolate nella fase di training, si utilizza un classificatore k-Nearest Neighbors modificato come segue:

1. Inizialmente, si mantiene invariato l'algoritmo k-NN: viene calcolata una lista di distanze tra la sequenza da classificare e tutti i template. Non viene applicato alcun modello di soglia.

2. Si selezionano i k template di costo minimo (anche questo passaggio rimane invariato).

3. Per ognuno dei k template trovati, si applica il modello di soglia descritto precedentemente: i template che hanno un costo superiore a θ_i ($1 \leq i \leq k$) vengono scartati.

4. Si calcolano le frequenze f_g delle gesture corrispondenti ad ogni template rimanente (cioè il numero di volte che la gesture g compare fra i template rimanenti, limitati superiormente da k), e si sceglie la gesture associata alla frequenza massima f_{max} come candidata. In caso di pareggio fra più frequenze, la gesture viene scartata.

5. Si calcola la probabilità di classificazione corretta come $P = f_{max}/k$. Se P è superiore ad una certa soglia P_θ, la classificazione avviene con successo, altrimenti la gesture viene scartata.

A differenza del caso del riconoscimento di gesture isolate, per aumentare la robustezza del sistema si è deciso di assegnare $k = 5$. Inoltre, si è posta la probabilità $P_\theta = 4/5 = 0.8$: in questo modo, per poter essere classificata correttamente, una gesture deve produrre un matching valido con almeno 4 template (su un massimo di 5).

Modello di segmentazione proposto

Per effettuare la segmentazione del segnale, si è deciso di utilizzare una semplice finestra temporale scorrevole. La larghezza della finestra w è stata posta pari a 100 campioni sia per le gesture dinamiche (corrispondente a 2 secondi) che per le gesture statiche (corrispondente a 2.5 secondi, dato che il segnale ha subito un *downsampling* di un fattore 5). Il passo p è stato posto uguale a 10 campioni, che rappresenta un buon compromesso tra prestazioni e sovrapposizione. Idealmente, la larghezza della finestra w dovrebbe essere sufficientemente ampia da poter coprire la lunghezza di tutte le possibili gesture, dato che sarà l'algoritmo Dynamic Time Warping ad isolarne gli estremi in maniera ottimale.

Ad ogni istante di tempo, si esegue la normalizzazione della sequenza contenuta nella finestra, e se ne effettua la classificazione. Nel caso delle gesture dinamiche, la normalizzazione consiste nel calcolo dello *z-score*, come già descritto nella sezione 3.7.1. Invece, per quel che riguarda le gesture statiche (basate sui segnali EMG), nel caso della classificazione di gesture isolate non veniva applicata alcuna forma di normalizzazione. Tuttavia, nel contesto del riconoscimento *real-time*, si è rivelata molto efficace una forma di normalizzazione basata sulla lunghezza di Manhattan. Denominando $\mathbf{x} = (\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_w})$ la sequenza da classificare (cioè il contenuto della finestra), si calcola la lunghezza di Manhattan di ogni suo feature vector $\mathbf{x_i}$; tra queste, si seleziona la lunghezza massima, e si dividono tutti i componenti per essa. Riassumendo:

$$l_i = \|\mathbf{x_i}\|_1 = \sum_{k=0}^{m} |x_{i,k}| \tag{3.23}$$

$$\mathbf{x'_i} = \frac{\mathbf{x_i}}{\max(\{l_1, l_2, \ldots, l_w\})} \tag{3.24}$$

dove l_i denota la lunghezza di Manhattan dell'i-esimo feature vector della sequenza.

Come nota sulle prestazioni, sebbene l'approccio basato su finestra possa sembrare dispendioso, è importante notare che sia la fase di learning che quella di riconoscimento si prestano bene ad essere parallelizzate (nella fattispecie, sono *embarrassingly parallel*). In ogni caso, qualora le prestazioni non risultassero accettabili, è possibile agire sul passo della finestra p e sulle tecniche di ottimizzazione di Dynamic Time Warping descritte nella sezione 2.1.3.

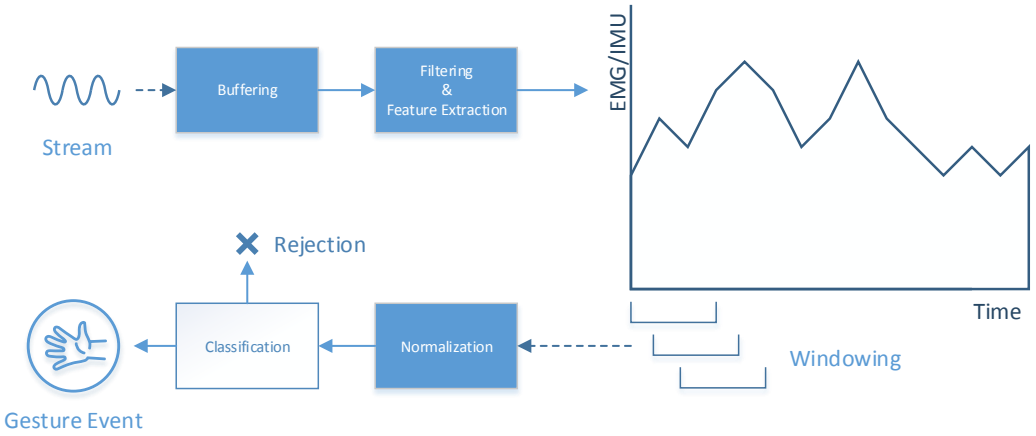


Figura 3.14: Pipeline per il riconoscimento di una gesture.

In fig. 3.14 si mostra uno schema a blocchi complessivo per il sistema di riconoscimento delle gesture. Si noti che, nel caso dei segnali EMG, è presente

una fase di buffering. Essa ha lo scopo di accumulare sufficienti campioni per eseguire il filtraggio con un filtro FIR, che in questo caso corrisponde ad una media mobile di 31 campioni. Come conseguenza, verrà introdotto un ritardo costante nel segnale pari a 15 campioni (la metà della finestra), che corrispondono a 75 ms (un ritardo accettabile). I segnali inerziali, invece, non essendo sottoposti ad alcun filtro, non subiscono ritardi.

In fig. 3.15, invece, viene mostrata la logica con cui vengono prodotti gli eventi in seguito al riconoscimento di una gesture in un determinato istante di tempo.

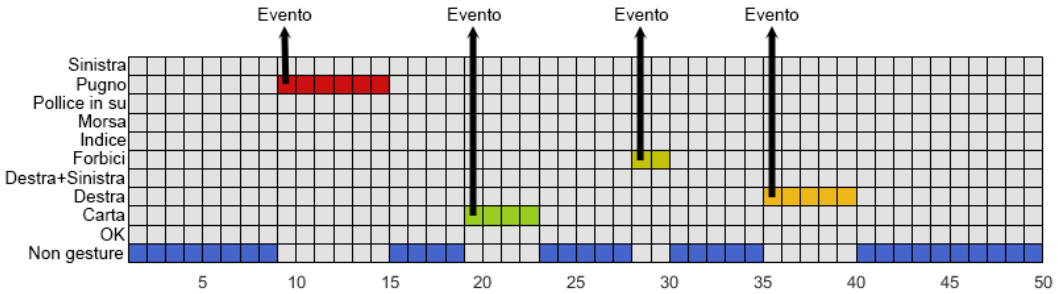


Figura 3.15: Traccia di esecuzione del sistema di riconoscimento. L'asse orizzontale rappresenta la progressione delle finestre temporali. Come si può vedere, ad ogni gesture riconosciuta corrispondono più classificazioni consecutive. L'evento associato ad una gesture viene lanciato in corrispondenza dell'inizio del rispettivo segmento contiguo (per minimizzare la latenza).

Capitolo 4

Risultati sperimentali e conclusioni

4.1 Risultati

4.1.1 Classificazione di gesture isolate

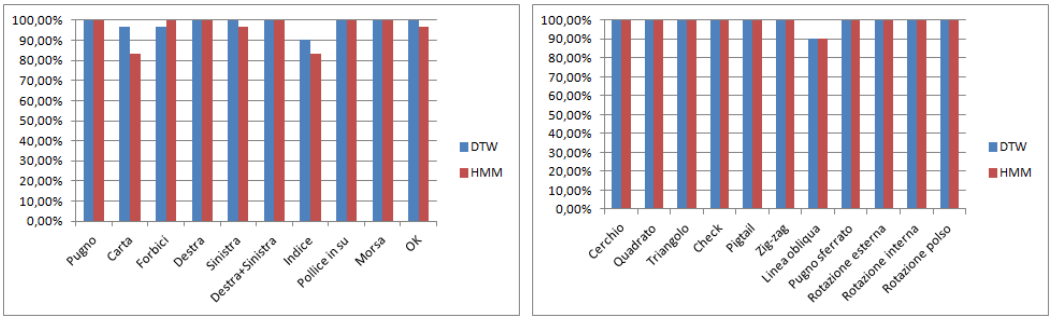
Nella tabella seguente, si mostrano i risultati relativi alla classificazione delle gesture con le tecniche proposte. In questo scenario, è stato coinvolto l'intero *training set*, composto da 60 campioni (esempi) per ognuna delle 21 gesture, per un totale di 1260 campioni. Sia il *validation set* che il *test set*, invece, consistono di 30 campioni per gesture ciascuno (seguendo la suddivisione del dataset in 50-25-25 %, come descritto nella sezione 3.4.3).

		Validation set	**Test set**
HMM	Gesture statiche (EMG)	94.67 %	96 %
	Gesture dinamiche (IMU)	100 %	99.09 %
DTW	Gesture statiche (EMG)	97.67 %	98.33 %
	Gesture dinamiche (IMU)	100 %	99.09 %

Come si può vedere, con l'approccio basato su Dynamic Time Warping si ottengono risultati migliori in tutti i casi, anche se le due tecniche mostrano percentuali molto simili. Nel complesso, i risultati sono ottimi, specialmente per quel che riguarda le gesture dinamiche.

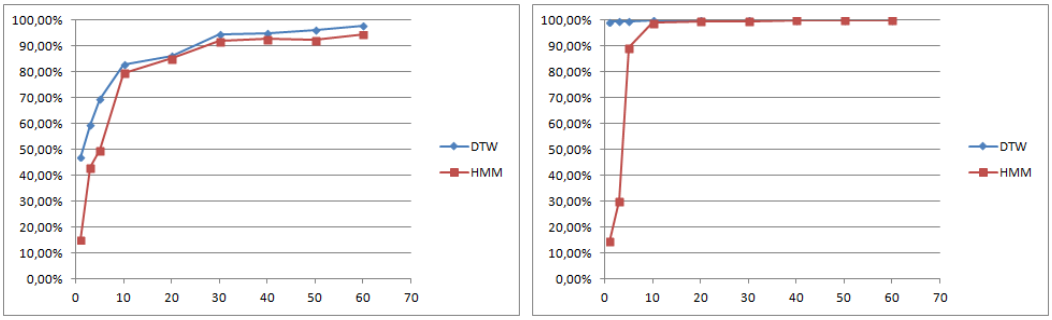
In fig. 4.1 vengono mostrati i tassi di riconoscimento delle singole gesture (sul *test set*), in modo da evidenziare quelle che potrebbero causare ambiguità nella fase di classificazione.

In fig. 4.2, invece, si mostra l'evoluzione del tasso di riconoscimento (sul *validation set*) al variare della dimensione del training set, cioè il numero di campioni per gesture .



(a) Gesture statiche (b) Gesture dinamiche

Figura 4.1: Confronto tra i tassi di riconoscimento delle singole gesture.



(a) Gesture statiche (b) Gesture dinamiche

Figura 4.2: Confronto tra i tassi di riconoscimento dei modelli al variare del numero di campioni del training set per ogni gesture. Sono stati eseguiti test con 1, 3, 5, 10, 20, 30, 40, 50 e 60 campioni per gesture. Si può notare che le gesture statiche (basate sui segnali EMG) sono molto sensibili al numero di esempi utilizzati per il training, mentre le gesture dinamiche necessitano di pochi campioni, specialmente per quel che riguarda il Dynamic Time Warping, che anche qui si rivela superiore in tutti i casi.

4.1.2 Riconoscimento real-time di gesture

Per quel che riguarda il framework per il riconoscimento in tempo reale delle gesture (basato esclusivamente su Dynamic Time Warping), il tasso di riconoscimento è stato valutato sulla base di un dataset composto da varie

registrazioni, all'interno delle quali sono state effettuate alcune serie di gesture in un determinato ordine (sezione 3.4.3), comprendendo anche movimenti casuali (per simulare uno scenario realistico).

I risultati sono stati raggruppati nella seguente tabella, che deve essere interpretata come segue: la linea *Corrette* rappresenta le gesture che sono state riconosciute e classificate correttamente; *Errate* rappresenta quelle che sono state riconosciute al momento giusto, ma che sono state classificate in modo errato (cioè una gesture confusa per un'altra); *Mancate* rappresenta le gesture che non sono state riconosciute. Infine, *Falsi positivi* indica il numero assoluto di gesture che il sistema ha riconosciuto, ma che non sono mai state eseguite.

	Statiche	**Dinamiche**	**Dinamiche [*]**
Corrette	82%	74.55%	91.11%
Errate	6%	0.91%	0%
Mancate	12%	24.55%	8.89%
Totale	100%	100%	100%
Falsi positivi	3	2	2

Per quel che riguarda le gesture dinamiche, è stata notata un'anomalia: le gesture *check* e *linea* hanno esibito un pessimo tasso di riconoscimento. Ciò è probabilmente dovuto alla loro breve durata e/o al fatto che coinvolgono movimenti troppo semplici, che potrebbero potenzialmente costituire parte di una gesture più grande (come *zig-zag*). L'esperimento è stato ripetuto escludendo queste due gesture dal sistema di riconoscimento, ottenendo i risultati esposti nella colonna [*], significativamente migliori dei precedenti.

4.1.3 Prestazioni

Il sistema è stato sviluppato in linguaggio C#, eccetto la procedura per il Dynamic Time Warping, che è stata scritta in C++, dato che rappresenta la parte più critica del framework dal punto di vista prestazionale. I *benchmark* sono stati effettuati con un processore Intel Core 2 Quad Q6600 (dotato di 4 core), sullo stesso dataset del riconoscimento real-time.

Mediamente, la classificazione di una finestra temporale richiede circa 17 ms di tempo (58 finestre/s) per i segnali EMG, e 22 ms (45 finestre/s) per i segnali inerziali. Per effettuare il riconoscimento di 1 secondo di dati, comprendendo sia i segnali EMG che quelli inerziali, sono necessari 177 ms di tempo.

In conclusione, il framework si è dimostrato in grado di effettuare il riconoscimento in tempo reale con ampio margine. Qualora si volesse ridurre la percentuale di utilizzazione della CPU, è possibile modificare alcuni parametri del sistema, come già spiegato nella sezione 3.8.3.

4.2 Considerazioni e sviluppi futuri

I risultati ottenuti dal framework sviluppato sono molto promettenti. In particolare, il modello riconoscimento di gesture isolate ha mostrato tassi di classificazione eccellenti, mentre il modello di riconoscimento real-time ha prodotto dei risultati più che accettabili, sebbene possa esserci un discreto margine di miglioramento. In particolare, allo stato attuale, la classificazione dei segnali EMG presenta difficoltà non trascurabili, data la loro estrema variabilità anche all'interno della stessa persona. Al contrario, i segnali inerziali sono consistenti e permettono di discriminare le varie gesture con facilità. In ogni caso, sia le tecniche basate su Hidden Markov Models che quelle basate su Dynamic Time Warping si sono rivelate valide. A parità di risultati, tuttavia, la seconda opzione richiede meno esempi per l'apprendimento.

Il sistema di riconoscimento sviluppato in questo progetto potrebbe essere migliorato tramite l'uso della cosiddetta tecnica del *Continuous Dynamic Programming* (CDP), che eliminerebbe la necessità di utilizzare una finestra temporale scorrevole, riducendo drasticamente la complessità computazionale del sistema. Inoltre, in un contesto di utilizzo realistico, è importante che il sistema funzioni indipendentemente da come viene indossato il dispositivo. Di conseguenza, si rende necessario implementare la tecnica citata nella sezione 3.4.2.

Dato che il mercato dei sistemi basati su gesture si sta espandendo in modo significativo, è probabile che negli anni a venire verranno sviluppati nuovi approcci per trattare questo tipo di problema. I possibili candidati potrebbero essere basati su reti neurali LSTM, che ancora non sono state impiegate su larga scala per quel che riguarda il riconoscimento delle gesture. Un altro campo di ricerca molto promettente è rappresentato dagli *Hidden Conditional Random Fields* (HCRF) [30], che possono essere interpretati come una generalizzazione degli Hidden Markov Models.

Bibliografia

[1] Statista, *Facts and statistics on wearable technology*, 2016.
http://www.statista.com/topics/1556/wearable-technology/

[2] Y. Zhao, Y. Cai, *Road to ultra low cost multi-sensor integration.* In *Transducers & Eurosensors XXVII: The 17th International Conference on Solid-State Sensors, Actuators and Microsystems*, 2013.

[3] Markets and Markets, *Gesture recognition & touchless sensing market worth 23.55 billion USD by 2020*, 2015.
http://www.marketsandmarkets.com/PressReleases/touchless-sensing-gesturing.asp

[4] R. E. Kalman, *A New Approach to Linear Filtering and Prediction Problems*, Journal of Basic Engineering, 1960.

[5] Google, *Project Soli*, 2015.
https://atap.google.com/soli/

[6] H. Sakoe, S. Chiba, *Dynamic programming algorithm optimization for spoken word recognition*, IEEE Transactions on Acoustics, Speech and Signal Processing, 26(1) pp. 43– 49, 1978, ISSN: 0096-3518

[7] J. Mei, M. Liu, Y. Wang, H. Gao, *Learning a Mahalanobis Distance-Based Dynamic Time Warping Measure for Multivariate Time Series Classification*, IEEE Transactions on Cybernetics, 2015

[8] Y. Chung, *Classification of Continuous Heart Sound Signals Using the Ergodic Hidden Markov Model.* In *Pattern Recognition and Image Analysis: Third Iberian Conference, IbPRIA 2007, Girona, Spain, June 6-8, 2007, Proceedings, Part I.*

[9] L. R. Rabiner, *A tutorial on hidden Markov models and selected applications in speech recognition.* In *Proceedings of the IEEE, Vol. 77, No. 2*, 1989.

[10] B. H. Juang, L. R. Rabiner, *The segmental K-means algorithm for estimating parameters of hidden Markov models*. In *IEEE Transactions on Acoustics, Speech, and Signal Processing (Volume: 38, Issue: 9)*, 1990.

[11] K. Hornik, *Approximation Capabilities of Multilayer Feedforward Networks*. In *Neural Networks, Volume 4, Issue 2*, 1991.

[12] S. Hochreiter, *Untersuchungen zu dynamischen neuronalen Netzen*, Institut f. Informatik, Technische Univ. Munich, 1991.

[13] V. Nair, G. Hinton, *Rectified Linear Units Improve Restricted Boltzmann Machines*. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, 2010.

[14] Y. LeCun, Y. Bengio, G. Hinton, *Deep learning*. In *Nature 521, 436–444*, 2015.

[15] A. Mosca, G. D. Magoulas, *Adapting Resilient Propagation for Deep Learning*. In *Proceedings of the UK workshop on Computational Intelligence (UKCI)*, 2015.

[16] S. Hochreiter, J. Schmidhuber, *Long short-term memory*. In *Neural Computation 9 (8): 1735–1780*, 1997.

[17] Google Research Blog, *Google voice search: faster and more accurate*, 2015.
https://research.googleblog.com/2015/09/google-voice-search-faster-and-more.html

[18] A. Graves, S. Fernandez, F. Gomez, J. Schmidhuber, *Connectionist Temporal Classification: Labelling Unsegmented Sequence Data with Recurrent Neural Networks*. In *Proceedings of ICML 06, pp. 369–376*, 2006.

[19] G. S. Rash, *Electromyography Fundamentals*, 1999.

[20] K. Shoemake, *Animating Rotation with Quaternion Curves*. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH 1985.

[21] J. Wu, J. Konrad, P. Ishwar, *Dynamic time warping for gesture-based user identification and authentication with Kinect*. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC*, 2013, pp. 2371-2375.

[22] G. A. ten Holt, M. J. T. Reinders, E. A. Hendriks, *Multi-Dimensional Dynamic Time Warping for Gesture Recognition*, Delft University of Technology, 2007.

[23] N. Gillian, R. B. Knapp, S. O'Modhrain, *Recognition Of Multivariate Temporal Musical Gestures Using N-Dimensional Dynamic Time Warping*, Queen's University Belfast, 2011.

[24] J. Yang, Y. Xu, *Hidden Markov Model for Gesture Recognition*, Carnegie Mellon University, 1994.

[25] H. K. Lee, J. H. Kim, *An HMM-Based Threshold Model Approach for Gesture Recognition*. In *IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume: 21, Issue: 10)*, 1999.

[26] M. Elmezain, A. Al-Hamadi, B. Michaelis, *Hand trajectory-based gesture spotting and recognition using HMM*. In *16th IEEE International Conference on Image Processing (ICIP)*, 2009.

[27] W. Rose, *Electromyogram analysis*, 2011.
https://www1.udel.edu/biology/rosewc/kaap686/notes/EMG analysis.pdf

[28] C. Mercer, *Data Smoothing : RC Filtering And Exponential Averaging*, 2003.
http://blog.prosig.com/2003/04/28/data-smoothing-rc-filtering-and-exponential-averaging/

[29] C. A. Ratanamahatana, E. Keogh, *Three Myths about Dynamic Time Warping Data Mining*, 2005.

[30] S. B. Wang, A. Quattoni, L. P. Morency, D. Demirdjian, T. Darrell, *Hidden Conditional Random Fields for Gesture Recognition*. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2006.