UNIVERSITÀ DEGLI STUDI DI FIRENZE
Facoltà di Ingegneria

Corso di Laurea in Ingegneria Elettronica

Un'interfaccia cervello-computer mediante metodi evoluzionistici di trattamento di segnali EEG

di

Luca Citi

Relatori:

Prof. Guido Valli

Prof. Carlo Marchesi

Prof. Riccardo Poli

Anno Accademico 2003-2004

Alla mia famiglia

Indice

Prefazione

Le interfacce uomo-computer (*Human-Computer Interface*, HCI) sono un campo di ricerca che ha visto un notevole sviluppo in tempi recenti. Uno sforzo crescente è stato speso al fine di rendere sempre più "a misura d'uomo" i dispositivi artificiali con cui egli interagisce. Il perfezionamento delle tecniche di riconoscimento vocale, la realtà virtuale, lo studio dell'ergonomia di piccoli e grandi elettrodomestici ne sono una prova.

In tutte queste interfacce lo scambio di informazioni tra la mente del soggetto ed il dispositivo artificiale avviene tramite l'intermediazione dei nostri sistemi sensoriale e motorio e di un adeguato insieme di attuatori e sensori ergonomici di cui il dispositivo deve essere dotato. Affinché la volontà del soggetto raggiunga il dispositivo artificiale, un impulso nervoso generato nel cervello deve percorrere le vie efferenti del sistema nervoso, raggiungere la muscolatura volontaria ed essere tradotto in un comando (la pressione di un pulsante, la pronuncia di un comando vocale e così via) che i sensori del dispositivo artificiale siano in grado riconoscere.

Un'interfaccia cervello-computer (dall'inglese *Brain-Computer Interface*, abbreviato BCI) è un tipo di HCI che stabilisce un canale di comunicazione diretto tra il cervello del soggetto ed un computer. Tale interfaccia prescinde dalle comuni vie efferenti del sistema nervoso centrale e pertanto può costituire l'unico mezzo di comunicazione per persone che abbiano perso completamente il controllo della muscolatura volontaria in seguito a patologie neuromuscolari.

Una di queste malattie, la sclerosi laterale amiotrofica (ALS, *Amyotrophic Lateral Sclerosis*), colpisce i motoneuroni e porta a una perdita del controllo della muscolatura volontaria. Con il progredire della malattia sempre più muscoli vengono interessati mentre le capacità intellettive e sensoriali rimangono intatte. Il malato è condannato ad assistere lucidamente alla perdita

progressiva delle proprie capacità motorie e quindi anche di comunicazione.

Nelle Brain-Computer Interfaces, il soggetto riesce a comunicare variando intenzionalmente determinate caratteristiche della propria attività cerebrale. Tali variazioni possono essere rilevate tramite le comuni tecniche di diagnostica e successivamente interpretate da un computer.

Tra le possibili modalità di diagnostica cerebrale – EEG, MEG, PET, fMRI o elettrodi impiantati sulla corteccia – la più utilizzata è senza dubbio l'*elettroencefalografia* in quanto non invasiva, poco costosa e poco ingombrante. Il fine, ancora lontano, è realizzare un ausilio che possa essere "indossato" da pazienti completamente paralizzati e consenta loro di controllare con una buona affidabilità gli apparati elettrici circostanti (illuminazione, televisore, ...), una sedia a rotelle, un arto bionico.

Diverse sono le metodologie con cui il soggetto può variare determinati parametri del proprio segnale elettroencefalografico. Una di queste sfrutta il fatto che alcune caratteristiche dei *potenziali evento-correlati* (ERP) elicitati da uno stimolo esterno variano a seconda del grado di attenzione che il soggetto presta allo stimolo stesso. I potenziali evento-correlati sono piccole variazioni transitorie del tracciato EEG indotte da uno stimolo e a questo temporalmente legate. Si possono considerare costituite da una risposta esogena, legata all'elaborazione primaria dello stimolo, e da una endogena, legata a processi cognitivi che esso innesca.

Al soggetto vengono presentati una serie di stimoli in successione. Dirigendo la propria attenzione verso l'uno o l'altro di questi stimoli, egli varia inconsapevolmente l'entità dei processi cognitivi che lo stimolo provoca e quindi la componente endogena degli ERP indotti. Il segnale EEG registrato viene elaborato da un classificatore che individua qual è lo stimolo cui seguono potenziali ERP con una marcata componente endogena. Tale stimolo è presumibilmente quello su cui il soggetto è concentrato. Costui ha quindi la possibilità di comunicare con la macchina destinando la propria attenzione all'uno o all'altro degli stimoli che gli vengono presentati. Se tali stimoli sono costituiti da lettere il soggetto può comporre un testo, se sono frecce il soggetto può muovere un puntatore sullo schermo e così via.

In questa tesi viene proposto un metodo di addestramento del classificatore tramite algoritmi evoluzionistici. La fase di classificazione è preceduta da un pretrattamento che consiste nel calcolo della *trasformata wavelet continua*

(CWT) di ciascun canale EEG per un determinato numero di scale e istanti di tempo. La risposta ERP relativa a ciascuno stimolo è quindi rappresentata da una matrice tridimensionale di features in cui un indice rappresenta il canale EEG, uno la scala della CWT e uno l'istante di tempo.

Questa enorme mole di *features* rende necessario uno stadio di selezione delle stesse al fine di utilizzarne un sottoinsieme costituito solo da quelle più informative. È stato scelto un approccio di tipo *wrapper*, il quale prevede che la selezione delle features avvenga contestualmente all'addestramento del classificatore.

Come già accennato questo addestramento viene effettuato tramite *algoritmi evoluzionistici* ovvero metodi che si ispirano alle leggi della selezione naturale di Darwin. Viene creata una popolazione di classificatori e di ciascuno di essi vengono valutate le prestazioni nel classificare una serie di esempi. Tali prestazioni incidono sulla probabilità che l'individuo ha di procreare. Questo fa sì che con il succedersi delle generazioni le caratteristiche utili alla classificazione tendano a diffondersi nella popolazione generando individui sempre migliori. In questo caso migliori significa più capaci di distinguere i potenziali ERP relativi allo stimolo cui il soggetto presta attenzione da quelli relativi agli stimoli che gli sono indifferenti.

L'approccio descritto è stato valutato su dati resi disponibili nel corso di una competizione (*BCI competition*) svoltasi nel corso del 2003. I risultati ottenuti sono simili a quelli dei metodi usciti vincitori dalla competizione.

Sulla scia di tale risultato è stato realizzato un intero sistema per la presentazione degli stimoli, l'acquisizione del segnale EEG e la classificazione in tempo reale. Tale sistema consente di spostare, seppur in modo lento e grossolano, un puntatore sullo schermo di un personal computer.

Capitolo 1

Introduzione

1.1 Brain-Computer Interfaces (BCI)

1.1.1 Aspetti generali

Le *Brain-Computer Interfaces* (BCI) sono interfacce cervello-computer che forniscono all'utente un canale di comunicazione che prescinde dalle comuni vie efferenti del sistema nervoso centrale. Nelle comuni interfacce uomo-macchina lo scambio di informazioni tra la mente del soggetto ed il dispositivo artificiale avviene tramite l'intermediazione del nostro sistema sensoriale e motorio e di un adeguato insieme di attuatori e sensori ergonomici di cui il dispositivo deve essere dotato. L'intenzione del soggetto viene quindi tradotta in un comando (la pressione di un pulsante, la pronuncia di un comando vocale, il movimento di una cloche, e così via) che poi deve essere interpretato dal dispositivo artificiale. Nelle Brain-Computer Interfaces questo passaggio intermedio viene saltato e l'intenzione del soggetto viene direttamente riconosciuta ed attuata dalla macchina. Si tratta quindi di un'interfaccia diretta tra il cervello del soggetto ed il computer.

1.1.2 Possibili applicazioni

Il principale stimolo allo studio e all'implementazione di BCIs risiede nella speranza di permettere una qualche forma di comunicazione a persone con gravi patologie che non consentono loro di muovere alcun muscolo in modo volontario. Poiché ogni tipo di comunicazione di cui siamo capaci – il movimento, il linguaggio, l'espressione del volto – richiede il controllo della

muscolatura volontaria, il venir meno di tale controllo pregiudica non solo il movimento ma, e forse ciò è anche più grave, ogni forma di comunicazione.

Una di queste patologie è la sclerosi laterale amiotrofica (ALS, *Amyotrophic Lateral Sclerosis*), una malattia che colpisce i motoneuroni (le cellule nervose che comandano il movimento dei muscoli). Letteralmente significa: raggrinzimento (sclerosi) della porzione laterale (laterale) del midollo spinale con perdita del trofismo muscolare (amiotrofica). Colpisce generalmente adulti oltre i 40 anni e nella sua prima fase si manifesta con alterazioni motorie, affaticamento degli arti, difficoltà di parola, crampi muscolari. Con il progredire della malattia sempre più muscoli vengono interessati mentre le capacità intellettive e sensoriali rimangono intatte. Il malato è condannato ad assistere lucidamente alla perdita progressiva delle proprie capacità motorie. Nello stadio avanzato la malattia porta a paralisi completa degli arti, impossibilità di deglutire, masticare, parlare e respirare (diventa necessaria la ventilazione assistita).

Finché la paralisi non è totale è possibile utilizzare alcuni gruppi muscolari al fine di controllarne altri (ad esempio i muscoli extraoculari per controllare un sintetizzatore vocale oppure i muscoli dorsali per controllare un arto). Quando la paralisi è totale (come in caso di ALS allo stadio avanzato o lesione del tronco encefalico) un approccio di questo tipo è inapplicabile ed è necessario trovare un canale di comunicazione alternativo indipendente dal controllo muscolare come le interfacce cervello-computer.

1.2 Basi fisiologiche

1.2.1 Il neurone

Il neurone (o cellula nervosa) è l'unità funzionale elementare del sistema nervoso. Il numero di neuroni presenti nel corpo umano è dell'ordine di 10^{11} e, poiché questi risultano tra loro fittamente interconnessi, il numero di interconnessioni si aggira intorno a 10^{14}. Alcuni dei neuroni presenti nel nostro organismo sono dislocati in posizione periferica, ma la maggior parte di essi è concentrata nel sistema nervoso centrale (SNC).

I neuroni sono cellule eccitabili capaci di ricevere e trasmettere impulsi elettrici ma la loro caratteristica più interessante è la capacità di elaborare

tali segnali. Sebbene l'operazione svolta da ciascun neurone sui segnali in ingresso sia in fin dei conti elementare, il risultato complessivo dell'interazione di più cellule nervose è sorprendente. È grazie a reti complesse di neuroni che possiamo svolgere tutte le attività mentali di cui siamo capaci, tra cui l'interpretazione di percezioni sensoriali, la programmazione di movimenti, la memoria, il linguaggio, l'intuizione nonché l'estrazione di contorni, regioni e oggetti da immagini.

La cellula nervosa può essere scomposta in tre parti: il *corpo cellulare*, i *dendriti* e l'*assone*. Nel corpo cellulare sono presenti i comuni organuli cellulari ed una struttura portante, che prende il nome di citoscheletro. I dendriti sono prolungamenti che si dipartono dal corpo cellulare, sono spesso brevi, numerosi e ramificati. Il loro compito è di estendere la superficie di ricezione del neurone e possono essere visti come gli ingressi della cellula nervosa. L'assone, che può essere considerato l'uscita del neurone, è un prolungamento del corpo cellulare e consente di portare l'impulso nervoso a distanza. La velocità di conduzione lungo l'assone dipende dalla sua sezione ed aumenta notevolmente se la sua superficie è ricoperta di una guaina mielinica. L'assone termina con un rigonfiamento detto bottone sinaptico. Le sinapsi sono delle interfacce che consentono la propagazione dell'impulso al neurone successivo.

In condizioni di riposo il potenziale misurato tra l'interno e l'esterno della membrana è negativo. Quando i segnali raccolti dai dendriti superano una certa soglia si innescano fenomeni spiccatamente non lineari che portano temporaneamente il potenziale di membrana dal suo valore a riposo ad un valore positivo. Questo transitorio è chiamato impulso nervoso o potenziale d'azione. Il valore di tensione raggiunto e la durata sono indipendenti dall'eccitazione. Per questo motivo si dice che il neurone ha una risposta "tutto o nulla" e l'informazione, non potendo essere codificata in modo analogico, risiede nella frequenza di ripetizione degli impulsi nervosi (il corrispettivo tecnologico potrebbe essere la modulazione FSK). L'impulso nervoso viene propagato lungo la membrana dell'assone per poi raggiungere il bottone sinaptico dove provoca il rilascio di una sostanza chimica detta neurotrasmettitore. Questo raggiunge la membrana postsinaptica del neurone successivo provocandone una variazione transitoria del potenziale. Nel caso tale variazione sia positiva si parla di *potenziale postsinaptico eccitatorio* (EPSP)

altrimenti di *potenziale postsinaptico inibitorio* (IPSP).

1.2.2 Il sistema nervoso centrale

Il sistema nervoso centrale è costituito dall'encefalo e dal midollo spinale. L'encefalo pesa circa 1300 grammi ed è contenuto all'interno della scatola cranica. È l'organo preposto al controllo e alla regolazione delle funzioni del nostro corpo, all'elaborazione degli stimoli sensoriali, alla programmazione di risposte motorie ed alle funzioni mentali superiori tra cui la memoria, il ragionamento, l'astrazione, il linguaggio e molte altre. È costituito da tre parti: *cervello*, *cervelletto* e *tronco encefalico*.

Il cervello è un organo di forma ovoidale, di colore grigiastro e consistenza molle. È costituito da due emisferi ciascuno dei quali può essere suddiviso in più regioni: frontale, parietale, temporale, occipitale, limbico e dell'insula. I due emisferi, pur anatomicamente identici, non lo sono dal punto di vista funzionale.

La superficie esterna del cervello è detta corteccia cerebrale. È caratterizzata da numerosi *giri* e *solchi* che ne aumentano la superficie effettiva. È costituita da sei strati di cellule nervose ed è di colore grigio. La corteccia cerebrale può essere suddivisa in numerose aree in base alle funzioni a cui sovrintendono (area motoria, area visiva primaria, area preposta al linguaggio e così via). Le cellule nervose presenti nella corteccia cerebrale vengono suddivise in *piramidali* e *non piramidali* in base alla forma del corpo cellulare.

1.2.3 L'elettroencefalogramma

L'elettroencefalogramma registra, in modo non invasivo, gli effetti sullo scalpo dell'attività dei neuroni all'interno del cranio. I potenziali misurati in superficie sono il risultato dell'attività dei neuroni piramidali corticali posti in corrispondenza dell'elettrodo. L'EEG misura la differenza di potenziale tra un elettrodo attivo, posto al di sopra dell'area di interesse, e un elettrodo indifferente, collocato ad una certa distanza dal primo.

Il contributo elettrico di ciascun neurone è infinitamente piccolo e per di più il segnale deve attraversare alcuni strati di tessuto (meningi, liquor, ossa del cranio, pelle) prima di raggiungere gli elettrodi. È necessaria pertanto la contemporanea attivazione di migliaia di neuroni affinché il segnale EEG

sia rilevabile. Questo ci fa capire come l'ampiezza del segnale registrato sia indice del grado di sincronizzazione dell'attività dei neuroni implicati. Uno stesso numero di neuroni eccitati può indurre segnali risultanti profondamente diversi a seconda di quanto la loro attività è sincronizzata. Nel caso sia poco sincronizzata i diversi contributi tendono ad annullarsi ed il segnale registrato sarà molto irregolare e di ampiezza minima. Nel caso l'attività sia sincronizzata i singoli contributi tendono a sommarsi ed il segnale sarà molto più lento, intenso e regolare.

L'origine più ovvia del segnale elettroencefalografico potrebbe sembrare il potenziale d'azione delle cellule nervose ma in realtà questo contribuisce solo in minima parte. Infatti il segnale EEG rilevato è quasi esclusivamente prodotto dai potenziali postsinaptici (eccitatori o inibitori) e non dai potenziali d'azione (ved. [2]). Questo perché i potenziali d'azione hanno durata molto breve (circa un millisecondo) e quindi è necessario un alto grado di sincronizzazione affinché questi si possano sommare. Le correnti sinaptiche hanno invece una durata compresa tra 10 e 40 ms che consente a questi potenziali di sommarsi più efficacemente dei rispettivi potenziali d'azione e quindi di creare differenze di campi elettrici sufficientemente ampie da poter essere rilevate.

La risoluzione temporale dell'EEG è molto buona mentre quella spaziale è piuttosto scarsa (nell'ordine del centimetro) e dipende dal numero di elettrodi utilizzati. Attualmente il numero di elettrodi può arrivare ad oltre un centinaio. Esiste un sistema internazionale standard di posizionamento degli elettrodi chiamato *sistema 10-20* che prevede l'utilizzo di 19 elettrodi attivi, disposti come in figura 1.1, riferiti al cortocircuito dei due elettrodi posti sui lobi auricolari.

Il segnale EEG ha uno scarso rapporto segnale-rumore. Segnali elettromiografici dovuti al movimento oculare o ad altri muscoli rappresentano disturbi addirittura maggiori del segnale elettroencefalografico che si intende registrare. È pertanto opportuno che il soggetto sia seduto in posizione comoda e, possibilmente, con un sostegno per la testa. È bene inoltre ridurre gli stimoli estranei all'esperimento e pertanto sarebbe auspicabile un ambiente insonorizzato e tranquillo.

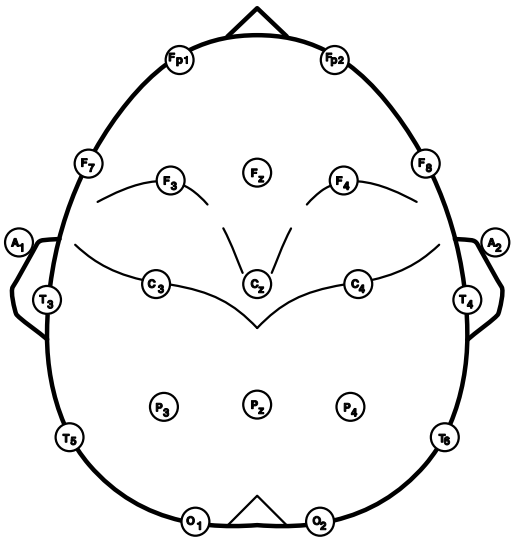

Figura 1.1: Sistema internazionale 10-20 di posizionamento degli elettrodi

1.2.4 L'attività di fondo

Il cervello è un organo in continua attività in quanto interagisce ininterrottamente con l'ambiente esterno, con se stesso e con gli atri organi. Lo studio dell'attività di fondo si occupa del segnale elettroencefalografico registrato quando il soggetto non è sottoposto a particolari stimoli esterni. L'attività spontanea può rivelare lo stato mentale (rilassamento, attenzione, etc.), il livello di coscienza (sonno, profondità dell'anestesia, livello di coma, etc.) o anche patologie (tumori cerebrali, epilessia, ictus, etc.).

Le oscillazioni (dette ritmi) del segnale EEG di fondo sono indice dell'intensità dell'attività cerebrale. Ritmi con alta frequenza e bassa ampiezza sono correlati all'attenzione, alla veglia o al sonno R.E.M. (la fase in cui si sogna) mentre ritmi con bassa frequenza e grande ampiezza indicano stadi di sonno profondo o coma. Questo perché, quando la corteccia è intensamente occupata nell'elaborazione delle informazioni, i neuroni corticali sono molto attivi ma anche piuttosto desincronizzati in quanto ciascuno di essi è impegnato nello svolgere una piccola parte di un compito complesso. Diversamente, durante stadi di sonno profondo, il cervello deve svolgere un'attività nettamente inferiore e molte aree sono eccitate ritmicamente da uno stimolo comune e lento. Ciò significa una maggiore sincronia e quindi i contributi dei singoli neuroni tendono a sommarsi determinando un segnale EEG più lento

ed ampio.

I ritmi del segnale EEG sono indicati con le lettere dell'alfabeto greco:

Onde delta (δ). Sono caratteristiche del sonno profondo e di condizioni patologiche come forme tumorali o coma. Sono onde di frequenza inferiore a 4 Hz, spesso di grande ampiezza ($> 75\ \mu V_{pp}$).

Onde teta (θ). Si ritrovano in stati di stress emozionale, in prossimità del passaggio da veglia a sonno e talvolta durante il sonno REM. La frequenza è compresa tra 4 e 8 Hz e l'ampiezza si aggira attorno ai $100\ \mu V_{pp}$.

Onde alfa (α). Onde ritmiche presenti durante stati di rilassamento ad occhi chiusi. Scompaiono durante il sonno e durante la veglia attenta. La frequenza è compresa tra 8 e 14 Hz e l'ampiezza è solitamente minore di $75\ \mu V_{pp}$.

Onde beta (β). Sono indice di attività mentale intensa e compaiono durante la veglia, il sonno leggero e il sonno REM. La frequenza è tra 14 e 30 Hz e l'ampiezza inferiore a $30\ \mu V_{pp}$.

Onde gamma (γ). Onde a frequenza superiore a 30 Hz e ampiezza inferiore a $30\ \mu V_{pp}$.

1.2.5 Event Related Potentials

Il potenziale evento-correlato (ERP, *event-related potential*) è una variazione del tracciato EEG in seguito ad una stimolazione esterna od interna (ad esempio l'intenzione di compiere un movimento). Viene usato il termine potenziale evento-correlato (ERP) per sottolinearne l'aspetto cognitivo rispetto al potenziale evocato (EP, *evoked potential*) focalizzato più sull'elaborazione primaria dello stimolo (ved. [3]).

I potenziali evento-correlati sono costituiti da particolari forme d'onda caratterizzate da una serie di deflessioni positive o negative. Ciascuna di esse è comunemente chiamata componente ed è caratterizzata da una polarità (positiva o negativa), un'ampiezza tipica ed un tempo di latenza tipico. Il

tempo di latenza è il tempo che intercorre tra lo stimolo e la componente stessa e, in prima approssimazione, è legato al livello di astrazione del processo alla base della componente.

Le componenti sono state suddivise in esogene ed endogene. Le componenti esogene hanno latenza minore e rappresentano la risposta allo stimolo sensoriale. Dipendono dalla natura fisica dello stimolo mentre non risentono del significato che lo stimolo assume per il soggetto. Al contrario le componenti endogene non dipendono dalle caratteristiche fisiche dello stimolo (come intensità o durata) ma risentono decisamente del significato che lo stimolo assume per il soggetto in relazione all'attività che sta svolgendo.

Tra le componenti endogene più studiate c'è sicuramente quella chiamata P3 (in quanto la terza onda positiva che segue lo stimolo) o anche P300 (in quanto ha una latenza di circa 300 ms). Questa onda viene evocata da una procedura, un metodo, chiamato *odd-ball paradigm*. Questo consiste nel presentare al soggetto una successione di stimoli sensoriali solitamente di natura visiva o acustica. La maggior parte di questi stimoli è di tipo *non-target*, il che significa che il soggetto è tenuto ad ignorarli. Saltuariamente, però, può presentarsi uno stimolo diverso, detto *target*, cui il soggetto è tenuto a rispondere ad esempio premendo un pulsante o contando il numero delle volte che si presenta.

Le risposte evocate dagli stimoli target presentano una chiara componente P300. Questa è dovuta agli aspetti che differenziano lo stimolo target dal non-target: è raro e *task relevant* ovvero significativo, importante ai fini del compito che il soggetto sta svolgendo.

Ci sono diverse teorie circa la natura del P300. La più nota e discussa è quella di Donchin e Coles [4] che ipotizzano che il P300 sia la manifestazione di un processo di aggiornamento della memoria di lavoro. Più precisamente il P300 rifletterebbe l'attività compiuta ogni qualvolta il nostro modello interno dell'ambiente esterno deve essere aggiornato.

1.3 Diversi approcci alle BCI

Allo scopo di chiarire il concetto di BCI, di definirne le caratteristiche principali e di fare il punto sullo stato dell'arte, alcuni tra i più attivi ricercatori in questo campo si sono riuniti nel giugno 1999 al Rensselaerville Institute

presso Albany, New York [5]. In questa occasione è stata data la seguente definizione: “A brain-computer interface is a communication system that does not depend on the brain’s normal output pathways of peripheral nerves and muscles”.

Un’interfaccia di questo tipo, che prescinda dalle comuni “uscite” del sistema nervoso centrale, è possibile grazie al fatto che determinate caratteristiche dell’attività cerebrale variano in relazione allo stato in cui il soggetto si trova, all’attività che sta svolgendo o all’interesse nei confronti di determinati stimoli esterni. Tale variazione può essere rilevata con strumenti di diagnostica cerebrale (EEG, MEG, PET, fMRI o tramite elettrodi impiantati sulla corteccia) ed elaborata da opportuni classificatori.

Come altri canali di comunicazione, queste interfacce hanno ingressi, uscite e algoritmi di trasferimento che trasformano i primi nelle seconde. Il funzionamento di queste interfacce è basato sull’interazione di due componenti: il cervello del soggetto che produce gli ingressi del sistema (ovvero segnali fisiologici misurabili) ed il sistema stesso che analizza e traduce questi segnali in uscite, comandi.

Un semplice esempio di BCI potrebbe sfruttare il livello di attenzione del soggetto per implementare un canale di comunicazione binario tra la sua mente ed un computer. È noto che il ritmo α (ved. par. 1.2.4) del segnale elettroencefalografico compare in caso di veglia rilassata e si attenua decisamente in caso di veglia attenta. Associando la presenza del ritmo α al valore binario 1 e la sua assenza allo 0, il soggetto potrebbe imparare a comunicare evocando pensieri che lo rilassano o altri che ne risvegliano l’attenzione.

Talvolta vengono spacciate per BCI interfacce uomo-computer che fanno uso di segnali diversi da quelli cerebrali come ad esempio quelli indotti dal movimento oculare o della muscolatura del volto.

I diversi approcci alla realizzazione di BCI possono essere suddivisi in invasivi e non invasivi. L’utilizzo di elettrodi impiantati sulla corteccia cerebrale consente di rilevare i potenziali di azione di ristretti gruppi di neuroni. Questo comporta alcune controindicazioni tra cui la necessità di sottoporre il soggetto ad un intervento tutt’altro che banale ed il fatto che dopo alcuni mesi il segnale rilevabile si attenua in quanto il cervello tende ad isolare il corpo estraneo. In alternativa ci sono metodi non invasivi che si basano su

tecniche quali la risonanza magnetica nucleare, la magnetoencefalografia e l'elettroencefalografia. Tra queste la più utilizzata è sicuramente l'ultima. Malgrado le prime due abbiano dei vantaggi (ad esempio in termini di risoluzione spaziale) hanno il grosso difetto di utilizzare tecnologie costose ma soprattutto ingombranti e tutt'altro che portatili.

In generale i diversi tipi di BCI si basano sul riconoscimento da parte del sistema di determinate caratteristiche dell'attività cerebrale. Questo richiede l'addestramento di un classificatore tramite una serie di esempi. BCI, in cui la sola parte artificiale viene addestrata, sfruttano l'approccio detto *pattern recognition*. Altri, che richiedono un addestramento anche del soggetto, sfruttano l'approccio *operant conditioning*. Tale apprendimento avviene fornendo al soggetto un *feedback* che gli consente di trovare la "strategia" migliore per ottenere l'uscita desiderata.

1.3.1 Il *Graz-BCI*

Presso l'università di Graz, Gert Pfurtscheller ed i suoi colleghi hanno realizzato un sistema BCI utilizzando le alterazioni dell'elettroencefalogramma in seguito all'immaginazione del movimento di un arto [6]. Infatti, tanto l'effettiva esecuzione di un movimento che la sola immaginazione, provocano l'attenuarsi di determinate bande del segnale EEG e l'intensificarsi di altre.

Il fatto che il soggetto immagini il movimento rappresenta uno stimolo interno. In seguito ad uno stimolo, interno o esterno, possono verificarsi variazioni dell'ampiezza di determinati ritmi EEG. L'aumento percentuale dell'ampiezza di un certo ritmo dopo l'evento rispetto ad un periodo antecedente, è chiamato ERS (*event-related synchronization*) mentre una sua attenuazione è detta ERD (*event-related desynchronization*).

Posizionando gli elettrodi sullo scalpo in corrispondenza dell'area motoria della corteccia cerebrale, è possibile rilevare la presenza di ERD sull'emisfero opposto a quello dell'arto che il soggetto immagina di muovere.

Il protocollo utilizzato prevede che tanto il sistema che il soggetto debbano essere sottoposti ad addestramento. In una prima fase al soggetto viene impartito l'ordine di immaginare il movimento di uno dei due arti a seconda della direzione di una freccia che compare sullo schermo di un computer. I parametri del classificatore vengono ottimizzati al fine di massimizzare il

riconoscimento. Nella fase successiva è il soggetto ad essere addestrato tramite feedback. Sullo schermo viene mostrata una barra che si estende tanto più verso destra quanto maggiore è l'ERD per l'emisfero sinistro e viceversa. Il soggetto può quindi migliorare la "strategia" che utilizza per immaginare il movimento cercando di spostare quanto più possibile la barra dal lato desiderato.

1.3.2 Ricerche al Wadsworth Center

Al Wadsworth Center, Wolpaw e collaboratori hanno sviluppato un sistema BCI basato sulla possibilità di imparare a variare il ritmo μ dell'EEG [7].

Il ritmo μ ha una frequenza attorno ai 10 Hz e un'ampiezza minore di 50 μV_{pp}. Malgrado queste caratteristiche siano simili a quelle del ritmo α si differenzia da questo in quanto fortemente legato alle funzioni delle aree motoria e sensoriale della corteccia. Il ritmo μ scompare in seguito all'esecuzione di un movimento o alla percezione di sensazioni tattili.

Il soggetto può imparare a regolare volontariamente i propri ritmi μ e β tramite *biofeedback*. Wolpaw e colleghi hanno sfruttato questa possibilità al fine di muovere un cursore sullo schermo.

1.3.3 Il *Thought Translation Device*

Niels Birbaumer e i suoi colleghi dell'università di Tübingen hanno realizzato, e provato con pazienti affetti da ALS, un sistema BCI basato sull'autoregolazione degli *Slow Cortical Potentials* (SCP) [8].

Gli SCP sono variazioni generalizzate del livello di depolarizzazione dei dendriti superficiali della corteccia. Sono rilevabili nel segnale EEG ed il soggetto può imparare a controllarne il segno.

Una prima fase di addestramento prevede che il soggetto impari a controllare gli SCP. Raggiunto un tasso di riconoscimenti corretti del 75%, il soggetto passa alla fase successiva in cui impara a selezionare delle lettere per comporre frasi.

1.3.4 Il *Donchin speller*

Questo tipo di BCI sfrutta le caratteristiche della componente P300 del potenziale visivo evocato (par. 1.2.5) al fine di immettere caratteri. Fu pre-

sentato per la prima volta nel 1988 da Donchin e dal suo assistente Farwell [9].

Una matrice 6x6 di caratteri viene mostrata sullo schermo (fig. 4.1 pag. 48). Le righe e le colonne della matrice vengono evidenziate in rapida successione. Il soggetto è istruito affinché presti attenzione al carattere che intende immettere, ad esempio contando quante volte viene illuminato.

Ogni riga o colonna evidenziata costituisce uno stimolo visivo capace di indurre una risposta esogena. Al contrario, solo la riga e la colonna che contengono il carattere da immettere presenteranno anche una risposta endogena. Per tale motivo riconoscere la risposta endogena, ed in particolare la componente P300, consente di risalire alla lettera prescelta (si veda il par. 4.1 e [10]).

1.3.5 Elettrodi impiantati

Presso l'università di Atlanta, Kennedy e colleghi hanno utilizzato elettrodi impiantati nella corteccia al fine di controllare il movimento di un puntatore sullo schermo [11]. Ciò consente, ad esempio, di immettere caratteri con una tastiera a video o di utilizzare un browser per internet.

Inizialmente il mouse si trova nell'angolo superiore sinistro. I potenziali di azione registrati da un elettrodo sono convertiti in impulsi di cui viene considerata la velocità di ripetizione. Un aumento di questa provoca lo spostamento verso destra del puntatore. Con procedimento analogo, un altro elettrodo controlla lo spostamento verso il basso mentre un terzo emula la pressione del pulsante del mouse. Ogni volta il puntatore viene riposizionato nell'angolo superiore sinistro.

1.3.6 BCI tramite fMRI

Un gruppo misto di ricercatori delle università di Tübingen e Maastricht ha sviluppato un sistema BCI tramite risonanza magnetica nucleare funzionale [12].

Sfruttando le differenti proprietà magnetiche dell'ossiemoglobina e della desossiemoglobina (ovvero, rispettivamente, dell'emoglobina legata o meno all'ossigeno) è possibile risalire al bilancio tra le due in ogni area del cervello. Poiché quando un'area viene attivata le viene resa disponibile una quantità

di sangue ossigenato maggiore del suo consumo effettivo, la distribuzione di ossiemoglobina è rappresentativa dell'attività di ciascuna area cerebrale. Questa tecnica, chiamata BOLD (*Blood Oxygenation Level Dependent*), consente quindi, con un ritardo di circa 1-2 secondi, di risalire ad una mappa dell'attività cerebrale.

Il sistema BCI è stato realizzato sfruttando questa tecnica. La differenza del segnale BOLD medio di due regioni, l'area SMA (*supplementary motor area*) e la PPA (*parahippocampal place area*), viene mostrato al soggetto tramite biofeedback. Il soggetto può apprendere a controllare tale parametro eseguendo compiti che coinvolgono maggiormente l'una o l'altra area.

A fini dimostrativi questo segnale è stato utilizzato affinché due soggetti, all'interno di due scanner, potessero controllare la posizione della propria racchetta in una sorta di ping pong virtuale.

Capitolo 2

Wavelets

2.1 Analisi tempo-frequenza

2.1.1 Trasformata di Fourier

Tra le varie modalità di analisi dei segnali quella in frequenza è tra le più potenti e diffuse. Permettendo uno studio della periodicità del segnale, risulta spesso utile per capire il meccanismo fisico che ne sta alla base.

Data una funzione $x(t) \in L^2(\mathbb{R})$[1], la sua trasformata di Fourier, $X(f)$, è definita tramite la coppia di equazioni

$$X(f) = F\{x(t)\} \triangleq \int_{\mathbb{R}} x(t) e^{-j2\pi ft} \mathrm{d}t \tag{2.1}$$

$$x(t) = F^{-1}\{X(f)\} \triangleq \int_{\mathbb{R}} X(f) e^{j2\pi ft} \mathrm{d}t. \tag{2.2}$$

La trasformata di Fourier può essere vista come la scomposizione di un segnale nell'integrale di tutte le armoniche che lo costituiscono. Sotto questo punto di vista $X(f)$ rappresenta l'ampiezza e la fase dell'armonica a frequenza f. Dall'equazione (2.1) si vede come la funzione $X(f)$ sia il prodotto interno tra la funzione $x(t)$ e la funzione complessa $e^{j2\pi f\, t}$, ovvero

$$X(f) = \left\langle x(t), e^{j2\pi f\, t} \right\rangle. \tag{2.3}$$

La trasformata di Fourier è uno strumento estremamente potente ed è impiegato in numerose applicazioni in ogni campo della scienza. Tra queste c'è lo studio dell'elettroencefalogramma (ved. [13], [14], [15], [16]) in cui

[1] L'appartenenza di x a $L^2(\mathbb{R})$ è una condizione sufficiente ma non necessaria per l'esistenza della trasformata di Fourier.

viene utilizzato principalmente per determinare come l'energia del segnale sia distribuita tra i vari ritmi descritti al paragrafo 1.2.4. Ciononostante è necessario sottolineare due svantaggi della trasformata di Fourier. Innanzitutto per poter essere applicata ad un processo è necessario che questo sia stazionario, ipotesi tutt'altro che verificata per l'EEG, specialmente nel caso dell'analisi dei potenziali evocati. Inoltre, per il principio di indeterminazione (ved. [17]), avendo una esatta localizzazione in frequenza presenta una completa delocalizzazione temporale. Questo la rende particolarmente inadatta allo studio di transitori che, invece, sono spesso fonte d'informazione nello studio dell'EEG e, in particolare, in quello dei potenziali evocati che possono essere visti come risposte transitorie ad uno stimolo esterno.

Nel caso di un segnale campionato, ovvero di una successione reale

$$x(n) = \{x_i \text{ con } i \in [0,\ N-1]\}$$

è possibile calcolare la cosiddetta trasformata di Fourier discreta (DFT). Questa e la sua inversa sono espresse come

$$X(k) = DFT\{x(n)\} \stackrel{\triangle}{=} \sum_{0}^{N-1} x(n) e^{-j2\pi k\, n/N} \tag{2.4}$$

$$x(n) = IDFT\{X(k)\} \stackrel{\triangle}{=} \sum_{0}^{N-1} X(k) e^{j2\pi k\, n/N} \tag{2.5}$$

dove la successione

$$X(k) = \{X_i \text{ con } i \in [0,\ N-1]\}$$

è a valori complessi e rappresenta il contributo dell'armonica con frequenza normalizzata[2] k/N. L'indeterminazione in frequenza è quindi f_s/N mentre quella temporale è estesa a tutti gli N campioni.

Poiché vale la relazione $X(k) = X^*(N-k)$, $X(k)$ è determinata da N coefficienti reali indipendenti[3] come $x(n)$. Non si ha infatti né perdita di informazione né ridondanza nel passaggio da $x(n)$ a $X(k)$ e viceversa.

[2]Definendo la frequenza normalizzata come $F = f/f_s$ dove f_s è la frequenza di campionamento.

[3]Se N è pari, $X(0)$ e $X(N/2)$ sono reali e gli altri $N-2$ elementi sono a due a due complessi coniugati. Se N è dispari, $X(0)$ è reale e gli altri $N-1$ coefficienti sono a due a due complessi coniugati. In ogni caso si hanno N coefficienti reali.

2.1.2 Short Time Fourier Transform

Per ovviare ai problemi che la trasformata di Fourier presenta (ipotesi di stazionarietà del processo e delocalizzazione temporale dei fenomeni transitori) è stata introdotta la Short Time Fourier Transform (STFT). Essa consiste nel calcolare la trasformata di Fourier su una opportuna finestra temporale scorrevole

$$F(f,\tau) = STFT\{f(t)\} \triangleq \int_{\mathbb{R}} f(t)\, g^*(t-\tau)\, e^{-j2\pi f\, t} \mathrm{d}t \tag{2.6}$$

e può essere riscritta come il prodotto interno

$$F(f,\tau) = \left\langle x(t), g(t-\tau)\, e^{j2\pi f\, t} \right\rangle . \tag{2.7}$$

Si ottiene quindi una trasformata da tempo a tempo-frequenza. La risoluzione nel tempo e nella frequenza è un elemento piuttosto critico e dipende dal tipo e dall'ampiezza della finestra temporale. Per quanto riguarda la forma della finestra, un'ottima scelta è la gaussiana in quanto questa ha una buona localizzazione sia nel tempo che in frequenza. In tal caso si parla di trasformata di Gabor.

Il concetto della STFT può essere applicato ad un segnale campionato. Data la successione $x(n)$ si prelevano sottosequenze di M campioni sovrapposte di $M/2$. Ciascuna di esse viene prima moltiplicata per una finestra (ad es. di Hamming) e poi ne viene calcolata la DFT. Anche in questo caso parametro cruciale è la scelta dell'ampiezza della finestra, ovvero M. Maggiore è M, maggiore è l'incertezza temporale in quanto la DFT viene calcolata ad intervalli di tempo maggiori. D'altronde minore è M e maggiore è l'incertezza in frequenza perché la DFT viene calcolata su una successione di lunghezza minore.

Rispetto alla trasformata di Fourier classica, la STFT consente di seguire l'evoluzione nel tempo dello spettro di un segnale. Considerando, ad esempio, un segnale sinusoidale modulato in frequenza da una sinusoide

$$x(t) = A\, \sin(2\pi f\, t) \quad \text{con } f = f_0 + \alpha \sin(\beta\, t). \tag{2.8}$$

si vede (fig. 2.1) come la trasformata di Fourier non metta assolutamente in risalto questa caratteristica ma presenti una banda di frequenze che si estende all'incirca tra i due estremi di f e completamente delocalizzata nel tempo.

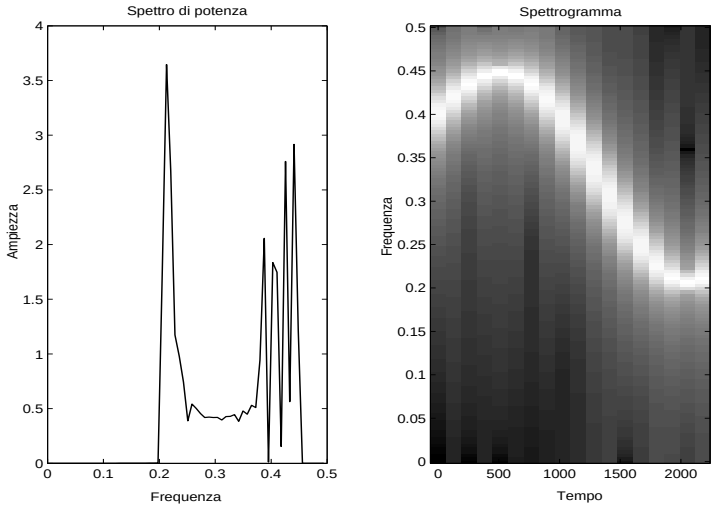

Figura 2.1: A sinistra lo spettro di potenza ($|X(f)|^2$) del segnale dato dall'espressione (2.8) calcolato tramite trasformata di Fourier. A destra il suo spettrogramma calcolato tramite STFT (l'ampiezza è codificata in scala di grigi dove al bianco corrisponde la massima ampiezza)

Al contrario la STFT, con una finestra opportunamente scelta, mostra una banda ristretta di frequenze che varia nel tempo con andamento sinusoidale.

La risoluzione temporale ed in frequenza della STFT sono vincolate dal principio di indeterminazione (ved. [17]) che afferma che si può migliorare l'una solo a scapito dell'altra

$$\sigma_t \cdot \sigma_f \geq \frac{1}{4\pi}. \tag{2.9}$$

Per una finestra gaussiana (trasformata di Gabor) vale l'uguaglianza. Il punto debole della trasformata STFT è di prevedere risoluzioni in frequenza e temporale costanti (in quanto la finestra è unica) per tutte le componenti di frequenza. Questo limite viene superato dalla trasformata Wavelet.

2.2 Trasformata Wavelet

2.2.1 Caratteristiche generali

Le equazioni (2.3) e (2.7) mostrano come la trasformata di Fourier e la STFT possano essere viste come il prodotto interno del segnale di partenza con una funzione "madre" che nel primo caso è un'esponenziale complessa, nel

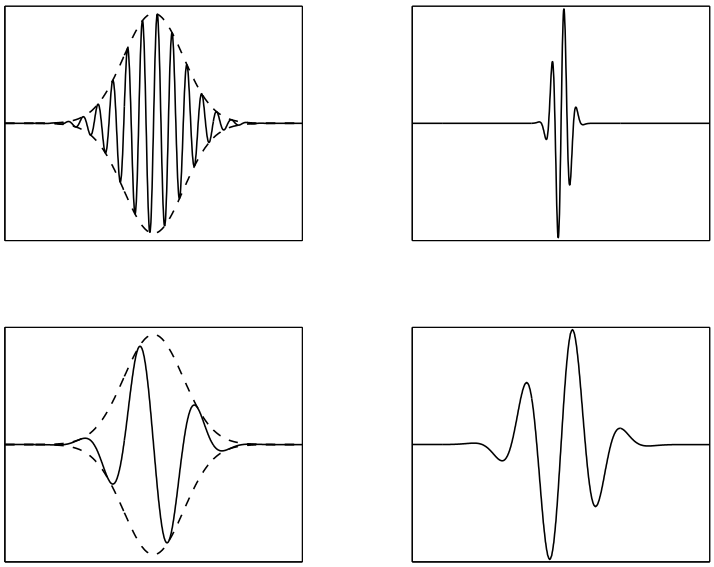

Figura 2.2: Sul lato sinistro la funzione "madre" della trasformata di Gabor per due diverse frequenze. Sul lato destro la *mother wavelet* "gaus7" a due diverse scale

secondo un'esponenziale complessa moltiplicata per una finestra temporale. Analogamente la *trasformata Wavelet* è definita come il prodotto interno tra il segnale ed una funzione detta *mother wavelet* opportunamente scalata e traslata nel tempo:

$$C_{a,b} \triangleq \left\langle x(t), \frac{1}{\sqrt{a}}\psi\left(\frac{t-b}{a}\right)\right\rangle = \int_{\mathbb{R}} s(t)\frac{1}{\sqrt{a}}\psi^*\left(\frac{t-b}{a}\right)\mathrm{d}t. \tag{2.10}$$

Il parametro $b \in \mathbb{R}$ rappresenta la traslazione temporale della wavelet mentre $a \in \mathbb{R}^+$ rappresenta la scala ovvero di quanto la funzione madre è dilatata del tempo. Il concetto di scala rappresenta, invertito, ciò che nella trasformata di Fourier è la frequenza. La funzione madre ψ appartiene a $L^2(\mathbb{R})$, è a media nulla e tende rapidamente a zero per $|t| \to \infty$. Altre proprietà matematiche dipendono dalla famiglia[4] wavelet, tra queste l'ortogonalità, la regolarità, la simmetria, l'avere supporto compatto e molte altre.

La differenza sostanziale tra la STFT e la trasformata wavelet è che mentre nella prima al variare della frequenza varia solo l'esponenziale complessa mentre la finestra temporale rimane invariata, nella seconda l'intera funzione madre varia con la scala come rappresentato nella figura 2.2.

[4]Le famiglie wavelet riuniscono funzioni ψ con origine e proprietà matematiche simili.

Questo significa che la risoluzione temporale (e di conseguenza quella in frequenza) variano con la scala: componenti ad alta frequenza hanno una minore risoluzione in frequenza ma una maggiore risoluzione temporale e viceversa.

2.2.2 Discrete Wavelet Transform

Nella trasformata wavelet discreta a e b assumono valori discreti e solitamente sono una combinazione delle potenze di 2:

$$a = 2^m \quad b = n\,2^m \quad \text{con } m, n \in \mathbb{Z} \tag{2.11}$$

Con questa scelta di a e b, le wavelets ottenute formano una base per lo spazio di Hilbert $L^2(\mathbb{R})$, la trasformata si chiama Dyadic Wavelet Transform e assume la forma

$$DWT_{m,n}\{f(t)\} \triangleq 2^{-\frac{m}{2}} \int f(t)\,\psi^*(2^{-m}t - n)\,dt \tag{2.12}$$

Solitamente per implementare tale operazione non si applica la definizione ma si usa un algoritmo introdotto da Mallat che consiste in una cascata di stadi costituiti da filtri in quadratura e decimatori. Questo schema era già utilizzato prima dell'introduzione delle wavelets ed era noto come *two channel subband coder*. Ogni stadio prevede la scomposizione del segnale in due componenti tramite due filtri FIR in quadratura, uno passa alto e uno passa basso, entrambi con frequenza di taglio pari alla metà della frequenza di Nyquist[5] del segnale in ingresso allo stadio. I segnali all'uscita dei filtri vengono decimati per 2 (fig. 2.3). L'uscita corrispondente al ramo con filtro passa alto è detta *dettaglio* mentre l'altra *approssimazione*. La decomposizione procede fino alla profondità desiderata scomponendo ulteriormente l'*approssimazione* (fig. 2.4). Con questo procedimento si ottengono i coefficienti della DWT dove il livello della decomposizione corrisponde a m mentre n è l'indice all'interno di ciascuna successione *dettaglio* o *approssimazione*.

Un altro tipo di trasformata discreta wavelet è la decomposizione Wavelet Packet (WP) che nasce come generalizzazione della DWT in quanto prevede la possibilità di decomporre ulteriormente sia il *dettaglio* che l'*approssimazione* come mostrato in figura 2.4.

[5]Definendo come frequenza di Nyquist la massima frequenza che non produce aliasing, ovvero la metà della frequenza di campionamento.

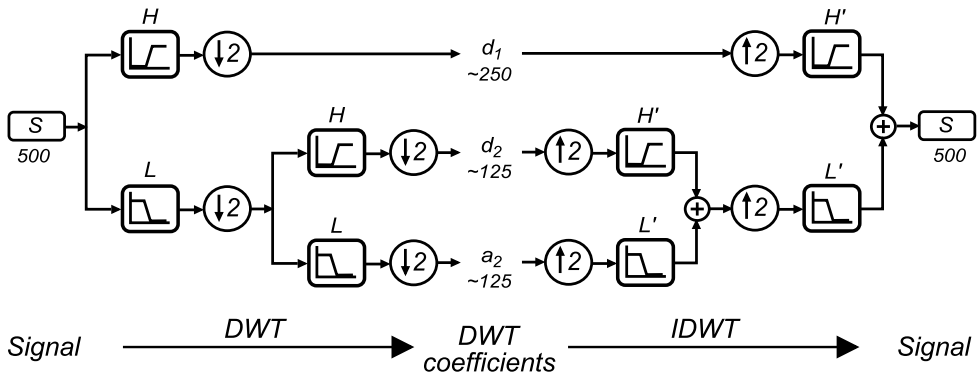

Figura 2.3: Decomposizione diadica a due stadi (a_j sono le approssimazioni, d_j i dettagli)

2.2.3 Continuous Wavelet Transform

La definizione data di trasformata wavelet (2.10) non fa ipotesi sulla natura continua o discreta del segnale o dei parametri a e b. Ovviamente, però, ogni volta che ci si occupa di elaborazione numerica dei segnali si ha a che fare con segnali campionati. Inoltre su un calcolatore reale non potremo far variare la scala a o il tempo b con continuità ma dovremo accontentarci di calcolare un numero finito di coefficienti $C_{a,b}$. Tuttavia anche in questo caso, malgrado né il segnale di partenza, né la trasformata siano continui, si parla di trasformata wavelet continua (CWT). Questo per differenziarla dalla trasformata wavelet discreta introdotta al paragrafo 2.2.2 e per sottolineare come a e b possano essere essere scelti con continuità senza sottostare alle relazioni (2.11).

Questo fa sì che CWT e DWT abbiano caratteristiche e proprietà diverse. È necessario scegliere di volta in volta l'una o l'altra a seconda del problema specifico. Prendiamo un segnale campionato con passo unitario e consideriamone la DWT e la CWT calcolata per traslazioni b intere (cioè sugli stessi istanti di campionamento del segnale) e scale a a piacere ottenendo così nel piano tempo–scala una griglia. Come accennato in precedenza, i valori di a e b impiegati nella DWT sono tali che le wavelets corrispondenti formino una base per $L^2(\mathbb{R})$. Ciò significa che i coefficienti trovati tramite la DWT consentono un'esatta ricostruzione del segnale di partenza senza essere ridondanti. Questo, in generale, non è vero per la CWT; infatti i coefficienti

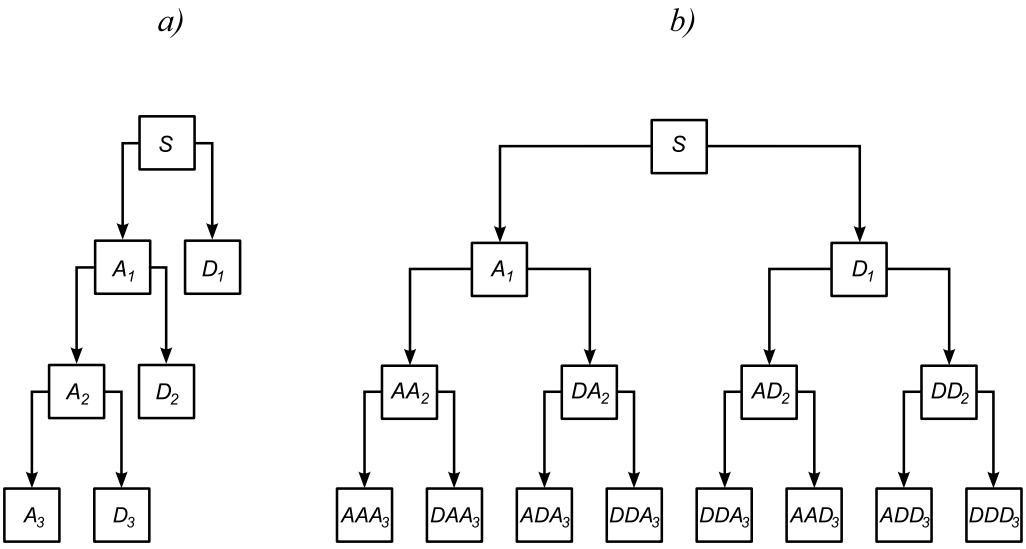

Figura 2.4: Confronto tra l'albero di decomposizione di (a) DWT e (b) Wavelet Packet (A sono le approssimazioni, D i dettagli)

ottenuti sono molti di più dei campioni del segnale di partenza e quindi sono ridondanti. Nel caso della DWT, inoltre, le operazioni di filtraggio e decimazione fanno sì che il passo si campionamento di ogni componente (*dettaglio* o *approssimazione*) sia sempre proporzionato al contenuto in frequenza del segnale. Ad esempio nel passaggio dal livello generico m al livello successivo $m+1$ (fig. 2.3) la scala a raddoppia (eq. (2.11)), ciò significa che la funzione wavelet con cui è convoluto il segnale è adatta a rilevare fenomeni con frequenza dimezzata rispetto al livello m: è perciò sufficiente una frequenza di campionamento che sia la metà di quella al livello m. Questo è ciò che accade nella DWT come mostrato in figura 2.3 e come può essere dedotto dall'espressione di b in (2.11). Nella CWT, al contrario, il passo di campionamento è costante per ogni scala e questo implica ridondanza. In caso di *denoising* o in caso *compressione* tale ridondanza rappresenta indubbiamente uno svantaggio e infatti viene preferita la DWT. In altri casi, invece, questa ridondanza può rappresentare un vantaggio in quanto rende i risultati della CWT più comprensibili e più facilmente interpretabili di quelli della DWT. Altro punto a favore della trasformata wavelet continua è che i coefficienti ottenuti possono essere rappresentati direttamente tramite una matrice cosa impossibile nel caso della DWT dove i vettori delle componenti ai vari livelli hanno lunghezza diversa. Infine la CWT ha il pregio di essere

tempo-invariante (almeno per traslazioni di un numero intero di campioni). La DWT non gode di questa proprietà, a meno di ricorrere a sue varianti quali la *Stationary Wavelet Transform* (ved. [18])

2.3 Algoritmo per il calcolo della trasformata wavelet continua

Per il calcolo della CWT è stato ripreso l'algoritmo descritto in [18].

Nella definizione di CWT (per semplicità prendiamo ψ reale)

$$C_{a,b} = \int_{\mathbb{R}} s(t) \frac{1}{\sqrt{a}} \psi\left(\frac{t-b}{a}\right) \mathrm{d}t \tag{2.13}$$

l'integrale su $\mathbb{R}$ può essere scomposto nella sommatoria su tutti gli intervalli di campionamento, dell'integrale calcolato su ciascun intervallo. Ovvero, per un passo di campionamento unitario, nel seguente modo

$$C_{a,b} = \sum_{k\in\mathbb{Z}} \int_{k}^{k+1} s(t) \frac{1}{\sqrt{a}} \psi\left(\frac{t-b}{a}\right) \mathrm{d}t. \tag{2.14}$$

Essendo s noto solo negli istanti di campionamento è necessario fare delle ipotesi sul suo andamento tra un istante di campionamento e l'altro. L'ipotesi più semplice è che sia costante a tratti ovvero, sempre nel caso di passo di campionamento unitario, che sia

$$s(t) = s(k) \quad \forall t \in [k, k+1)\,. \tag{2.15}$$

La (2.14) diventa quindi

$$\begin{aligned} C_{a,b} &= \frac{1}{\sqrt{a}} \sum_{k\in\mathbb{Z}} s(k) \int_{k}^{k+1} \psi\left(\frac{t-b}{a}\right) \mathrm{d}t & (2.16)\\ &= \frac{1}{\sqrt{a}} \sum_{k\in\mathbb{Z}} s(k) \left[\int_{-\infty}^{k+1} \psi\left(\frac{t-b}{a}\right) \mathrm{d}t - \int_{-\infty}^{k} \psi\left(\frac{t-b}{a}\right) \mathrm{d}t\right] & (2.17) \end{aligned}$$

Chiamando γ l'integrale di ψ si ha che

$$\int_{-\infty}^{k} \psi\left(\frac{t-b}{a}\right) \mathrm{d}t = \int_{-\infty}^{k-b} \psi\left(\frac{\tau}{a}\right) \mathrm{d}\tau = a \cdot \gamma\left(\frac{k-b}{a}\right) = a \cdot \gamma_a(k-b) \tag{2.18}$$

dove γ_a è definito

$$\gamma_a(t) \triangleq \gamma\left(\frac{t}{a}\right). \tag{2.19}$$

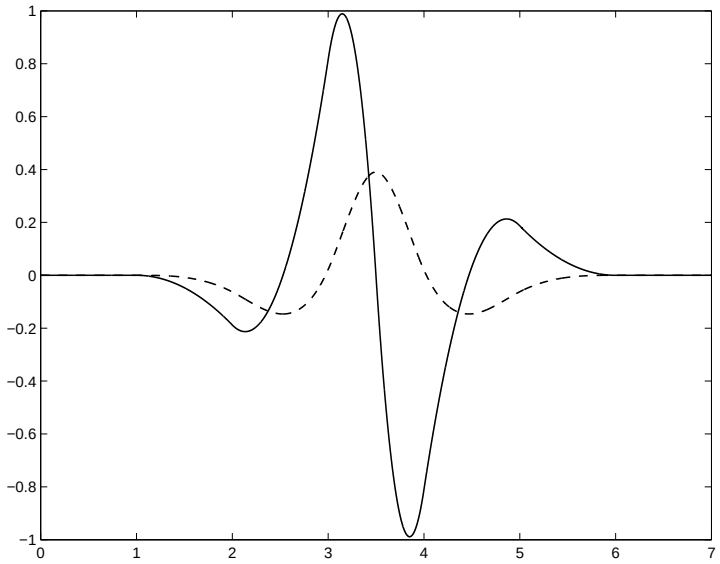

Figura 2.5: Le funzioni ψ (a tratto continuo) e γ (tratteggiata)

La (2.17) diventa quindi

$$C_{a,b} = \frac{1}{\sqrt{a}} \sum_{k\in\mathbb{Z}} s(k) \left[a \cdot \gamma_a(k+1-b) - a \cdot \gamma_a(k-b)\right] \tag{2.20}$$

$$= \sqrt{a} \left[\sum_{k\in\mathbb{Z}} s(k)\gamma_a(k+1-b) - \sum_{k\in\mathbb{Z}} s(k)\gamma_a(k-b)\right]. \tag{2.21}$$

Poiché partendo da un segnale campionato vogliamo ottenere, fissata la scala a, la sua trasformata wavelet negli stessi istanti, b assumerà solo valori interi. γ_a può essere perciò vista come una successione che dipende solo dalla funzione wavelet scelta e dalla scala a. Quindi, essendo tanto s che γ_a successioni, è possibile riconoscere in (2.21) l'operatore *correlazione discreta* che è così definito

$$R_{x,y}(i) \stackrel{\triangle}{=} \sum_{j\in\mathbb{Z}} x(j+i)y(j) = \sum_{j\in\mathbb{Z}} x(j)y(j-i). \tag{2.22}$$

L'equazione (2.21) può quindi essere scritta come

$$C_{a,b} = \sqrt{a}\left[R_{s,\gamma_a}(b-1) - R_{s,\gamma_a}(b)\right] \tag{2.23}$$

$$= -\sqrt{a}\left[R_{s,\gamma_a}(b) - R_{s,\gamma_a}(b-1)\right]. \tag{2.24}$$

La funzione madre ψ è sempre a media nulla e, per molte famiglie wavelet, a supporto compatto (per tutte comunque tende rapidamente a zero fuori da

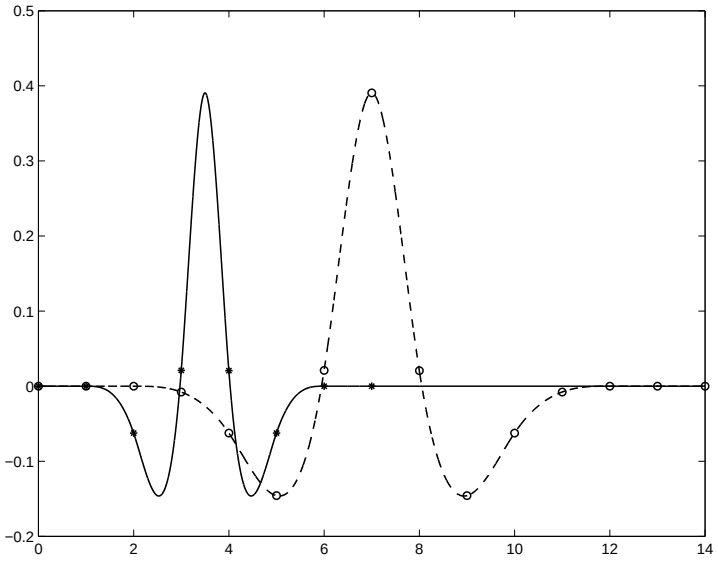

Figura 2.6: La funzione $\gamma(t)$ alle scale 1 (a tratto continuo) e 2 (tratteggiata). Le rispettive successioni $\gamma_a(m)$ per $a = 1$ (stelline) e $a = 2$ (cerchietti).

un intervallo). Questo significa che anche γ e γ_a sono a supporto compatto (o al limite trascurabili al di fuori di un intervallo). Se ψ ha supporto $[0, c]$, γ_a è composta da circa $c \cdot a$ elementi (figura 2.6). Se il segnale s è una sequenza finita di lunghezza m, la trasformata wavelet calcolata ha lunghezza circa[6] $m + c \cdot a$ ma se vogliamo che il segnale trasformato abbia la la stessa lunghezza di quello originario possiamo trascurare le code laterali conservando solo gli m campioni centrali. Questo è l'algoritmo utilizzato dalla funzione cwt del *wavelet toolbox* di MATLAB® (ved. [18]) e che è stato ripreso per implementare la classe CWavelet descritta al paragrafo A.2.1. Calcolare la CWT di un segnale di lunghezza m mantenendo solo gli m campioni centrali del segnale trasformato equivale a calcolare la CWT per $\bar{b} \in [0,\, m-1]$ dove $\bar{b}$ è la traslazione temporale quando la funzione ψ ha supposto simmetrico rispetto all'origine.

Per effettuare la *correlazione* si può seguire la procedura indicata dalla definizione (2.22) oppure lavorare nel dominio trasformato di Fourier. Questo secondo metodo sfrutta il fatto che la *convoluzione* e la *correlazione* diventano, nel dominio trasformato, semplici moltiplicazioni elemento per elemento. Si può pertanto calcolare la FFT (*Fast Fourier Transform*) dei due segnali,

[6] La correlazione di due sequenze di lunghezza m e n ha lunghezza $m + n - 1$

effettuare la moltiplicazione e antitrasformare il risultato tramite IFFT. Questo metodo conviene decisamente nel caso si lavori con successioni piuttosto lunghe e si voglia calcolare in blocco l'intera sequenza di uscita. Nel nostro caso il segnale è formato da 128 campioni (ved. par. 5.1) e, inoltre, talvolta è sufficiente conoscere coefficienti sparsi e non $C_{a,b}$ per ogni b. In queste condizioni è più vantaggioso lavorare nel dominio temporale utilizzando la definizione di correlazione.

Riassumendo, l'algoritmo per calcolare gli m coefficienti centrali della trasformata wavelet alla scala a di una successione s di lunghezza m è composto dai seguenti passi:

1. calcolare o importare (precedentemente calcolata) $\gamma(t)$;
2. creare da $\gamma(t)$ la successione $\gamma_a(m)$;
3. calcolare la correlazione per $m + 1$ istanti;
4. effettuare la differenza tra ogni campione della correlazione ed il successivo e moltiplicare per $-\sqrt{a}$.

Non tutte le operazioni devono essere eseguite ogni volta. In particolare il primo passo può essere eseguito una volta per tutte prima di effettuare qualsiasi trasformata. Anche il secondo passo può essere economizzato creando una sorta di *cache* che consenta di riciclare una γ_a calcolata precedentemente per la solita scala. L'implementazione software dell'algoritmo presentato è descritta al paragrafo A.2.1.

Capitolo 3

Algoritmi genetici

3.1 Caratteristiche generali

Gli algoritmi genetici sono stati introdotti nel 1975 da John Holland. Sono un modello computazionale che si ispira all'evoluzione naturale di Charles Darwin. Appartengono alla categoria degli algoritmi evoluzionistici assieme a strategie evoluzionistiche, programmazione evoluzionistica e programmazione genetica.

Vengono utilizzati in problemi di ottimizzazione di parametri, per la progettazione automatica o per lo *scheduling* di operazioni.

Una popolazione di possibili soluzioni si evolve seguendo *regole di selezione* ed operatori di ricerca: *crossover* e *mutazione*. Ogni individuo riceve una misura delle proprie prestazioni nel risolvere il problema dato. Tale misura ne determina la probabilità di ricombinarsi ovvero la probabilità che il proprio patrimonio genetico si propaghi nella popolazione. Con il procedere delle generazioni si ha pertanto una maggiore probabilità che gli individui presentino caratteristiche adatte alla soluzione del problema.

I passi tipici con cui un algoritmo genetico si evolve possono essere riassunti nei seguenti:

1. Inizializzazione della popolazione (di solito con individui generati casualmente)

2. Calcolo delle prestazioni (*fitness*) di ciascun individuo della popolazione

3. Ripetizione per un numero predefinito di generazioni o finché un determinato obiettivo non è raggiunto di:

(a) Selezione, in base alla fitness, degli individui candidati ad essere genitori

(b) Con una determinata probabilità, ricombinazione (crossover) degli individui scelti; altrimenti copia nella generazione successiva.

(c) Con una determinata probabilità mutazione dei nuovi individui

(d) Calcolo della nuova fitness

3.2 Confronto con altri algoritmi di ricerca

Altri metodi di ricerca sono quelli basati sul gradiente, quelli enumerativi e quelli casuali.

I metodi di ricerca basati sul gradiente utilizzano appunto il gradiente della funzione obiettivo al fine di indirizzare la direzione in cui la ricerca deve procedere.

I metodi enumerativi campionano uno a uno i punti dello spazio di ricerca al fine di determinare il massimo della funzione obiettivo.

La ricerca casuale (*random search*) campiona casualmente lo spazio di ricerca salvando il punto con funzione obiettivo massima incontrata fino a quel momento.

Questi ultimi due metodi sono decisamente inefficienti quando si ha a che fare con spazi di ricerca molto vasti. I metodi basati sul gradiente sono inadeguati per spazi di ricerca rumorosi ed inoltre non sono applicabili nel caso non sia possibile calcolare il gradiente.

3.3 Impieghi degli algoritmi genetici

Gli algoritmi genetici sono stati applicati con successo in numerosi campi della ricerca e dell'industria come ad esempio:

- ottimizzazione e studio dell'aerodinamica di aeroplani;
- progetto del propulsore del Boeing 777;
- programmazione delle operazioni di produzione e distribuzione di ossigeno liquido e azoto a 10000 destinatari;

- progettazione di antenne;
- progettazione di circuiti logici;
- progettazione di circuiti analogici;
- progettazione di FPGA;
- ottimizzazione della rete di distribuzione idrica;
- analisi finanziaria e previsioni di mercato;
- ricerca della struttura di molecole di proteine;
- trattamento e classificazione di segnali bioelettrici;
- segmentazione di immagini;
- ottimizzazione della traiettoria di robot;
- progettazione di reti neurali (struttura e/o pesi);
- analisi e previsione di sistemi dinamici non lineari;
- ottimizzazione del routing di pacchetti nella rete.

3.4 La popolazione e l'individuo

Gli algoritmi genetici cercano la soluzione migliore ad un problema evolvendo, con regole e meccanismi ispirati all'evoluzione naturale, una popolazione di possibili soluzioni. Tale popolazione viene inizialmente creata in modo casuale e quindi, in generale, i suoi individui risolveranno il problema in modo piuttosto deludente.

La popolazione viene evoluta per un certo numero di generazioni e i meccanismi di selezione, riproduzione e mutazione, fanno sì che gli individui migliorino le loro prestazioni di generazione in generazione. Poiché tali meccanismi sono regolati da fenomeni casuali, l'esatta evoluzione delle prestazioni della popolazione nel corso delle generazioni è imprevedibile. Ciononostante, il fatto che gli individui con prestazioni migliori abbiano maggiori probabilità di riprodursi, genera una tendenza della popolazione a migliorare ovvero a risolvere meglio il problema dato.

Ogni individuo rappresenta, quindi, una possibile soluzione del problema. In caso il problema sia l'ottimizzazione di parametri, ogni soluzione è rappresentata, determinata proprio da tali parametri detti *parametri oggetto*. Questi costituiscono il *fenotipo*. È possibile che il crossover non lavori direttamente sul fenotipo ma su una sua rappresentazione tramite stringa binaria che ne costituisce il *genotipo*.

Ad esempio, parametri interi possono essere rappresentati con la loro codifica binaria. Parametri reali possono essere convertiti, dividendone la dinamica in un numero finito di punti, in interi per poi prenderne la codifica binaria.

Utilizzando questo approccio, l'algoritmo genetico evolve indirettamente una soluzione, rappresentata dal fenotipo, manipolandone la sua rappresentazione tramite genotipo. Poiché spesso i parametri della soluzione (il fenotipo) sono soggetti a restrizioni è necessario verificare che il genotipo generato dalle operazioni di crossover e mutazione rappresenti un individuo valido e quindi una soluzione valida al problema.

Ad esempio supponiamo di voler trovare il massimo rapporto tra i seni dei due numeri usciti dal lancio di due dadi. La popolazione sarà costituita da individui determinati ciascuno da una coppia di numeri da 1 a 6. Ciascuno di questi numeri è un parametro oggetto mentre insieme costituiscono il fenotipo dell'individuo. Con la rappresentazione binaria il genotipo di ciascun individuo potrebbe essere costituito dalla codifica binaria a 3 bit dei due parametri. Cosa succede se in seguito a crossover o mutazione del genotipo la rappresentazione binaria di un parametro assume il valore 111 oppure 000? Il genotipo rappresenta un fenotipo non valido e pertanto non può essere immesso nella popolazione. In tali circostanze si può scegliere di eliminare l'individuo oppure di convertire un valore non valido in uno valido. Nel primo caso l'algoritmo può rallentare in modo inaccettabile qualora il numero di configurazioni invalide non sia trascurabile rispetto al numero di quelle valide. Nel secondo caso è necessario fare attenzione a non creare combinazioni "favorite" che potrebbero rendere difficile la ricerca, come avverrebbe nell'esempio precedente correggendo il valore n con $1 + (n \bmod 6)$ quando non è compreso fra 1 e 6; così facendo i valori 1 e 2 sarebbero più probabili degli altri.

La codifica del fenotipo nel suo genotipo è, quindi, di estrema importanza.

Una codifica ideale possiede le seguenti caratteristiche:

- deve rappresentare tutte le possibili soluzioni del problema;
- deve codificare solo le soluzioni ammissibili del problema;
- deve rappresentare tutte le soluzioni con uguale probabilità;
- deve codificare in modo compatto le soluzioni;
- piccole variazioni del genotipo devono implicare piccole variazioni nel fenotipo.

Per rispettare maggiormente l'ultima proprietà, talvolta si fa ricorso al codice Gray per codificare valori interi. In questo modo, infatti, piccole variazioni del genotipo implicano variazioni del fenotipo minori che nel caso binario. Si ha però lo svantaggio che non piccole variazioni del genotipo portano a variazioni del fenotipo maggiori che nel caso binario.

Talvolta si può scegliere di non far ricorso ad una codifica del fenotipo nel genotipo. In tal caso fenotipo e genotipo coincidono e gli operatori di crossover e mutazione devono essere studiati affinché possano operare direttamente sul fenotipo. Il gene, ovvero l'unità minima che costituisce il cromosoma e su cui operano gli operatori di ricerca, assume in questo caso un significato diverso. Non è più un singolo bit della stringa binaria che rappresenta il genotipo ma è un intero parametro del fenotipo. Questo approccio è detto *rappresentazione naturale* ed è utilizzato in caso di *strategie evoluzionistiche* o *algoritmi genetici a codifica reale*.

Non sempre il problema da risolvere è l'ottimizzazione di una serie di parametri. Si può essere interessati nel determinare la sequenza migliore con cui svolgere una serie di operazioni, come nel problema del commesso viaggiatore dove si vuole trovare l'ordine con cui visitare un insieme di località al fine di percorrere il minor numero di chilometri.

Un'altra categoria di problemi è quella che prevede l'ottimizzazione di un progetto o della sua struttura, come nel caso in cui si utilizzino algoritmi genetici per determinare la struttura di una rete neurale artificiale. In tutti questi casi la rappresentazione degli individui non è banale e gli operatori di crossover e mutazione devono assere adattati al problema specifico.

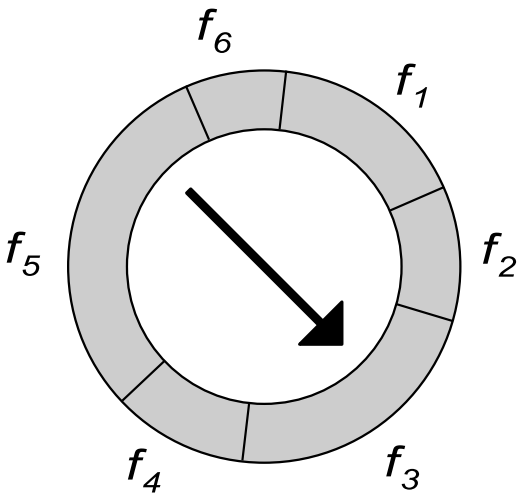

Figura 3.1: Rappresentazione grafica del metodo di selezione proporzionale alla fitness detto anche della roulette.

3.5 Fitness e strategie di selezione

La *fitness* è una misura delle prestazioni dell'individuo nel risolvere il problema assegnato. È una quantità sempre positiva e determina la probabilità dell'individuo di riprodursi: quanto maggiore è la sua fitness, tanto più alta è la probabilità che questo possa essere selezionato per il crossover con un altro individuo.

Talvolta, a seconda delle strategie di selezione, il ruolo della fitness può essere svolto dalla *objective function* (funzione obiettivo). Analogamente alla fitness, la funzione obiettivo è rappresentativa delle prestazioni di un individuo nel risolvere il problema. Si differenzia dalla fitness in quanto, durante la selezione, il valore della funzione obiettivo di un individuo non viene utilizzato se non per confrontare tra loro più individui. La objective function può essere indifferentemente minimizzata o massimizzata. Di un individuo non è importante la funzione obiettivo in sé ma in rapporto a quella degli altri. Addirittura con queste strategie di selezione (*rank selection* e *tournament selection*) sarebbe sufficiente definire un operatore per il confronto tra individui senza ricorrere a funzioni numeriche.

Selezione proporzionale alla fitness

È il metodo tradizionale con cui vengono selezionati gli individui per il crossover. Ogni individuo ha una probabilità di essere scelto proporzionale alla sua fitness:

$$p_i = \frac{f_i}{\sum_j f_j}.$$

Questo metodo è anche detto della roulette (fig. 3.1). Si estrae un numero casuale e si seleziona l'individuo puntato dalla freccia. Gli individui con fitness maggiore, avendo un settore di ampiezza maggiore, hanno maggior probabilità di essere estratti.

Questo metodo presenta alcuni difetti. Nel caso un individuo abbia fitness molto maggiore della media della popolazione ma molto minore della massima fitness possibile, questo verrà selezionato molto spesso ed il suo patrimonio genetico tenderà a diffondersi rapidamente generando una popolazione con una fitness mediocre. Un secondo problema si presenta poiché, dopo un certo numero di generazioni, tutti gli individui tendono ad avere fitness molto simile e quindi hanno probabilità molto simile di essere selezionati. Questo fa sì che la ricerca rallenti mano a mano che la popolazione si evolve.

Selezione per rango

La selezione per rango (*rank selection*) viene effettuata ordinando gli individui in base alle loro prestazioni (tramite la fitness o la funzione obiettivo). Viene poi assegnata a ciascun individuo una probabilità di essere selezionato in ragione della posizione che questo ricopre nella graduatoria. Ad esempio, utilizzando una distribuzione di probabilità proporzionale alla posizione occupata, si ha:

$$p_i = \frac{i}{\sum_j j} \text{ dopo aver ordinato gli individui in modo che } f_1 < \ldots < f_N.$$

Questo metodo supera i problemi visti nel caso precedente in quanto nessun individuo avrà probabilità di essere selezionato molto maggiore degli altri e non è possibile che tutti gli individui abbiano probabilità simile di essere selezionati. Il grosso svantaggio di questo metodo è che, essendo necessario un ordinamento, è computazionalmente pesante.

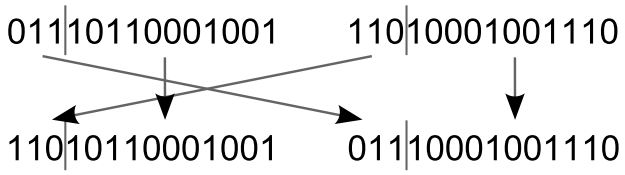

Figura 3.2: Esempio di crossover singolo-punto in caso di codifica binaria. In alto i genitori, in basso i figli.

Selezione per torneo

La selezione tramite torneo (*tournament selection*) avviene estraendo casualmente un ristretto numero di individui (poche unità) dalla popolazione e selezionando quello dalle prestazioni migliori (in termini di fitness o funzione obiettivo).

Anche questo metodo supera i problemi visti per la selezione proporzionale alla fitness ed ha il vantaggio, rispetto a quella per rango, di essere computazionalmente poco oneroso.

3.6 Crossover

Dopo che alcuni individui sono stati selezionati per formare il cosiddetto *mating pool*, nuovi individui vengono creati a partire dal patrimonio genetico dei genitori. Tramite crossover, ciascun nuovo individuo, come in natura, eredita parte del proprio codice genetico da un genitore e parte dall'altro.

A seconda di come è realizzato l'algoritmo genetico, da una coppia di genitori possono essere creati due figli o uno solo. È possibile scegliere di non utilizzare per il crossover tutti gli individui selezionati. In tal caso i rimanenti vengono importati invariati (a meno di mutazione) nella nuova generazione.

Crossover singolo-punto

Con il metodo del crossover singolo-punto, a partire dai geni contenuti nel cromosoma dei genitori si crea il cromosoma dei figli prendendo i primi k geni da un genitore ed i restanti dall'altro (fig. 3.2). Il parametro k è scelto casualmente di volta in volta.

Crossover doppio-punto

In questo caso, il cromosoma dei genitori viene spezzato in due punti ed i figli ereditano i geni interni da un genitore e gli esterni dall'altro. I due punti sono determinati casualmente di volta in volta.

Crossover uniforme

Con questo metodo, ciascun gene del cromosoma del figlio viene selezionato casualmente dall'uno o dall'altro dei due genitori.

Flat crossover

I precedenti tipi di crossover operano una selezione dei geni dai cromosomi dei genitori. I geni possono essere tanto bit di stringhe binarie che parametri di algoritmi genetici a codifica reale. In questo secondo caso, questi operatori riescono solo a scegliere alcuni parametri da un genitore ed i rimanenti dall'altro ma non possono cambiare i parametri stessi. Per questa ragione, in caso di algoritmi genetici a codifica reale, vengono solitamente utilizzati operatori di crossover di tipo diverso.

Uno di questi, il *flat crossover*, prevede che ciascun parametro del cromosoma del figlio venga scelto casualmente con probabilità uniforme nell'intervallo compreso tra i valori che lo stesso parametro assume nei due genitori. Il cromosoma dei genitori può essere espresso come $(a_1^1, \ldots, a_i^1, \ldots, a_n^1)$ e $(a_1^2, \ldots, a_i^2, \ldots, a_n^2)$, dove il pedice indica il coefficiente, l'apice il genitore. Il figlio avrà cromosoma $(a_1^f, \ldots, a_i^f, \ldots, a_n^f)$ dove a_i^f è scelto casualmente con probabilità uniforme in

$$\left[\min(a_i^1, a_i^2), \max(a_i^1, a_i^2)\right].$$

Blend crossover

Una variazione del flat crossover è il *blend crossover*, detto anche BLX-α. In questo caso il figlio avrà cromosoma $(a_1^f, \ldots, a_i^f, \ldots, a_n^f)$ dove a_i^f è scelto casualmente con probabilità uniforme in

$$\left[\min(a_i^1,\ a_i^2) - \alpha \cdot \Delta_i, \max(a_i^1, a_i^2) + \alpha \cdot \Delta_i\right] \text{ dove } \Delta_i = \left|a_i^1 - a_i^2\right|.$$

Questo significa che il parametro del figlio viene scelto casualmente nell'intervallo compreso tra i parametri dei genitori ma con la possibilità di sconfinare

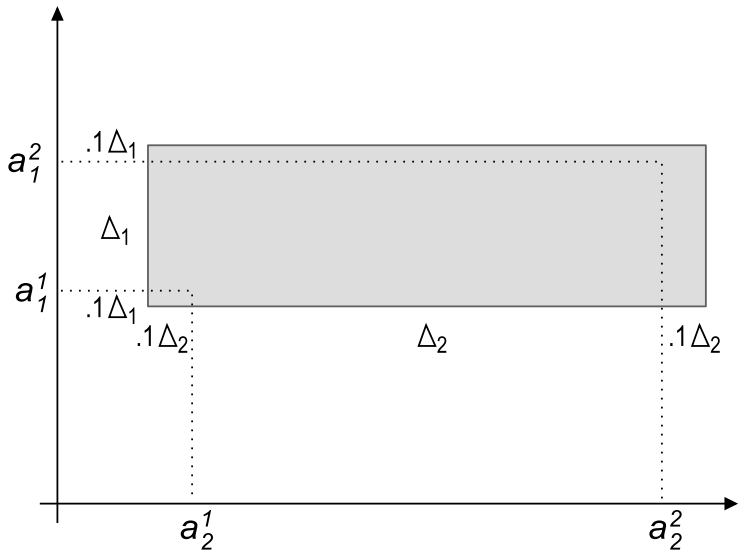

Figura 3.3: Esempio di blend crossover (in questo caso BLX-.1) per un cromosoma costituito da due coefficienti. Il pedice indica il coefficiente, l'apice il genitore. Dai due genitori, (a_1^1, a_2^1) e (a_1^2, a_2^2) può nascere un individuo con cromosoma (a_1^f, a_2^f) appartenente al rettangolo grigio.

da entrambi i lati di una frazione α. Questo implica che il parametro del figlio, pur essendo determinato da quello dei genitori, ha comunque la possibilità di esplorare fuori dai limiti da questi imposti, come mostrato nella figura 3.3.

3.7 Mutazione

La mutazione è l'operatore che agisce su un individuo introducendo una variazione nel suo cromosoma.

Inversione di un bit

Negli algoritmi genetici in cui il genotipo è rappresentato da una stringa binaria, la mutazione può essere realizzata invertendo il valore di un bit. Questo deve avvenire con una probabilità molto piccola (tipicamente dell'ordine di $0,001$) in quanto negli algoritmi genetici l'operatore di ricerca principale è il crossover. La mutazione serve invece ad introdurre delle variazioni nel pa-

trimonio genetico della popolazione ed a consentire che ciascun punto dello spazio di ricerca abbia una probabilità non nulla di essere campionato.

Sostituzione casuale di un parametro

È il corrispettivo del metodo precedente negli algoritmi genetici a codifica reale. Un parametro del cromosoma può essere sostituito con un valore scelto casualmente con probabilità uniforme nel dominio del parametro.

Variazione casuale di un parametro

Anche questo si applica ad algoritmi genetici a codifica reale. Un parametro del cromosoma viene variato di una quantità casuale scelta con una determinata distribuzione di probabilità.

Headless chicken

La mutazione *headless chicken*, letteralmente "pollo senza testa", viene effettuata in modo molto semplice. Viene applicato il crossover tra l'individuo che si intende mutare ed un individuo generato casualmente. A differenza dei metodi precedenti, le variazioni introdotte da questo operatore sono di una certa entità ed il nuovo individuo differisce fortemente da quello da cui è originato.

3.8 Variazioni

Elitismo

Consiste nell'effettuare una copia attraverso le generazioni dell'individuo migliore (o degli individui migliori). Infatti i meccanismi di selezione, avendo alla base processi casuali, non garantiscono che un individuo dalle buone prestazioni si propaghi alla generazione successiva. L'elitismo evita che una buona soluzione trovata vada perduta.

Dev'essere usato con attenzione perché c'è il rischio che le caratteristiche di tale individuo diventino dominanti nella popolazione e che l'algoritmo resti "incagliato" in un minimo locale.

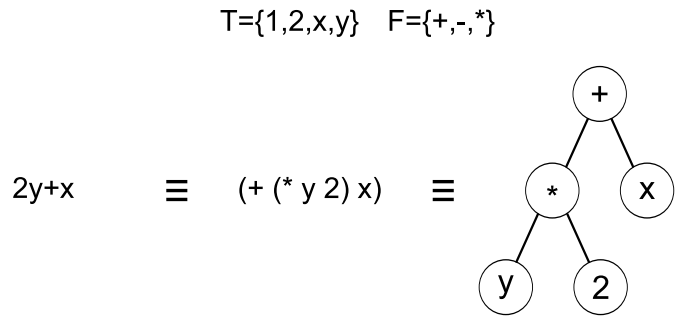

Figura 3.4: Programmazione genetica. Rappresentazione di un individuo tramite la sua s-expression e tramite albero.

Sottopopolazioni

Normalmente negli algoritmi genetici viene evoluta una sola popolazione e ogni coppia di individui al suo interno ha una probabilità di crossover che dipende esclusivamente dalla fitness dei due.

Talvolta le prestazioni dell'algoritmo genetico migliorano se all'interno della popolazione vengono evolute delle sottopopolazioni, ciascuna delle quali cerca in modo quasi indipendente la soluzione. Si tratta di favorire il crossover tra individui della stessa sottopopolazione e consentire, con una probabilità minore, l'incrocio tra individui di sottopopolazioni diverse. Questo può essere realizzato facendo dipendere la probabilità di crossover di ciascuna coppia, oltre che dalla loro fitness, dalla distanza tra i due individui. A tale scopo nello spazio delle soluzioni deve essere definita una distanza tra individui.

3.9 Programmazione genetica

Nel 1992 John Koza ha introdotto la programmazione genetica utilizzando gli algoritmi genetici per evolvere programmi in grado di svolgere un determinato compito. Ogni individuo è quindi un programma la cui fitness viene calcolata eseguendo il programma stesso. Tipicamente questi programmi sono scritti tramite le *s-espressioni* del linguaggio *Lisp*. Ultimamente però, al fine di ridurre i tempi di esecuzione, vengono impiegati altri linguaggi e si fa ricorso a metodi che consentano l'esecuzione in parallelo del codice.

Inizialmente devono essere dichiarati un insieme di simboli terminali (ingressi, costanti, ecc.) e un insieme di simboli di funzione (ad esempio operazioni matematiche, condizionali, ecc.). Ciascun individuo, oltre che dalla

sua espressione, può essere rappresentato in forma grafica tramite un albero i cui nodi rappresentano le funzioni e le foglie i simboli terminali (fig. 3.4).

Ovviamente la programmazione genetica necessita di operatori di crossover e mutazione diversi da quelli visti. I nuovi operatori devono essere in grado di operare sugli alberi generando soluzioni programmi corrette.

Capitolo 4

Paradigmi

4.1 Donchin Speller

Il Donchin Speller è un tipo di BCI che permette di immettere caratteri [10]. Fu presentato per la prima volta nel 1988 da Donchin e dal suo assistente Farwell [9].

Viene mostrata una matrice 6x6 di caratteri sullo schermo (fig. 4.1). I caratteri che costituiscono la matrice sono le 26 lettere dell'alfabeto inglese, le cifre da 1 a 9 e lo spazio. Ogni 125 ms una riga o una colonna nella matrice viene evidenziata per una durata di 100 ms. La sequenza con cui le righe e le colonne vengono illuminate è casuale ma tale che in ogni gruppo di 12, detto *trial*, ciascuna riga e ciascuna colonna vengano evidenziate una volta.

Viene istruito il soggetto affinché conti quante volte viene illuminato il carattere prescelto. Questo avverrà due volte ogni trial ovvero quando ad essere evidenziata è la riga contenente il carattere desiderato e quando lo è la colonna.

Il fatto che una riga o una colonna venga illuminata costituisce uno stimolo capace di generare una risposta di tipo ERP con una componente esogena dovuta all'elaborazione primaria dello stimolo visivo. Quando però ad essere evidenziata sarà la riga o la colonna cui la lettera scelta appartiene, oltre alla componente esogena sarà presente una marcata componente endogena. Questo perché il fatto che la lettera desiderata venga illuminata rappresenta un evento raro (con probabilità 1/6), inatteso (perché l'ordine con cui le righe e le colonne si illuminano è casuale) e *task relevant* ovvero significativo, importante ai fini del compito che il soggetto sta svolgendo, cioè contare le

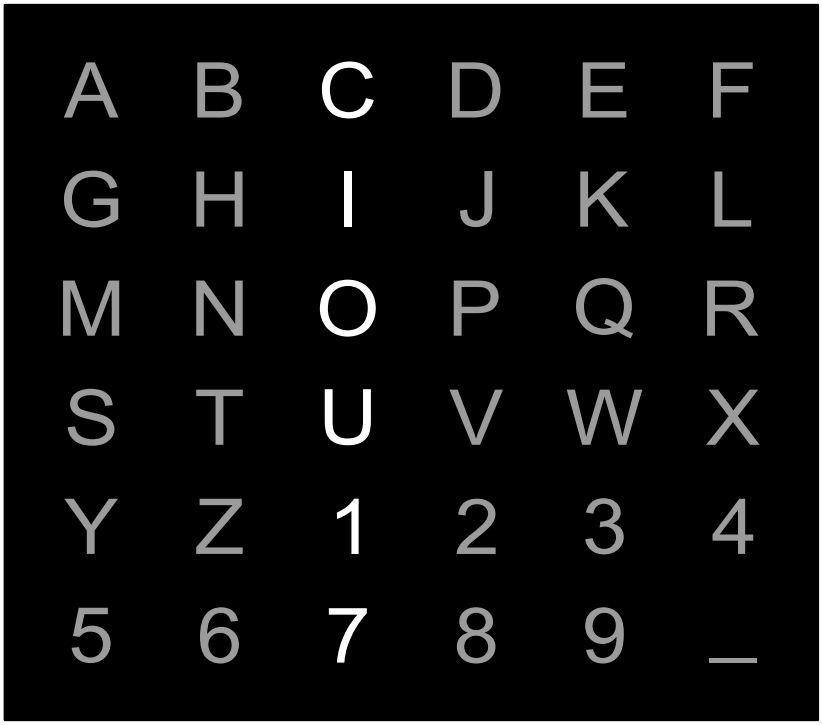

Figura 4.1: Matrice di caratteri utilizzata nel *Donchin speller* mentre è evidenziata la terza colonna.

volte che tale lettera viene illuminata. Questa è la condizione ideale affinché compaia un'onda di tipo P300, come descritto al paragrafo 1.2.5.

Essere in grado di rilevare la componente P300 significa poter risalire alla colonna ed alla riga a cui la lettera appartiene e quindi alla lettera stessa. Viene calcolata l'uscita del classificatore per ogni riga e si assume che la riga con uscita massima sia quella che contiene il carattere prescelto. Analogo procedimento viene effettuato per le colonne e dall'incrocio tra riga e colonna si risale al carattere che il soggetto vuole immettere.

Purtroppo non è facile realizzare un classificatore la cui attendibilità sia tale da poter risalire con un solo trial alla lettera prescelta. Si ricorre perciò a combinare i risultati di più trials al fine di abbattere la probabilità di errore. Si ottengono buoni risultati con 7-10 trials (si veda ad esempio [20]) il che significa circa quattro caratteri al minuto. Ovviamente realizzare un classificatore che riduca il numero di trials necessari per ottenere una percentuale d'errore accettabile, significa aumentare considerevolmente la velocità con cui il soggetto riesce a comunicare.

4.2 Mouse BCI

Nel realizzare quello che noi abbiamo chiamato *Mouse BCI* abbiamo ripreso l'idea del Donchin Speller ovvero di utilizzare il P300 per risalire, tra una serie di diversi stimoli, a quello su cui il soggetto sta focalizzando la sua attenzione.

Nel caso del Mouse BCI vengono mostrati quattro rettangoli scuri ai quattro lati dello schermo. Ciascun rettangolo corrisponde ad una delle quattro direzioni: su, giù, destra, sinistra. Ogni 180 ms viene presentato uno stimolo. Questo può essere costituito dal fatto che uno dei quattro rettangoli si colori di rosso per 100 ms oppure può essere uno stimolo "neutro" ovvero nessun rettangolo viene evidenziato (fig. 6.3 pag. 69).

All'interno di un gruppo di sei stimoli (trial) sono presenti, in ordine casuale, i quattro stimoli corrispondenti alle quattro direzioni e due stimoli neutri. Al fine di ridurre problemi legati ad una difficile percezione di due stimoli identici consecutivi (ved. [21]), il primo stimolo di un nuovo trial non può ripetere l'ultimo del trial precedente. I due stimoli neutri sono stati introdotti per due motivi. Il primo è avere la stessa probabilità che nel caso Donchin Speller che uno stimolo sia il target, ovvero 1/6. Il secondo è per evitare la ripetitività che si avrebbe nel caso che ogni 180 ms un rettangolo venisse comunque evidenziato.

Il soggetto deve contare quante volte viene evidenziato il rettangolo corrispondente alla direzione nella quale egli intende muovere il puntatore. Il periodo di tempo che inizia con la presentazione di uno stimolo e dura un secondo viene chiamato *epoca*. In fase di *training* le epoche vengono annotate con la direzione dello stimolo presentato e la direzione dello stimolo che il soggetto sta contando. Quando queste coincidono dovrebbe essere presente un segnale di tipo P300. Le epoche annotate vengono utilizzate per evolvere in modo supervisionato un classificatore.

In fase di utilizzo il classificatore precedentemente addestrato cerca di discriminare a quale dei quattro stimoli nelle quattro direzioni segua un segnale di tipo P300 o simile. Si assume che l'uscita analogica rappresenti il grado di fiducia con cui l'epoca può essere classificata come target. Perciò viene spostato il puntatore verso l'alto proporzionalmente alla differenza tra l'uscita del classificatore per l'epoca relativa all'accensione del rettangolo in

alto e quella relativa all'accensione del rettangolo in basso. Analogamente viene spostato verso destra proporzionalmente alla differenza tra l'uscita del classificatore per l'epoca relativa all'accensione del rettangolo a destra e quella relativa all'accensione del rettangolo a sinistra.

Capitolo 5

Classificatore

5.1 Preprocessing

5.1.1 Filtro FIR

Vengono considerati i 19 canali corrispondenti al sistema 10-20 presentato al paragrafo 1.2.3. Ciascuno di essi viene filtrato tramite un filtro passa-basso numerico FIR del trentesimo ordine. Questo significa che ogni uscita è calcolata come combinazione lineare dell'ingresso attuale e degli ultimi trenta campioni. I filtri FIR hanno il pregio di avere fase lineare introducendo un ritardo costante di $N/2$ campioni dove N è l'ordine del filtro. In realtà è possibile fingere di avere un filtro non causale con ritardo nullo, lavorando con il cosiddetto futuro relativo.

I coefficienti del filtro sono stati ottenuti con il *Filter Design Toolbox* di MATLAB®. L'algoritmo utilizzato è di tipo *Least square* ovvero tale da minimizzare l'errore quadratico tra il profilo in frequenza del filtro ideale e quello del filtro reale. Nel calcolo di tale errore è possibile specificare il peso dell'errore alle frequenze che si intende far passare ed il peso dell'errore alle frequenze che si intende bloccare. È stato impostato lo stesso valore per i due pesi, ovvero $W_{pass} = W_{stop} = 1$. La frequenza di campionamento è 256 Hz, la frequenza f_{pass}, cioè quella al di sotto della quale si vuole che il segnale passi il più possibile inalterato, è 34 Hz mentre la frequenza f_{stop}, ovvero quella oltre la quale si desidera che il segnale venga abbattuto, è 47 Hz.

Una volta filtrato, il segnale viene decimato per due prendendo un campione sì ed uno no. Questo è lecito in quanto il segnale decimato risulta campionato a 128 Hz e quindi più del doppio della massima frequenza pre-

sente dopo l'operazione di filtraggio (47 Hz). Grazie a ciò, essendo rispettato il teorema di Shannon, non si va incontro ad *aliasing*.

5.1.2 Preparazione delle potenziali features

Dopo che il segnale è stato filtrato e decimato, vengono preparate le potenziali *features* che l'algoritmo genetico potrà scegliere di utilizzare per realizzare il classificatore. Viene considerata *epoca* la finestra temporale della durata di un secondo che ha inizio con la presentazione dello stimolo. Per ciascuna epoca il classificatore dovrà indicare se è presente un segnale di tipo P300 relativo allo stimolo in questione. Essendo gli stimoli presentati con un tempo di ripetizione minore della durata di un'epoca, queste risulteranno parzialmente sovrapposte. Ciò rende ancora più arduo il compito del classificatore in quanto, sovrapposte alle componenti con latenza maggiore relative allo stimolo a cui l'epoca si riferisce, potranno esserci le componenti con latenza minore relative agli stimoli immediatamente successivi.

La classificazione avviene sula base della singola epoca. Dall'epoca in questione vengono estratte le features che sono costituite dai coefficienti della trasformata wavelet continua. Di ciascun canale elettroencefalografico viene calcolata la CWT per 30 scale comprese tra 2 e 40. Le scale scelte non sono distribuite uniformemente bensì con legge quadratica

$$a = \frac{1}{45}s^2 + \frac{2}{3}s + 2 \text{ con } s \in [0, 29]. \tag{5.1}$$

Questo perché con una distribuzione uniforme le pseudo-frequenze corrispondenti alle scale minori risultano molto distanti mentre quelle relative alle scale maggiori risultano molto ravvicinate. La distribuzione utilizzata tende a compensare questo effetto perché, con gli stessi valori estremi e a parità di punti intermedi, tende a ravvicinare le scale piccole ed allontanare quelle grandi.

Per quanto riguarda il tempo, solo i coefficienti che rientrano nella finestra temporale di pertinenza del P300 vengono conservati. Pertanto solo le traslazioni

$$\bar{b} = t + 30 \text{ con } t \in [0, 39] \tag{5.2}$$

vengono considerate, ovvero solo i coefficienti tra 30 e 69 del risultato della

funzione CWT descritta al paragrafo A.2.1. Tale intervallo corrisponde, a 128 Hz, alla finestra tra 235 e 540 ms.

Riassumendo, per ogni epoca da classificare ci sono 19 canali, di ciascuno di essi viene calcolata la trasformata CWT per 30 diverse scale e di ciascuna di queste vengono presi 40 istanti di tempo. Questo significa una rosa di $19 \cdot 30 \cdot 40$ potenziali features tra le quali l'algoritmo genetico deve scegliere le più adatte a discriminare la presenza o meno di una componente di tipo P300. Si può immaginare di rappresentare queste features tramite una matrice tridimensionale $\mathbf{V}(c, s, t)$, in cui il primo indice è il canale, il secondo determina la scala tramite la relazione (5.1) ed il terzo il tempo tramite la (5.2).

Un numero così elevato di *features* rappresenta indubbiamente una cospicua fonte di informazione ma anche un problema in termini di utilizzo delle stesse ai fini della classificazione. Si rivela quindi necessario uno stadio di selezione delle features al fine di estrarne un sottoinsieme ristretto costituito da quelle più utili e informative ai fini della classificazione.

Kohavi e John [22] hanno evidenziato come lo stadio di selezione delle features possa precedere quello di addestramento del classificatore, e in tal caso si parla di *filter approach*, oppure avvenire contestualmente, e in questo secondo caso si parla di *wrapper approach*. Abbiamo scelto questo secondo metodo lasciando all'algoritmo genetico il compito di ottimizzare tanto la selezione delle features, tramite gli indici della matrice $\mathbf{V}$, quanto il classificatore, tramite la scelta dei coefficienti del polinomio.

5.2 Fase di evoluzione del classificatore

5.2.1 Rappresentazione dell'individuo "polinomio"

Per classificare un'epoca vengono presi alcuni degli elementi della matrice di features $\mathbf{V}$ e combinati in un polinomio del tipo

$$P(\mathbf{V}) = a_0 + \sum_{h=1}^{N} a_h \prod_{k=1}^{M} \mathbf{V}(c_{h,k}, s_{h,k}, t_{h,k})^{e_{h,k}}. \tag{5.3}$$

L'uscita $P(\mathbf{V})$ viene poi costretta nell'intervallo $[-1, 1]$ tramite una funzione identità con saturazione. Quando tale valore è maggiore di una determinata soglia σ l'epoca viene classificata come *target* altrimenti come *non-target*.

Questi due termini stanno ad indicare se lo stimolo relativo all'epoca in esame sia o meno lo stimolo obiettivo sul quale il soggetto è concentrato, quello che il soggetto intende selezionare. Abbiamo scelto una rappresentazione tramite polinomio in quanto può, in via teorica, consentire operazioni sulle features che razionalmente potrebbero risultare utili. Tra queste la combinazione lineare di features, il rapporto tra due features, il quadrato di una feature (che oltretutto ha un effetto raddrizzante) e molte altre.

Fin dalle prime volte che abbiamo fatto evolvere l'algoritmo genetico alla ricerca di un classificatore è stato chiaro che i termini lineari giocano un ruolo fondamentale. Con un'impostazione come quella dell'equazione (5.3), per ottenere un termine lineare è necessario che per un determinato h, $M-1$ degli esponenti $e_{h,k}$ siano nulli ed uno solo sia 1. Ciò significa che per ottenere un termine lineare è necessario un certo sforzo da parte dell'algoritmo genetico. Per facilitare il compito all'algoritmo genetico abbiamo deciso di aggiungere una parte strettamente lineare al polinomio ottenendo l'espressione

$$P(\mathbf{V}) = a_0 + \sum_{j=1}^{L} a'_j \cdot \mathbf{V}(c'_j, s'_j, t'_j) + \sum_{h=1}^{N} a_h \prod_{k=1}^{M} \mathbf{V}(c_{h,k}, s_{h,k}, t_{h,k})^{e_{h,k}}. \quad (5.4)$$

Il compito dell'algoritmo genetico è quello di determinare in modo ottimale i parametri del polinomio, ovvero:

- gli $N+1$ elementi reali del vettore a_h;
- gli L elementi reali del vettore a'_j;
- gli L elementi interi di ciascuno dei vettori c'_j, s'_j e t'_j dove $c'_j \in [0, 18]$, $s'_j \in [0, 29]$ e $t'_j \in [0, 39]$;
- gli $N \cdot M$ elementi interi di ciascuna delle matrici $c_{h,k}$, $s_{h,k}$, $t_{h,k}$ e $e_{h,k}$ dove $c_{h,k} \in [0, 18]$, $s_{h,k} \in [0, 29]$, $t_{h,k} \in [0, 39]$ ed infine $e_{h,k} \in \{-3, -2, -1, 0, +1, +2, +3\}$.

È chiaro come, ottimizzando tutti questi parametri assieme, stiamo effettivamente svolgendo le operazioni di *features selection* e addestramento del classificatore congiuntamente.

Poiché ciascun polinomio è identificato dai parametri visti sopra, questi ne costituiscono il patrimonio genetico che deve poter essere memorizzato in

modo semplice ed efficiente. Il numero di termini lineari (L), il numero di monomi non lineari (N) ed il numero di variabili in ogni monomio (M) sono costanti. Si può pertanto creare un blocco di byte che contenga in sequenza tutti i parametri del polinomio nell'ordine con cui scriveremmo il polinomio per esteso. Ad esempio il seguente polinomio con due termini lineari e un termine non lineare composto da due variabili

$$+0.1 - 0.2 \cdot \mathbf{V}(3,4,5) - 0.6 \cdot \mathbf{V}(7,8,9) + 0.9 \cdot \mathbf{V}(7,5,3)^{-1} \cdot \mathbf{V}(8,6,4)^2$$

verrebbe codificato (tenendo conto che i coefficienti sono `float` mentre gli indici e gli esponenti `short int`)

0.1f	−0.2f	3	4	5	−0.6f	7	8	9	0.9f	7	5	3	−1	8	6	4	2
CDCCCC3D	CDCC4CBE	0300	0400	0500	9A9919BF	0700	0800	0900	6666663F	0700	0500	0300	FFFF	0800	0600	0400	0200

dove la riga inferiore rappresenta la codifica esadecimale del parametro e la riga superiore il corrispondente valore decimale.

5.2.2 Tipo di algoritmo genetico

Come visto sopra, l'intero polinomio è memorizzato in un'unica stringa. Non viene utilizzata la rappresentazione binaria bensì la *rappresentazione naturale* e ciascun gene del cromosoma rappresenta un parametro (par. 3.4). Si ha quindi un algoritmo genetico a codifica reale (*real coded genetic algorithm*).

Per ciascun parametro, e quindi gene, viene utilizzato un crossover di tipo *BLX-*α (par. 3.6) con α pari a 0.1. Questo significa che ogni parametro dell'individuo figlio sarà scelto casualmente nell'intervallo compreso tra i parametri dei genitori, con la possibilità di sconfinare da entrambi i lati del 10% dell'ampiezza dell'intervallo stesso. Quindi pur essendo il parametro del figlio determinato da quello dei genitori, c'è comunque la possibilità di esplorare fuori dai limiti da questi imposti, come mostrato nella figura 3.3 a pagina 43.

I genitori vengono scelti tramite tournament (par. 3.5) ovvero vengono estratti un certo numero di individui ed il migliore di questi viene scelto come genitore; con procedimento identico viene scelto l'altro genitore. La popolazione ha dimensione costante ed ogni nascituro ricopre la posizione precedentemente occupata da un altro individuo. L'individuo da rimpiazzare viene scelto tramite tournament negativo cioè vengono estratti un certo

numero di individui e viene scelto il peggiore in termini di objective function. La *tournament size*, ovvero il numero di individui che partecipano al tournament è 3.

La mutazione viene effettuata con un procedimento detto *headless chicken* (par. 3.7) ovvero viene incrociato l'individuo di partenza con un altro individuo generato casualmente.

La popolazione è costituita da 20000 individui.

5.2.3 Fitness binaria

Per il calcolo della fitness di ciascun individuo, o meglio – in questo caso – della objective function, sono utilizzati due metodi. Il primo ha come fine evolvere un classificatore che, sulla base di una singola epoca, sia in grado di indicare se questa sia *target* o *non-target*.

Vengono quindi considerati un egual numero di esempi per le due classi, target e non-target, e per ciascuno di essi viene valutato il valore del polinomio codificato nel cromosoma dell'individuo. Tale valore viene poi limitato all'intervallo $[-1, 1]$ tramite saturazione. Viene infine calcolata la media, tra i vari esempi, del quadrato dell'errore tra l'uscita limitata del classificatore e l'uscita desiderata, ovvero -1 in caso di non-target, $+1$ in caso di target. Poiché l'algoritmo genetico è realizzato in modo da minimizzare l'objective function, l'errore quadratico medio sopra descritto verrà minimizzato ovvero verrà cercato il classificatore con le prestazioni migliori.

5.2.4 Fitness endpoint

Il secondo metodo riguarda esclusivamente il *Mouse BCI*. Vengono raggruppati gli esempi in quartetti contenenti un'epoca per ciascuna delle quattro direzioni e viene valutato il polinomio per ognuna delle quattro epoche. Il valore del polinomio viene limitato all'intervallo $[-\frac{\pi}{2}, +\frac{\pi}{2}]$ tramite la funzione arcotangente.

Viene poi calcolato lo spostamento nella direzione desiderata come differenza tra l'uscita relativa allo stimolo in quella direzione e quella relativa allo stimolo nella direzione opposta. Lo spostamento in direzione ortogonale è invece dato dalla differenza tra le uscite nelle due direzioni ortogonali a quella desiderata.

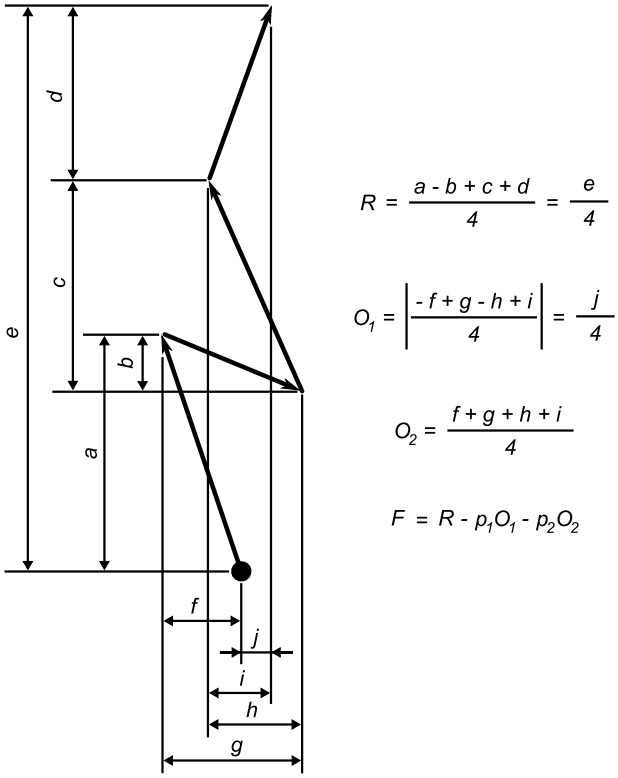

Figura 5.1: Calcolo della *Fitness endpoint* per quattro trials se la direzione desiderata è *su*. La fitness F è formata da un parametro di merito R e due penali O_1 e O_2. Il parametro di merito è la distanza media nella direzione desiderata. La prima penale è la deviazione finale media cioè la distanza finale nella direzione ortogonale divisa per il numero di passi (e quindi il numero di trials). La seconda penale è la deviazione media ad ogni passo. (Le lettere $a \ldots j$ sono distanze e quindi valori positivi)

L'objective function del polinomio è data da un fattore di merito e due penali (fig. 5.1). Il fattore di merito è lo spostamento medio nella direzione desiderata. La prima penale è lo spostamento medio in direzione ortogonale e fa sì che il punto finale cui si giunge non abbia un'eccessiva deviazione ortogonale. La seconda penale è la media dei valori assoluti dello spostamento in direzione ortogonale e penalizza le deviazioni ortogonali ad ogni passo, imponendo che, oltre che arrivare nel punto giusto, ci si arrivi andando il più dritto possibile. La realizzazione software del calcolo dell'objective function è presentata al paragrafo A.3.1.

Capitolo 6

Risultati

6.1 BCI competition

Abbiamo provato ad applicare il metodo descritto nel capitolo 5 al dataset IIb [26] della *BCI competition 2003*. Questo dataset è stato registrato utilizzando il paradigma di Donchin introdotto al paragrafo 4.1. Sono stati registrati i tracciati EEG di un singolo soggetto in tre diverse sessioni. Ogni sessione è costituita da cinque o sei *run* e in ciascuno di questi il soggetto ha focalizzato la sua attenzione su alcuni caratteri in successione al fine di formare una breve parola. L'immissione di ciascun carattere è formata da 15 trials ovvero gruppi di dodici intensificazioni di una riga o colonna. Al soggetto è stata lasciata la libertà di decidere se fissare un punto fisso oppure spostare lo sguardo sul carattere da immettere.

Abbiamo utilizzato soltanto la prima sessione che è composta da cinque run, per un totale di 19 caratteri e quindi $19 \cdot 15 \cdot 12$ intensificazioni. Ciascuna di esse individua un'epoca e quindi un esempio su cui addestrare o testare il classificatore.

Per addestrare il classificatore e testarne le capacità di generalizzazione, è stata impiegata una procedura di tipo *5-fold cross-validation* basata sui cinque run disponibili. Ogni volta sono stati utilizzati, a turno, quattro dei cinque run per addestrare il classificatore ed il rimanente per valutarne le prestazioni. I risultati ottenuti sono quindi la media di quelli conseguiti su ciascun run con il classificatore addestrato sugli altri quattro.

Il classificatore binario deve essere addestrato su un ugual numero di esempi target e non-target mentre nel dataset il rapporto tra queste due

classi è 1 a 5 (delle sei righe/colonne una è quella scelta mentre le altre cinque non lo sono). Per questo ciascun training set è stato formato con tutti gli esempi target ed un ugual numero di non-target scelti casualmente tra i disponibili.

6.1.1 Risultati

Prima di descrivere i risultati quantitativi c'è da sottolineare un aspetto qualitativo che emerge chiaramente dall'addestramento dei classificatori. L'algoritmo genetico sceglie esclusivamente di impiegare termini lineari utilizzando anche la parte non lineare in modo lineare, ponendo in ogni monomio un esponente a uno e tutti gli altri a zero. Questo può significare essenzialmente due cose. La prima è che un classificatore lineare sia effettivamente più adatto allo scopo. La seconda è che il modo con cui sono rappresentati i termini non lineari, tramite polinomio, non sia il più adatto. Infatti termini quadratici, cubici o frazioni possono condurre a valori numerici molto elevati che necessitano, per essere compensati, di coefficienti molto piccoli e quindi difficili da trovare per l'algoritmo genetico. Sarebbe interessante un'indagine accurata al fine di determinare quale delle due spiegazioni sia quella corretta.

Per valutare le prestazioni del classificatore sono stati calcolati sul *validation set* quattro parametri: percentuale di veri positivi, percentuale di falsi positivi, percentuale di risposte corrette e mutua capacità di canale.

Questi ultimi due valori tengono conto della diversa frequenza di epoche di tipo target e non-target nel Donchin speller. La percentuale di risposte corrette è stata calcolata come

$$Corr = \frac{1}{6} \cdot \mathrm{TP} + \frac{5}{6} \cdot (1 - \mathrm{FP}). \tag{6.1}$$

La mutua capacità di canale è un concetto attinto dalla teoria dell'informazione di Shannon. Dato un canale di comunicazione, la mutua capacità di canale è data dall'entropia del segnale in ingresso meno la cosiddetta equivocazione, ovvero l'entropia a posteriori dell'ingresso quando sia nota l'uscita. Possiamo applicare questo concetto al nostro caso considerando il classificatore come un canale binario con un ingresso S, costituito dallo stimolo e che può assumere i valori target e non-target, ed un'uscita R che è la risposta del classificatore e può assumere gli stessi valori target e non-target. In questo

		$N = 3$		$N = 4$	
		MaxCorr	MaxInfo	MaxCorr	MaxInfo
TP	mean	51.63%	65.61%	51.82%	70.58%
	std	6.89%	6.35%	5.58%	6.94%
FP	mean	6.00%	11.94%	5.22%	13.15%
	std	1.93%	4.64%	1.64%	3.99%
$Corr$	mean	86.94%	84.32%	87.62%	84.14%
	std	1.33%	2.99%	1.17%	2.45%
$I(S,R)$	mean	0.137	0.148	0.146	0.163
	std	0.026	0.016	0.023	0.018

Tabella 6.1: Prestazioni del classificatore binario sul dataset IIb tramite 5-fold cross-validation

caso la mutua capacità di canale può essere scritta come

$$I(S,R) = H(S) - H(S/R) = \sum_{s_i \in S} \sum_{r_j \in R} P(s_i, r_j) \lg_2 \frac{P(s_i, r_j)}{P(s_i)P(r_j)}. \qquad (6.2)$$

Ovviamente a parità di polinomio tutti questi valori (TP, FP, $Corr$, $I(S,R)$) risentono della soglia σ scelta. Alzare la soglia significa privilegiare la specificità a scapito della sensibilità mentre abbassarla ha l'effetto opposto. Sono stati utilizzati due metodi per determinare σ. Il primo, che abbiamo chiamato *MaxCorr*, consiste nello scegliere il valore della soglia che massimizza, sul training-set, la percentuale di risposte corrette. In questo modo si ottiene un classificatore più specifico che tende a sbagliare meno nel classificare epoche non-target in quanto queste sono più numerose. Con il secondo metodo, che abbiamo chiamato *MaxInfo*, viene scelto il valore di σ che massimizza la mutua capacità di canale sul training-set. In questo caso il classificatore tende ad essere più sensibile perché l'ingresso target, essendo il più raro, è quello che porta più informazione e quindi deve essere preservato il più possibile.

I risultati dei due approcci sono mostrati nella tabella 6.1. La percentuale di veri positivi, di falsi positivi, di risposte corrette e la mutua informazione di canale sono stati calcolati per ogni validation-set della *5-fold cross-validation*. In tabella ne sono mostrati i valori medi e la deviazione standard. Tali risultati sono suddivisi in base al numero di termini lineari, $N = 3$ e $N = 4$, e al criterio con cui è stata determinata la soglia, *MaxCorr* e *MaxInfo*.

6.1.2 Discussione dei risultati

Dai risultati mostrati in tabella si vede come il metodo proposto consenta di raggiungere una percentuale di classificazioni corrette fin oltre l'87%. È interessante osservare come, con l'approccio *MaxCorr* rispetto al *MaxInfo*, ad una maggiore percentuale di classificazioni corrette segua una minore informazione mutua di canale. Questo fa capire come talvolta la percentuale di risposte corrette possa trarre in inganno, specialmente nel caso di probabilità dei simboli in ingresso fortemente disomogenee.

Addestrando il classificatore con solo due termini lineari molto spesso si sono ottenute soluzioni del tipo

$$P(\mathbf{V}) = -0.335 + 0.100 \cdot \mathbf{V}(10, 15, 11) - 0.159 \cdot \mathbf{V}(16, 15, 11). \qquad (6.3)$$

Questo significa che quando la differenza pesata fra le correlazioni tra ciascuno dei canali C4 ($c = 10$) e T6 ($c = 16$) e la mother wavelet, traslata di circa 330 ms ($t = 11$) e dilatata di 17 volte ($s = 15$ ved. eq. (5.1)), è maggiore di $\sigma + 0.335$ l'epoca è classificata come target, altrimenti come non-target. Malgrado abbia due soli termini lineari questo classificatore ha, per $\sigma = 0$, una percentuale di veri positivi del 77% e di falsi positivi del 24% sul validation set.

Ma ciò che è ancora più interessante è che l'operazione sopra descritta può essere reinterpretata in modo più semplice. Grazie al fatto che tutte le operazioni svolte sono lineari (in particolare la correlazione che è alla base della CWT) può essere scambiato l'ordine con cui queste sono effettuate. La (6.3) può essere interpretata come la correlazione tra la mother wavelet traslata di circa 330 ms e dilatata di 17 volte e la differenza dei due canali C4 e T6. Questo ha dato spunto ad ulteriore indagine ed ha permesso di formulare una possibile ipotesi del modo con cui tale classificatore funzioni.

La figura 6.1 mostra il segnale registrato sui canali C4 e T6 in caso di epoca target e non-target. Al fine di ridurre il rumore, tali segnali sono la media su più epoche. È interessante notare come in caso di non-target il segnale sia simile tra i due canali e soprattutto in fase. Al contrario, in caso di target, il segnale risulta profondamente diverso. L'operazione scelta dall'algoritmo genetico, la differenza tra i due canali, tende quindi a cancellare il segnale in caso di non-target ed a rafforzarlo in caso di target,

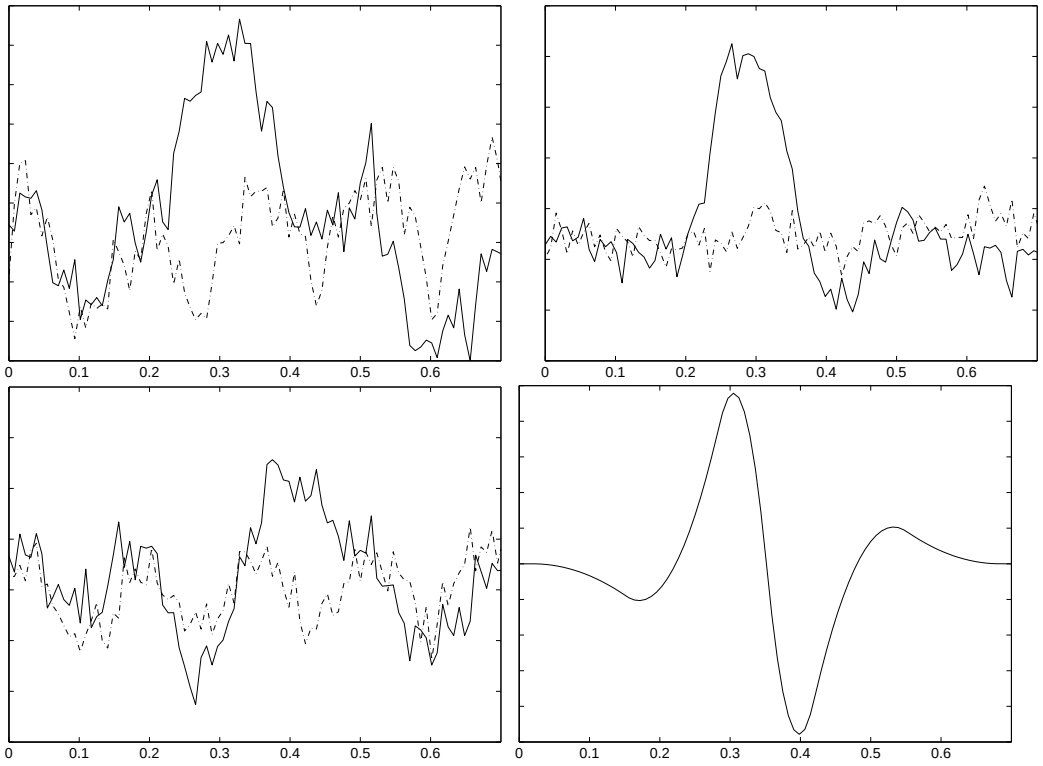

Figura 6.1: A sinistra la media delle epoche target (linea continua) e non-target (tratto-punto) del canale C4 (in alto) e T6 (in basso). A destra in alto la differenza pesata indicata dall'espressione (6.3) e, in basso, la wavelet rbio3.3 appropriatamente traslata e dilatata.

come mostrato sempre nella stessa figura. A questo punto la trasformata wavelet fa una sorta di *matching* tra il segnale ottenuto e la funzione wavelet e se la correlazione è maggiore di una certa soglia l'epoca è classificata come target.

6.1.3 Errori di percezione

Oltre agli errori imputabili al classificatore possono essercene altri, a monte, dovuti fenomeni percettivi. In neurofisiologia sono noti alcuni effetti dovuti alla presentazione di stimoli in rapida successione. Non è escluso che tali fenomeni possano verificarsi anche nel Donchin speller (ved. [21]). In particolare potrebbe accadere che l'intensificarsi di una riga o colonna vicino a quella che contiene il carattere da immettere provochi un'onda spuria, simile al P300, che può trarre in inganno il classificatore.

Abbiamo svolto un'ulteriore indagine che sembra avvalorare tale ipotesi. Sono state selezionate le epoche delle prime due sessioni del dataset IIb relative all'intensificazione della prima colonna. Queste sono successivamente state raggruppate in base alla colonna che, nello stesso trial, conteneva il carattere prescelto e ne è stata fatta la media. Ci sarebbe da aspettarsi che solo il segnale mediato sulle epoche relative alla prima colonna contenga un P300. La figura 6.2 mostra in rosso il segnale medio relativo alla prima colonna quando questa conteneva il carattere prescelto, in verde il segnale medio relativo sempre alla prima colonna ma quando era la seconda a contenere il carattere prescelto ed infine in blu è mostrato il segnale della prima colonna quando era la terza a contenere il carattere prescelto. Questi tracciati sono relativi al canale Cz che, non appartenendo alla corteccia visiva, non risente direttamente della direzione dello sguardo.

Il tratto interessante è quello attorno ai 300 ms dove tipicamente si presenta il P300. Si vede chiaramente che, quando il soggetto conta le intensificazioni di un carattere nella terza colonna, il segnale relativo alla prima non presenta onde di tipo P300. Quando invece vengono contate intensificazioni di un carattere sulla seconda colonna, il segnale relativo alla prima presenta una sorta di onda P300 attenuata. Questo può significare che un'onda simile al P300 si presenta frequentemente in forma attenuata ogni volta che si illumina una colonna adiacente a quella prescelta. Altra ipotesi è che l'onda P300 si presenti più saltuariamente nella forma classica ma risulti attenuata a causa dell'operazione di media.

Un'altra possibile fonte di errore potrebbe essere il fenomeno noto ai neuropsicologi come *repetition blindness* dovuto a stimoli target presentati a distanza di tempo ravvicinata. Quando due stimoli di tipo target rientrano in una finestra temporale di 500 ms può accadere che solo il primo venga correttamente riconosciuto (ved. [21]).

6.2 Mouse-BCI

Incoraggiati dai buoni risultati ottenuti sui dati della BCI competition abbiamo sviluppato un intero sistema in grado di presentare gli stimoli, acquisire i segnali, classificarli ed agire di conseguenza. Il modello con cui sono presenta-

ti gli stimoli è del tipo *Mouse BCI* introdotto al paragrafo 4.2. La figura 6.3 mostra come appare sullo schermo il pannello contenete gli stimoli.

6.2.1 Setup sperimentale

È stato impiegato un dispositivo di acquisizione per segnali elettroencefalografici modello Mindset24 realizzato dalla ditta Nolan Computer Systems. Tale dispositivo consente di acquisire fino a 24 canali in modalità differenziale o con un riferimento comune.

Ciascun canale presenta un'amplificazione di 90 dB e un SNR (rapporto segnale-rumore) di 64 dB. Tale segnale viene filtrato con due filtri passa-banda del quarto ordine con banda a 3 dB compresa tra 1.5 e 34 Hz. Successivamente viene campionato ad una frequenza selezionabile tra i 64 ed i 512 campioni al secondo e quantizzato a 16 bit. Il segnale così acquisito viene trasmesso ad un personal computer tramite interfaccia SCSI.

Per semplificare l'operazione di posizionamento degli elettrodi è stata impiegata una cuffia della Electrocap International con elettrodi già predisposti secondo lo standard 10-20 descritto al paragrafo 1.2.3.

Per l'esperimento è stato impiegato un solo soggetto, maschio, dell'età di 43 anni. L'esperimento è stato effettuato nel primo pomeriggio. Il soggetto era seduto su una poltrona comoda a circa un metro dallo schermo. Le impedenze tra i canali erano comprese tra i 2 e i 5 kΩ.

Sono stati presentate sequenze di stimoli come descritto al paragrafo 4.2. Ogni run del nostro esperimento era costituito da 30 trials ed il soggetto era tenuto a contare le 30 intensificazioni in una direzione predeterminata tra le quattro possibili. Sono stati acquisiti 12 run iniziando dalla direzione "su" e procedendo in senso orario. Il totale di 12 run è stato suddiviso in due gruppi: i primi quattro, i quattro centrali e gli ultimi quattro. È stato poi applicato un *3-fold cross-validation* per addestrare classificatori e verificarne le capacità di generalizzazione.

6.2.2 Risultati

Per addestrare il Mouse-BCI è stato utilizzato sia il metodo con fitness binaria (par. 5.2.3), sia quello con fitness endpoint (par. 5.2.4).

La tabella 6.2 mostra i risultati sul singolo trial nel caso di fitness binaria,

		$N = 4$		$N = 6$	
		MaxCorr	MaxInfo	MaxCorr	MaxInfo
TP	mean	40.43%	68.76%	51.61%	80.21%
	std	3.08%	14.81%	7.71%	3.60%
FP	mean	6.71%	22.63%	6.74%	25.33%
	std	1.67%	8.45%	1.76%	1.59%
$Corr$	mean	84.47%	75.93%	86.31%	75.59%
	std	0.88%	4.58%	0.27%	0.94%
$I(S,R)$	mean	0.081	0.097	0.128	0.130
	std	0.002	0.017	0.018	0.013
R	mean	0.544		0.907	
	std	0.064		0.103	
O_1	mean	0.057		0.072	
	std	0.037		0.033	
$\phi = \arctan(O_1/R)$	mean	5.70°		4.59°	
	std	3.03°		2.44°	

Tabella 6.2: Prestazioni del classificatore binario sui dati acquisiti con il MouseBCI. Risultati ottenuti tramite 3-fold cross-validation.

		$N = 4$	$N = 6$
R	mean	1.295	1.209
	std	0.370	0.271
O_1	mean	0.138	0.129
	std	0.092	0.070
$\phi = \arctan(O_1/R)$	mean	5.77°	6.40°
	std	3.02°	3.38°

Tabella 6.3: Prestazioni del classificatore endpoint sui dati acquisiti con il MouseBCI. Risultati ottenuti tramite 3-fold cross-validation.

ovvero gli stessi parametri mostrati per i dati della BCI competition: TP, FP, $Corr$, $I(S,R)$ calcolati sia con l'approccio MaxCorr che MaxInfo.

La tabella 6.3 mostra i risultati ottenuti con fitness endpoint. Sono mostrati il fattore di merito R (la distanza media percorsa nella direzione nella quale il soggetto desidera spostare il cursore), la penale O_1 (la deviazione media nella direzione ortogonale a quella desiderata) e infine l'angolo che la direzione effettiva forma con quella desiderata.

Questi tre parametri sono mostrati anche in tabella 6.2 per quanto riguarda i classificatori ottenuti con fitness binaria. In questo caso non intervengono in alcun modo nell'addestramento ma possono essere utili al fine di

raffrontare i risultati ottenuti con i due diversi tipi di fitness.

Tutti questi risultati sono stati ottenuti tramite elaborazione offline di dati acquisiti con il Mouse-BCI in modalità *training*, ovvero presentando gli stimoli senza alcun tipo di feedback per il soggetto. La figura 6.4 mostra un esempio dei risultati dell'addestramento tramite fitness binaria utilizzando i primi quattro e gli ultimi quattro run come training set ed i quattro centrali come validation set. Questa figura, come del resto i valori di ϕ nelle tabelle 6.2 e 6.3, mostra come la deviazione complessiva rispetto alla direzione desiderata sia piuttosto limitata (pochi gradi).

Utilizzando il Mouse-BCI in modalità *trace*, ovvero effettuando una classificazione online e tracciando il percorso su schermo, si ottengono risultati peggiori. Questo perché probabilmente il soggetto deve destinare parte della propria attenzione a controllare se la direzione riconosciuta dal classificatore sia quella da lui scelta. Questo feedback rappresenta quindi una causa di distrazione che degrada le prestazioni del sistema.

6.2.3 Uso del Mouse-BCI per immissione caratteri

Considerato che l'esperimento con il Mouse-BCI aveva dato risultati interessanti abbiamo pensato di realizzare un metodo di immissione di caratteri sfruttando il Mouse-BCI. Al centro dello schermo viene disegnato un cerchio suddiviso in 8 settori ciascuno dei quali contiene uno degli otto caratteri più diffusi della lingua inglese, come mostrato in figura 6.5.

Per immettere un carattere il soggetto deve portarsi nel settore corrispondente e possibilmente raggiungere la circonferenza. Il carattere relativo al settore in cui ci si trova viene acquisito non appena si verifica una di tre condizioni:

- quando il cursore raggiunge la circonferenza;
- quando, ordinando i settori in base al tempo trascorso in ciascuno di essi, la differenza tra quello in cui si è trascorso più tempo ed il successivo è netta;
- quando è trascorso un certo tempo limite.

Il carattere immesso viene accodato alla stringa attuale e mostrato in alto a sinistra.

I risultati di questo ulteriore esperimento sono stati buoni in quanto il soggetto è riuscito ad immettere correttamente la frase “she has”. L’obiettivo era in realtà “she has a hat” ma il soggetto ha preferito terminare in anticipo. Il processo è in effetti lungo e, richiedendo un’estrema concentrazione, piuttosto faticoso. Questo la dice lunga sulla strada ancora da compiere per ottenere un interfaccia cervello-computer che consenta, se non di comunicare alla velocità con cui si parla, almeno di farlo con una velocità che permetta, seppur in modo rallentato, una qualche forma di dialogo.

Ovviamente i risultati ottenuti non hanno grande valore scientifico in quanto relativi ad un’unica sessione, con un solo soggetto ed inoltre per la mancanza di un parametro di misura obiettivo con cui valutare il risultato ottenuto. Ciononostante è stato utile al fine di verificare la fattibilità della cosa e per individuarne i limiti e, di conseguenza, le direzioni in cui è necessario lavorare.

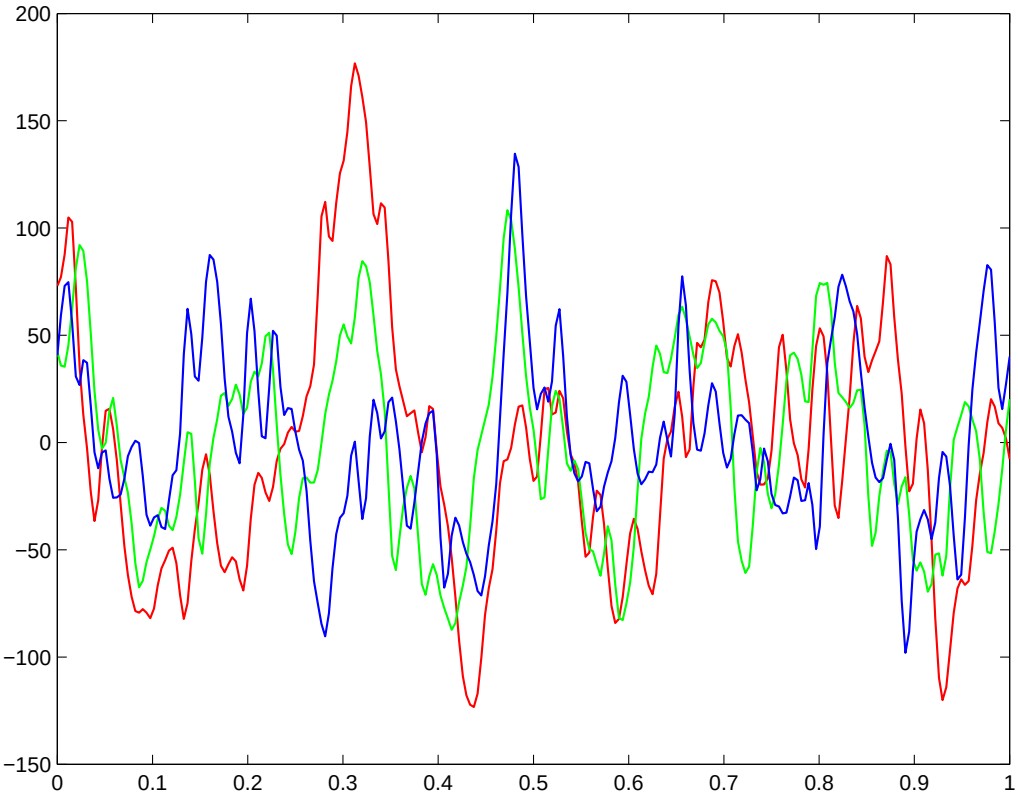

Figura 6.2: Segnale del canale Cz inerente alle epoche relative alla prima colonna quando il carattere prescelto si trovava nella prima colonna (rosso, media di 150 epoche), nella seconda (verde, media di 150 epoche) e nella terza (blu, media di 105 epoche).

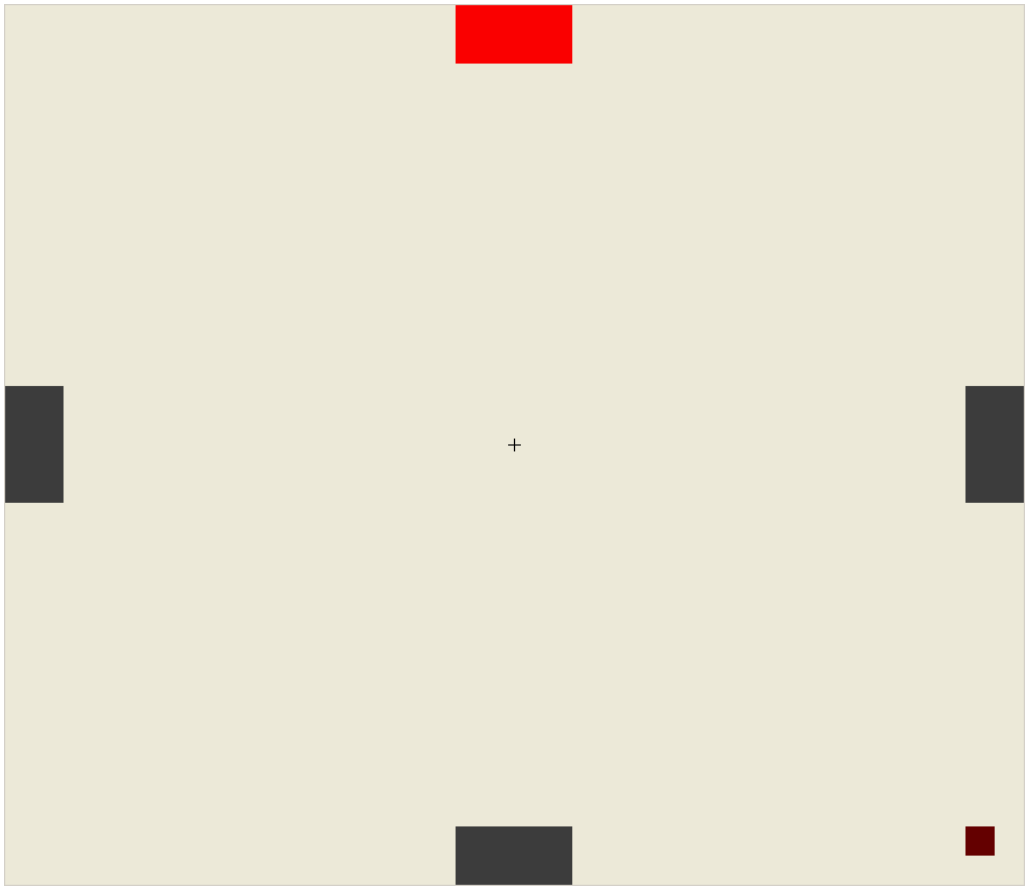

Figura 6.3: Mouse-BCI mentre viene presentato lo stimolo “su”.

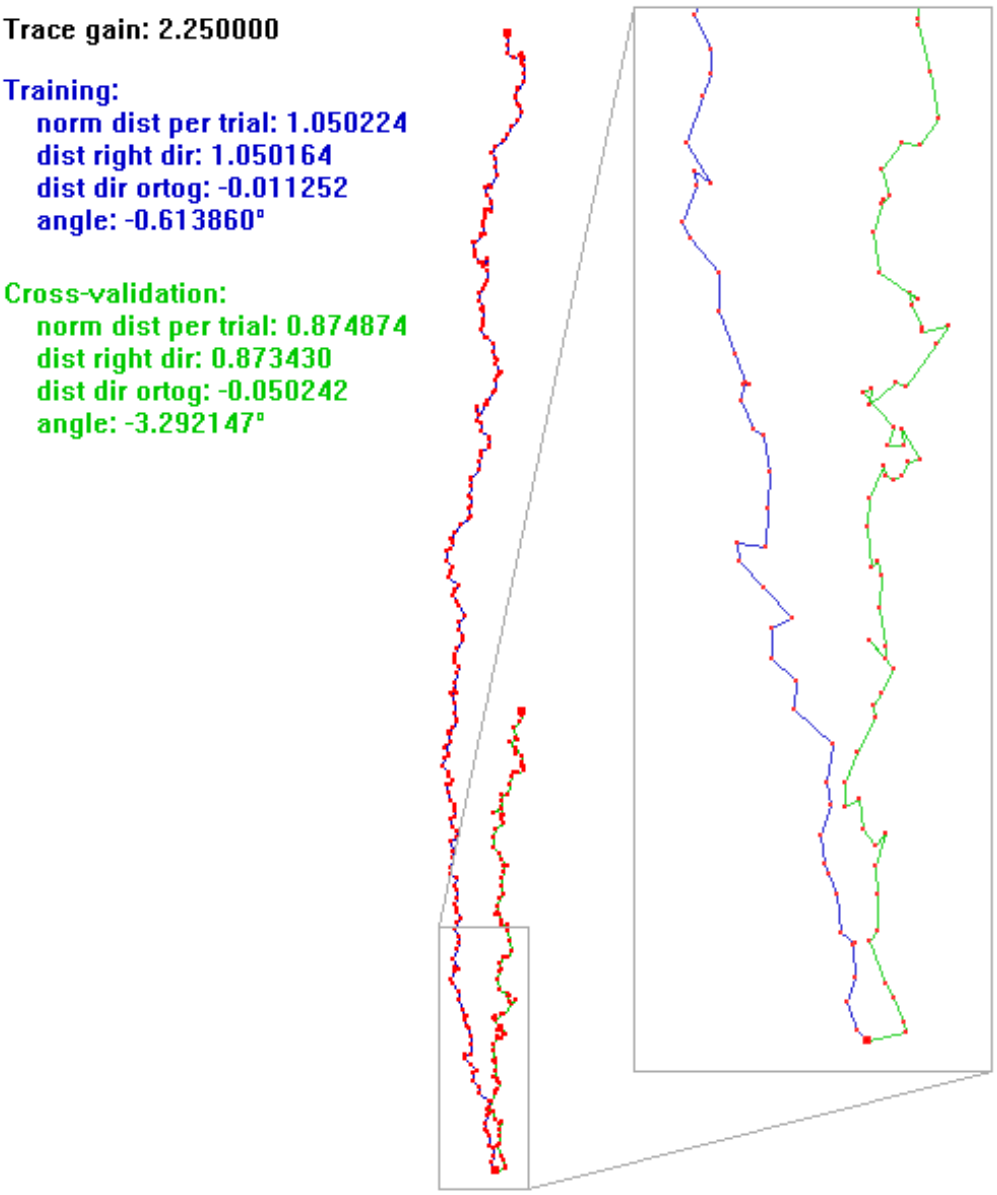

Figura 6.4: Tracce ottenute classificando offline il training set (run 1–4 e 9–12) ed il validation set (run 5–8). Il classificatore era stato evoluto tramite fitness binaria ma risultati del tutto analoghi si ottengono con fitness endpoint.

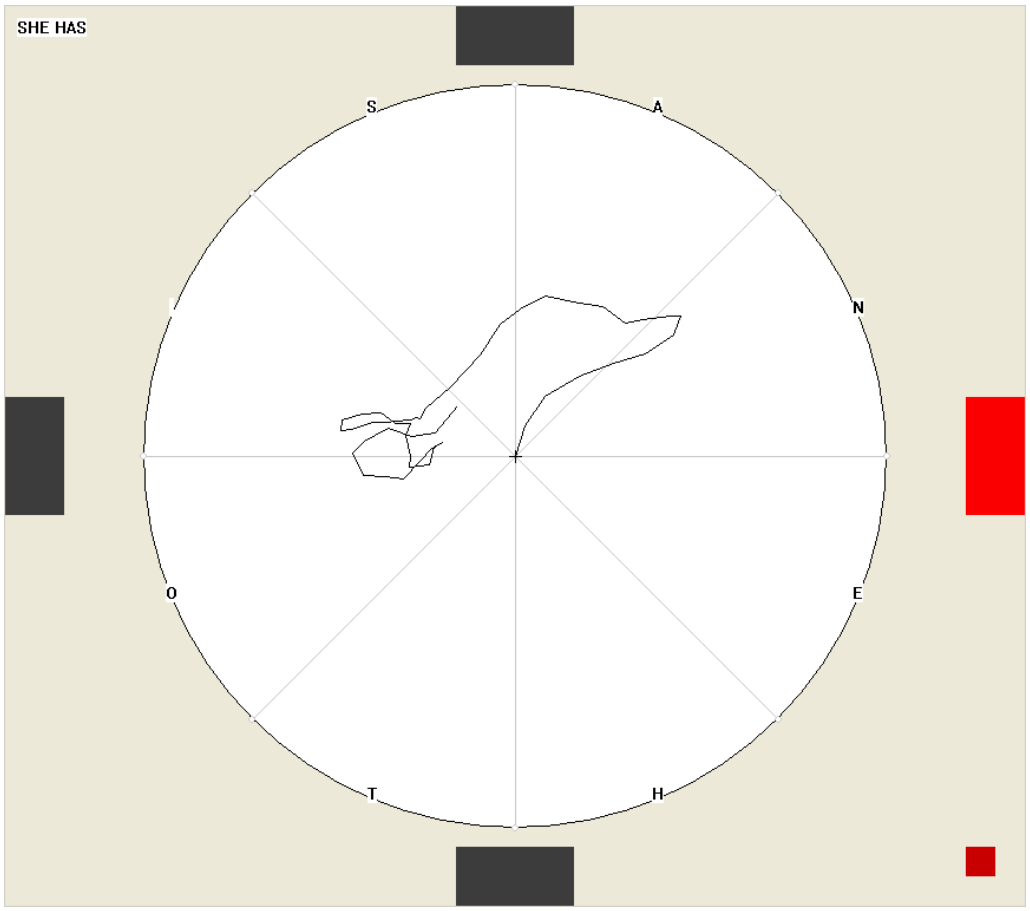

Figura 6.5: MouseSpeller-BCI dopo che il soggetto era riuscito ad immettere la frase “SHE HAS”.

Capitolo 7

Possibili evoluzioni

7.1 Classificatore

Sarebbe interessante riuscire a capire quale, delle due ipotesi fatte al paragrafo 6.1.1 riguardo alla preferenza del classificatore per i termini lineari, sia corretta. Per far ciò è necessario rivedere il modo con cui i termini non lineari sono rappresentati. Una soluzione potrebbe essere impiegare altre basi, come i polinomi di Bessel. Oppure si potrebbe prevedere una opportuna scalatura di ciascun coefficiente in relazione al grado del monomio a cui si riferisce così da facilitare il compito dell'algoritmo genetico. Un'altra soluzione potrebbe essere limitare ciascun monomio tramite una funzione limitatrice come l'arcotangente.

Infine potrebbero essere utilizzati classificatori di tipo completamente diverso. Ad esempio con un approccio *nearest neightbour*, si potrebbe utilizzare l'algoritmo genetico per definire i centri delle varie sottoclassi e scegliere come queste debbano essere unite per formare le classi di uscita.

Un altro possibile miglioramento potrebbe essere l'utilizzo di più famiglie wavelet tra le quali l'algoritmo genetico possa scegliere. Ciò potrebbe esser fatto aggiungendo una quarta dimensione alla matrice $\mathbf{V}$. Questo comporterebbe però dei rischi. In primo luogo per l'algoritmo genetico uno spazio di ricerca più ampio significa uno spazio più ampio da esplorare e quindi minori probabilità di trovare la soluzione ottima. Inoltre aggiungere una dimensione significa incrementare il numero di parametri da determinare (e quindi i gradi di libertà) aumentando così la probabilità di *overfitting* del training set qualora questo non contenga un numero sufficiente di esempi.

7.2 Donchin Speller

Applicare il classificatore ad ogni stimolo del Donchin speller indipendentemente dal fatto che questo rappresenti una riga o una colonna potrebbe non essere la scelta migliore. Abbiamo provato ad addestrare un classificatore affinché riconoscesse le epoche relative alle righe da quelle relative alle colonne. La percentuale di risposte corrette sul validation set è risultata ben maggiore del 50%[1] e questo significa che i due stimoli provocano una risposta diversa nell'individuo. Potrebbe pertanto essere una forzatura pretendere di evolvere un classificatore che abbia buone prestazioni in entrambi i casi. Probabilmente addestrare due distinti classificatori, uno per le righe ed uno per le colonne, potrebbe portare ad un miglioramento complessivo delle prestazioni.

In alternativa si potrebbe addestrare un classificatore che operi congiuntamente sulla riga e sulla colonna. Come spiegato in precedenza ogni trial è composto da 12 intensificazioni, sei righe e altrettante colonne. Accodando alle sei epoche di ogni riga ciascuna delle sei di ogni colonna si ottengono 36 "grandi epoche" ciascuna delle quali è associata ad un carattere. Il classificatore verrebbe poi addestrato a discriminare se ciascuna di esse rappresenti o meno il carattere prescelto. Delle 36 "grandi epoche" di ogni trial una sola dovrebbe provocare una risposta positiva da parte del classificatore e le altre 35 negativa.

Addirittura si potrebbe completamente rivedere il fatto che nel Donchin speller i caratteri vengono intensificati a gruppi di sei costituiti esclusivamente da righe o colonne. Sarebbe possibile intensificare gruppi di sei caratteri sparsi, valutare l'uscita del classificatore per quell'epoca ed incrementare di conseguenza il punteggio di ciascuno dei sei caratteri. Dopo un numero predefinito di intensificazioni o quando lo scarto tra il carattere con punteggio maggiore ed il successivo supera una determinata soglia, il carattere vincente verrebbe acquisito.

In questo modo si potrebbe imporre la condizione che un carattere non possa illuminarsi due volte nell'arco di 500 ms, evitando così il rischio di *repetition blindness* o fenomeni simili. Inoltre sarebbe possibile, nei trial

[1] Se le due classi sono equiprobabili, una percentuale di risposte corrette del 50% significa che il classificatore non riesce a distinguere le due classi.

successivi al primo, variare la probabilità di ciascun carattere di essere intensificato in relazione al punteggio ottenuto nei trial precedenti. Sarebbe così possibile concentrarsi prevalentemente sui caratteri con punteggio maggiore implementando una sorta di “play off”.

Appendice A

Realizzazione software

A.1 Presentazione stimoli e acquisizione

A.1.1 Classe CSCSI

La classe CSCSI funziona da wrapper nei confronti delle routine C di basso livello che si occupano della comunicazione con il dispositivo di acquisizione EEG. Tali routine sono state fornite dal costruttore e sono specifiche per il modello adottato. La classe CSCSI ha lo scopo di realizzare un'interfaccia di alto livello verso l'hardware, al fine di rendere lo sviluppo del resto dell'applicazione il più indipendente possibile dal particolare dispositivo di acquisizione adottato. Le operazioni fondamentali che la classe consente sono: inizializzazione del dispositivo, impostazione dei parametri e inizio/arresto del campionamento.

Il metodo SCSI_Init serve per inizializzare il dispositivo e restituisce l'esito di tale operazione. SCSI_Debug serve in fase di debug e consente di specificare una finestra nelle quale vengono mostrate informazioni sullo stato del dispositivo tra cui il numero di campioni trasferiti, il numero medio di chiamate al secondo e il numero di buffer underflow. Il metodo SCSI_SetSampleRate consente di impostare la frequenza di campionamento e le dimensioni di ogni blocco di campioni trasferito. Le frequenze di campionamento possibili sono 128, 256, 512 e 1024 Hz, ma è necessario tener presente che all'interno del dispositivo un filtro passa-basso analogico attenua fortemente le frequenze oltre i 38 Hz. La dimensione del blocco di campioni definisce la minima unità trasferibile tra il dispositivo ed il programma. Può variare tra 96 e 768 bytes e quindi il numero di campioni trasferiti per ogni canale può variare tra 2 e

16 (in quanto ci sono 24 canali ed ogni campione è 2 bytes). Impostare un valore basso significa avere un ritardo minore tra l'istante di campionamento e l'effettiva disponibilità del dato per il programma. Questo comporta, però, un maggior numero di chiamate al secondo e quindi un maggior impegno per il processore; ciò rende tale scelta giustificata solo per applicazioni real time. I metodi SCSI_StartSampling e SCSI_StopSampling consentono di avviare e sospendere il campionamento, mentre SCSI_IsSampling consente di sapere se il dispositivo stia campionando o meno. SCSI_GetCurrentSampleNum restituisce il numero progressivo del campione che viene campionato all'istante corrente. In concreto effettua la somma dei campioni trasferiti al programma, di quelli presenti nel buffer circolare implementato via software dalle routine C e di quelli presenti nel buffer circolare presente all'interno del dispositivo. È utile perché consente, nel momento in cui il programma presenta uno stimolo, di conoscere il numero progressivo del campione corrente e quindi di marcare via software i segnali acquisiti senza ricorrere ad altri espedienti.

Ci sono inoltre dei metodi virtuali che devono essere implementati nella classe derivata da CSCSI e che vengono chiamati in risposta ad eventi relativi al dispositivo di acquisizione. Affinché un metodo sia virtuale (talvolta chiamato anche astratto) deve essere definito tramite l'attributo virtual. Il metodo SCSI_OnDispatchData è sicuramente il più importante perché viene chiamato ogni volta che il dispositivo ha dei campioni disponibili. I parametri forniti a questo metodo sono: un puntatore ad un buffer contenete i dati, la dimensione in bytes e l'indicazione se questo sia il primo blocco trasmesso. SCSI_OnMessage viene chiamato ogni volta che il dispositivo incontra un errore. Fornisce il testo del messaggio stesso e, ad esempio, può essere implementato affinché mostri il messaggio sullo schermo oppure lo salvi in un file di log. SCSI_OnForcedAbort viene chiamato quando il dispositivo ha la necessità di sospendere l'acquisizione in seguito ad un errore (es. buffer overflow). Infine il metodo SCSI_OnShowBufferStatus viene chiamato quando è disponibile l'informazione sul livello del buffer interno al dispositivo e può essere utilizzato per mostrare un indicatore sullo schermo.

Per chiarire l'utilizzo di questi metodi e la ragione per cui sia necessario definirli virtuali, può essere d'aiuto un esempio concreto. Supponiamo di derivare da CSCSI la classe figlia CSignalAcq affinché acquisisca i dati e permetta di salvarli in un file. Tale classe erediterà i metodi del suo predecessore

```
class CSCSI
{
public:
  virtual ~CSCSI();
  BOOL SCSI_Debug(BOOL activate, HWND dbgwnd = 0);
  BOOL SCSI_Init();
  void SCSI_SetSampleRate(SAMPLE_RATE samplerate, BLOCK_SIZE blocksize = BLOCKSIZE_768);
  BOOL SCSI_StartSampling();
  BOOL SCSI_StopSampling();
  unsigned long SCSI_GetCurrentSampleNum();
  BOOL SCSI_IsSampling();

  // pure virtual functions: the derived class MUST implement them
  virtual void SCSI_OnDispatchData(BYTE *data, int num, bool first) = 0;
  virtual void SCSI_OnMessage(LPCSTR msg) = 0;
  virtual void SCSI_OnForcedAbort() = 0;
  virtual void SCSI_OnShowBufferStatus(int bytecount) = 0;
};
```

Listato A.1: Definizione della classe CSCSI

e dovrà implementare i metodi virtuali, in particolare SCSI_OnDispatchData affinché salvi i dati nel file. Al momento in cui il programma chiama il metodo SCSI_Init per l'oggetto di tipo CSignalAcq, l'indirizzo dell'oggetto viene salvato in un puntatore pSCSI a cui le routine C possono accedere ogni qualvolta abbiano necessità di comunicare con il resto del programma, ad esempio quando ci siano dei campioni disponibili da elaborare. Affinché CSCSI funzioni effettivamente da interfaccia è necessario che l'applicazione comunichi con le routine C tramite CSCSI e che queste comunichino con il resto dell'applicazione sempre tramite la sua mediazione, ignorando come l'interfaccia CSCSI sia effettivamente implementata nei suoi successori. Quindi il suddetto puntatore non può essere di tipo *CSignalAcq bensì *CSCSI. Utilizzando metodi non-virtuali, quando le routine C hanno dati disponibili e chiamano il metodo SCSI_OnDispatchData tramite il puntatore pSCSI, ad essere chiamato è il metodo CSCSI::SCSI_OnDispatchData e non CSignalAcq::SCSI_OnDispatchData come vorremmo. Con l'utilizzo di metodi virtuali, invece, viene sempre chiamato il metodo relativo all'oggetto, anche se la chiamata viene fatta tramite un puntatore ad un predecessore, quindi nell'esempio viene correttamente chiamato CSCSI::SCSI_OnDispatchData.

A.1.2 Classe CEP

La classe CEP è derivata direttamente da CSCSI e può essere utilizzata come fondamenta per applicazioni che acquisiscano, e magari successivamente ela-

borino, la risposta evocata da uno stimolo (o evento) acustico o visivo. Dal punto di vista dell'acquisizione del segnale, si trova un gradino sopra CSCSI in quanto si occupa di filtrare e decimare il segnale acquisito, memorizzarlo in un buffer e invocare un metodo virtuale ogni qual volta l'epoca relativa ad uno stimolo risulti completo. Si occupa inoltre di tutto il timing relativo alla presentazione e alla rimozione dello stimolo (o evento).

Il metodo StartStimuli serve per iniziare la presentazione degli stimoli. Devono essere indicati la durata (in ms) dello stimolo, il periodo con cui l'evento deve essere ripetuto e la durata dell'epoca relativa allo stimolo (ovvero l'intervallo post-stimolo che deve essere considerato connesso all'evento). StopStimuli ferma il processo.

La classe utilizza due buffer circolari, uno per i segnali acquisiti ed uno per gli stimoli presentati. Il primo di questi viene creato tramite AllocateSignalBuffer che riserva una quantità di memoria sufficiente a contenere i segnali almeno per il numero di secondi richiesto. Questo buffer viene rilasciato tramite FreeSignalBuffer. Il metodo virtuale SCSI_OnDispatchData di CSCSI viene implementato in CEP e si occupa di convertire il segnale da AD units a μV, filtrarlo e decimarlo (come descritto nel paragrafo 5.1.1) e infine salvarlo nel buffer circolare. GetSample e GetSamples restituiscono, rispettivamente, un singolo campione o l'indirizzo di un buffer di appoggio in cui viene copiato un intervallo di campioni. Entrambi restituiscono dati corretti nell'ipotesi che dall'istante dei campioni richiesti il buffer circolare non abbia compiuto un giro completo.

Il secondo buffer circolare serve invece per contenere informazioni sugli ultimi stimoli (o eventi) presentati. Tra queste c'è il numero progressivo dello stimolo e il numero progressivo del campione nel momento in cui lo stimolo è stato presentato (ovvero il valore restituito da SCSI_GetCurrentSampleNum in quel momento). Oltre a ciò, ad ogni stimolo può essere associata una struttura personalizzata contenente i dati relativi al particolare tipo di stimolo implementato. Questa può essere utilizzata, ad esempio, nel caso del Donchin Speller, per memorizzare quale riga/colonna sia illuminata o, nel caso di un mouse, la direzione. AllocateStimuliBuffer riserva memoria per questo buffer circolare mentre FreeStimuliBuffer la libera. GetStimulusData può essere utilizzato per restituire i dati relativi allo stimolo indicato. Il metodo SaveCompleteEpoch viene chiamato alla fine di ogni epoca relativa ad uno sti-

```
class CEP : public CSCSI
{
public:
  CEP();
  virtual ~CEP();

  static uint MsecToSamp(uint ms);
  static uint SampToMsec(uint samples);

  BOOL StartStimuli(uint stimuliduration, uint stimuliperiod, uint epochlength, const char *
      savetofilename = NULL);
  BOOL StopStimuli();

protected:
  void SCSI_OnMessage(LPCSTR msg);
  void SCSI_OnForcedAbort();
  void SCSI_OnShowBufferStatus(int bytecount);

  BOOL AllocateSignalBuffer(float minsizesec);
  BOOL AllocateStimuliData(uint numstimulitoremember, uint stimulusdatasize);
  void FreeSignalBuffer();
  void FreeStimuliData();

  virtual BOOL TimerStimuli(uint stimulusnumber, uint stimulussample, void *stimulusdata) {
      return FALSE;}
  virtual void TimerStimuliOff(uint stimulusnumber, uint stimulussample, void *stimulusdata) {}
  virtual BOOL TimerEpochComplete(uint stimulusnumber, uint stimulussample, void *stimulusdata,
       BOOL overthreshold) {return FALSE;}

  BOOL SignalOverthreshold(uint startsample, uint numsamples);
  BOOL GetStimulusData(uint stimulusnumber, uint *stimulussample, uint *stimulustask, void **
      stimulusdata);
  float *GetSamples(uint startsample, uint numsamples, uint channel);
  float GetSample(uint samplenum, uint channel);
  BOOL SaveCompleteEpoch(std::ofstream *file, uint stimulusnumber);

//  data members
//  ...

private:
  void SCSI_OnDispatchData(BYTE *data, int num, bool first);

  static void CALLBACK TimerCallback(UINT uTimerID, UINT uMsg, DWORD dwUser, DWORD dw1, DWORD
      dw2);
  static void CALLBACK QueueTimerCallback(PVOID lpParameter, BOOLEAN TimerOrWaitFired);
  static void GetStimulusDataPtr(void *p, uint **stimulusnumber, uint **stimulussample, uint **
      stimulustask, CEP ***obj, void **stimulusdata);
};
```

Listato A.2: Definizione della classe CEP

molo ed accoda al file indicato i dati relativi allo stimolo e l'epoca stessa, cioè il segnale acquisito per un periodo che inizia con la presentazione dell'evento e si estende per la durata precedentemente impostata.

La classe dichiara tre metodi virtuali che devono essere implementati nelle classi derivate. Il primo di questi è TimerStimuli che si occupa di decidere lo stimolo e visualizzarlo. A questo metodo viene fornito il numero progressivo dello stimolo, il numero progressivo del campione attuale ed un puntatore alla struttura associata allo stimolo. Questa struttura è vuota e deve essere riempita da TimerStimuli in base allo stimolo visualizzato. Nel caso del Donchin Speller, ad esempio, questo metodo deve scegliere l'indice della riga/colonna da evidenziare, memorizzarlo nella struttura ed infine occuparsi dell'output evidenziando tale riga/colonna sullo schermo. In fase di training deve ritornare un valore booleano che indica se questo stimolo sia task-relevant ovvero se sia quello a cui il soggetto deve prestare attenzione. Il secondo metodo virtuale è TimerStimuliOff, il cui compito è quello di rimuovere lo stimolo dallo schermo. I parametri che gli vengono forniti sono gli stessi di TimerStimuli ma questa volta la struttura non è vuota ma contiene i dati che sono stati impostati da quest'ultimo metodo al momento in cui lo stimolo è stato presentato (in particolare il tipo di stimolo). L'ultimo di questi metodi virtuali è TimerEpochComplete che viene chiamato quando l'epoca relativa ad uno stimolo viene completata. Questo significa il segnale correlato all'evento è disponibile e può essere salvato o elaborato. I dati che vengono forniti a questo metodo sono gli stessi che nei due visti sinora, più un parametro booleano che indica se nel corso dell'epoca il segnale abbia superato una determinata soglia, suggerendo quindi la presenza di un segnale estraneo all'attività cerebrale (ad esempio di tipo muscolare). Questo metodo può utilizzare GetSamples per ottenere i segnali acquisiti nell'intervallo temporale d'interesse.

Finora è stato descritto come utilizzare la classe CEP ma può essere interessante capire il suo funzionamento. Come detto in precedenza la parte di acquisizione e preelaborazione del segnale è svolta da SCSI_OnDispatchData. La rimanente parte, ovvero il timing degli eventi, si basa sul meccanismo di timer e callback fornito dal sistema operativo. Questo meccanismo consiste nell'effettuare una chiamata al sistema operativo affinché imposti un timer (evento unico o periodico) allo scadere del quale deve essere eseguita una fun-

zione (callback) di cui gli forniamo l'indirizzo. I timer che sono stati utilizzati sono di due tipi: il *mulimedia timer* ed il *queue timer*. Il primo è molto preciso ma piuttosto pesante dal punto di vista delle risorse di sistema impiegate. Inoltre tutti gli eventi impostati per questo tipo di timer vengono eseguiti da un unico thread, quindi, se la funzione chiamata in risposta ad uno di questi richiede più tempo del previsto, ne possono risentire le prestazioni di tutto il sistema. È quindi adatto a compiti brevi in cui è richiesta un'alta precisione temporale. Il secondo invece può decidere di creare un thread separato per eseguire le funzioni callback risultando perciò più adatto per compiti in cui la precisione temporale non è determinante ma che possono richiedere più tempo del previsto. A causa di queste diverse caratteristiche dei due tipi di timer, per la visualizzazione e rimozione degli stimoli viene utilizzato un *multimedia timer* mentre per chiamare TimerEpochComplete al termine della epoca viene usato un *queue timer*.

L'intero processo inizia quando StartStimuli viene chiamato. I suoi parametri stimuliduration , stimuliperiod e epochlength rappresentano rispettivamente la durata di ogni stimolo, ogni quanto viene presentato e quanto dura l'epoca ad esso relativa. Al fine di ottenere due timer periodici di periodo stimuliperiod , sfasati di stimuliduration , vengono inizialmente creati due timer: uno periodico con periodo pari a stimuliperiod che scandirà l'istante in cui andrà presentato lo stimolo; l'altro, "una tantum", con durata pari a stimuliduration al cui scadere verrà impostato un nuovo timer periodico di periodo stimuliperiod che scandirà l'istante in cui andrà rimosso lo stimolo. Per ognuno di questi timer viene impostata come funzione callback il metodo TimerCallback. Affinché sia possibile ottenerne l'indirizzo (indispensabile al momento della creazione del timer) è necessario che il metodo sia definito static . I metodi statici operano sulla classe e non sull'oggetto in quanto sono privi del puntatore this e non possono accedere direttamente agli altri membri (variabili o metodi) della classe. È possibile aggirare questo problema grazie al fatto che al momento della creazione del timer è possibile specificare un argomento che verrà poi passato alla funzione di callback. Se impostiamo tale argomento tramite la parola chiave this (che è un puntatore all'oggetto corrente) al momento in cui TimerCallback viene eseguito esso, benché sia un metodo statico, può utilizzare tale argomento per accedere a tutte le variabili ed i metodi dell'oggetto. In particolare potrà chiamare TimerStimuli o TimerStimuliOff affinché si occupino

```
class CBCI : public CEP
{
public:
  CBCI();
  virtual ~CBCI();

  BOOL LoadPolynomial(const char *filename);

protected:
  float ClassifyEpoch(uint startsample, uint numsamples);

  CWavelet m_Wavelet;
  char *m_Polynomial;
};
```

Listato A.3: Definizione della classe CBCI

di mostrare o rimuovere lo stimolo corrente. Ogni volta che TimerCallback è chiamata in corrispondenza dell'inizio dello stimolo questa crea un nuovo timer "una tantum", questa volta di tipo *queue timer*, affinché dopo un tempo epochlength ci informi che l'epoca relativa all'evento è completa. Questo ultimo timer utilizza QueueTimerCallback come callback. Anche in questo caso al momento della creazione del timer è consentito specificare un parametro che verrà poi passato alla callback al momento della chiamata. Tramite questo parametro viene passato un puntatore alla struttura contenente i dati relativi allo stimolo. QueueTimerCallback si occupa di salvare l'epoca e di chiamare il metodo virtuale TimerEpochComplete per l'elaborazione del segnale correlato all'evento.

A.1.3 Classe CBCI

La classe CBCI è derivata da CEP ma invece che svilupparne le funzionalità si limita ad affiancare a queste la capacità di classificare un'epoca. Il metodo ClassifyEpoch consente infatti di calcolare la combinazione lineare di alcuni coefficienti della trasformata wavelet continua (CWT) del segnale. La definizione della combinazione lineare da implementare può essere caricata da un file tramite il metodo LoadPolynomial per poi essere utilizzata da ClassifyEpoch. Questo file indica quali coefficienti della trasformata wavelet utilizzare, a quale scala, di quale canale EEG e come questi debbano essere combinati linearmente. Per eseguire la trasformata wavelet viene utilizzato l'oggetto m_Wavelet di tipo CWavelet. L'utilizzo e il funzionamento di questa classe è spiegato al paragrafo A.2.1.

A.1.4 Classe CMouseBCI

La classe CMouseBCI è derivata da CBCI ed è la prima delle classi derivate da CSCSI che può essere concretamente utilizzata. Mentre le altre sono classi astratte che definiscono o ereditano metodi virtuali la cui implementazione è demandata ad una delle classi figlie, CMouseBCI è una classe concreta ed istanziabile. Questa classe implementa una specie di mouse. Quattro rettangoli sono disposti ai quattro lati dello schermo e vengono evidenziati uno alla volta in ordine casuale. Il fatto che una delle quattro direzioni venga evidenziata rappresenta quindi uno stimolo, un evento. La risposta esogena per lo stimolo a cui il soggetto è interessato si distinguerà dalle altre. Viene valutata l'uscita del classificatore corrispondente all'epoca che segue ciascuno stimolo. Il cursore viene poi spostato orizzontalmente di un valore proporzionale alla differenza tra l'uscita relativa allo stimolo sulla destra e quella relativa allo stimolo sulla sinistra. Analogo procedimento viene fatto per la direzione verticale.

Per realizzare questo mouse è necessario innanzitutto istanziare un oggetto di tipo CMouseBCI. È necessario informare l'oggetto se si vuole iniziare una sessione di *train* o di *use*. La prima consiste nel sottoporre il soggetto agli stimoli e registrarne l'attività EEG conseguente al fine di ottenere un training set con cui addestrare un classificatore. La seconda consiste invece nel testare un classificatore precedentemente addestrato. Questo può essere fatto in due modi: disegnando la traccia determinata dallo spostamento del mouse oppure muovendo l'immagine del desktop mantenendo il puntatore del mouse al centro dello schermo (con un funzionamento simile al *Magnifier* di Windows®). Il metodo che consente di impostare il tipo di sessione e, nel caso di una sessione di *train*, quale sia la direzione "target" e la lunghezza della sessione. Nel caso invece si tratti di una sessione di *use* è necessario chiamare LoadPolynomial affinché carichi da file la funzione utilizzata dal classificatore.

Per iniziare la sessione è necessario chiamare il metodo Init che a sua volta utilizza AllocateSignalBuffer e AllocateStimuliBuffer per creare i due buffer circolari descritti nel paragrafo A.1.2. All'interno del buffer circolare per gli stimoli viene riservata memoria affinché ad ogni stimolo venga associata una struttura di tipo MouseData. Tale struttura (mostrata nel listato A.4) contiene la direzione dello stimolo, la direzione "target" (in fase di adde-

```
class CMouseBCI : public CBCI
{
public:
  CMouseBCI();
  virtual ~CMouseBCI();

  BOOL Init();
  BOOL SetMode(uint mode, uint targetstimulus, uint numsets, uint updateevery);
  void PrepareMouseControl(HWND outputwnd, int innerwindowwidth, int innerwindowheight,
       COLORREF inactivecolor, COLORREF activecolor, COLORREF activebtncolor);

protected:
  void DrawMouseControl(ushort box, COLORREF color, BOOL overthreshold = FALSE);
  void RedrawMagnifier();
  void Move(float dx, float dy);

  virtual BOOL TimerStimuli(uint stimulusnumber, uint stimulussample, void *stimulusdata);
  virtual void TimerStimuliOff(uint stimulusnumber, uint stimulussample, void *stimulusdata);
  virtual BOOL TimerEpochComplete(uint stimulusnumber, uint stimulussample, void *stimulusdata,
       BOOL overthreshold);

private:
// data members
// ...

};

struct MouseData {
  ushort control;
  ushort targetcontrol;
  float out;
};
```

Listato A.4: Definizione della classe CMouseBCI e della struttura MouseData

stramento) e una variabile in cui verrà salvata l'uscita del classificatore per quella epoca non appena questa sia stata calcolata. A questo punto chiamando SCSI_StartSampling si fa iniziare il campionamento, con PrepareMouseControl si mostra su schermo il pannello "spento" e con StartStimuli si dà inizio alla presentazione degli stimoli. Quando viene chiamato PrepareMouseControl è necessario indicare le dimensioni dei pulsanti e l'*handle* della finestra nella quale dovrà apparire il pannello affinché possa disegnare al suo interno. Per interrompere la presentazione degli stimoli è suffuciente chiamare StopStimuli e per terminare l'acquisizione SCSi_StopSampling.

Il nucleo della classe MouseBCI consiste essenzialmente nell'implementazione dei tre metodi virtuali dichiarati in CEP: TimerStimuli, TimerStimuliOff e TimerEpochComplete. Il primo di questi ha lo scopo di decidere quale dei quattro pulsanti dovrà illuminarsi. Gli stimoli sono raggruppati in gruppi di sei e in ciascuno di questi sono presenti due stimoli "neutri" ed i restanti sono uno per ciascuna delle quattro direzioni. TimerStimuli deve quindi scegliere il

prossimo stimolo tenendo conto di quelli che, all'interno del solito gruppo, sono già stati presentati. Deve inoltre evitare che lo stesso stimolo venga presentato due volte di seguito (cioè che l'ultimo di un gruppo si ripeta nel primo del nuovo gruppo di sei) in quanto questo può falsare la percezione come ipotizzato al paragrafo 6.1.3. Una volta scelto lo stimolo da presentare, ne viene salvato il codice nella struttura MouseData dopo di che viene disegnato sullo schermo colorando il rispettivo rettangolo. TimerStimuliOff non fa altro che leggere dalla struttura MouseData quale fosse lo stimolo presentato e colorare il rispettivo rettangolo di grigio tornando così alla configurazione neutra iniziale. Il metodo TimerEpochComplete, chiamato al termine dell'epoca relativa ad uno stimolo, applica ClassifyEpoch all'epoca a cui si riferisce e salva l'uscita di questa funzione nella struttura MouseData. Nel caso lo stimolo attuale sia l'ultimo di un gruppo di sei, TimerEpochComplete recupera dalle strutture MouseData relative a ciascuno degli ultimi sei stimoli qual'era la direzione illuminata e il valore dell'uscita del classificatore. Combinando queste informazioni calcola di quanto deve essere spostato il cursore in direzione orizzontale e verticale e chiama Move affinché applichi tale spostamento.

A.2 Trasformata wavelet

A.2.1 Classe CWavelet

La classe CWavelet serve ad effettuare la trasformata wavelet continua di un segnale. Tale trasformata viene calcolata tramite l'algoritmo descritto alla fine del paragrafo 2.3. La dichiarazione di CWavelet è mostrata nel listato A.5.

La prima operazione necessaria per utilizzare CWavelet è chiamare il metodo LoadPsiInteg per caricare da file la funzione $\gamma(t)$. È necessario specificare il nome del file e le dimensioni della cache per γ_a ovvero quante diverse successioni γ_a potranno essere conservate affinché al successivo utilizzo non debbano essere ricavate di nuovo. Il formato richiesto per il file con la funzione $\gamma(t)$ e un esempio di come possa essere creato con MATLAB® sono mostrati nel listato A.6.

A questo punto per calcolare la CWT di un segnale è sufficiente chiamare CWT specificando il vettore che contiene i campioni del segnale, il numero

```
class CWavelet
{
public:
  CWavelet();
  virtual ~CWavelet();

  bool LoadPsiInteg(const char* nomefile, uint scalescachesize);
  int CWT(float *x, int xlen, float scale, float **result);
  float CWTSample(float *x, int xlen, float scale, int sample);

  int Corr(float *sig1, int sig1len, float *sig2, int sig2len, float **result, int support = 0)
      ;
  int Corr(float *sig1, int sig1len, float *sig2, int sig2len, float **result, int imin, int
      support);
  int Diff(float coeff, float *x, int xlen);
  void DestroyAll();
  uint GetNumCoefs(float scale);
  char m_WaveletName[SIZE_WAVELET_NAME];

protected:
  uint GetScaleCoeffs(float scale, float **coeffs);
  float *m_ScalesCache;
  uint m_ScalesCacheSize;
  float **m_ScalesCacheCoeffs;
  float m_XMax;
  float *m_PsiInteg;
  uint m_PsiIntegPoints;
  float *m_ResultBuffer;
  uint m_ResultBufferSize;
};
```

Listato A.5: Definizione della classe CWavelet

```
% Format specifications for the binary file with the gamma(x) function
%
% type      description
%-------------------------------------------------------------------------
% char[]    Null-terminated string (var. lenght <20) with the name of the wavelet
% uint      number (n) of samples of the integral of the psi function
% float     x value of the last sample
% float[n]  vector of length nsamp with the values of gamma

wname = 'rbio3.3'; % name of the wavelet
precis = 17; % precision (a good choice is between 12 and 20)
[fgamma, xval] = intwave('rbio3.3', precis);
fid = fopen('intpsi.bin', 'w');
fwrite(fid, wname, 'char');
fwrite(fid, 0, 'char');
fwrite(fid, length(fgamma), 'uint');
fwrite(fid, xval(end), 'float');
fwrite(fid, fgamma, 'float');
fclose(fid);
```

Listato A.6: Formato del file con la funzione $\gamma(t)$ e un esempio di come possa essere ottenuto.

```
int CWavelet::CWT(float *x, int xlen, float scale, float **result)
{
  float *psiinteg;
  int psiinteglen = (int)GetScaleCoeffs(scale, &psiinteg);
  if(!psiinteglen)
    return 0;
  int corrlen = Corr(x, xlen, psiinteg, psiinteglen, result, xlen + 1);
  int resultlen = Diff(-sqrt(scale), *result, corrlen);
  return resultlen;
}
```

Listato A.7: Il metodo CWavelet::CWT

di campioni, la scala a ed un puntatore che al ritorno contiene la posizione del vettore con i risultati. Il metodo restituisce la lunghezza del vettore dei risultati che, se l'operazione è stata eseguita con successo, coincide con la lunghezza del segnale in ingresso in quanto vengono conservati solo i campioni centrali della correlazione e scartate le code. Nel listato A.7 si può vedere come l'algoritmo implementato rispecchi quello riportato alla fine del paragrafo 2.3.

Se, invece di voler calcolare la trasformata di un segnale, si desidera calcolare solo coefficienti sparsi, è sufficiente chiamare CWTSample per ciascuno di essi in quanto questo metodo dà in uscita il valore di un singolo coefficiente della trasformata wavelet. Devono essere indicati il vettore con i campioni del segnale, il numero di campioni, la scala e l'indice del campione che si vuole calcolare. Questo indice segue la stessa numerazione di CWT ovvero: il risultato ottenuto chiamando CWTSample con indice 3 è uguale all'elemento di indice 3 del vettore ottenuto con CWT.

A.3 Algoritmo genetico

A.3.1 Classe PolinomialPopulation

Questa classe rappresenta una popolazione di polinomi ed i relativi meccanismi di evoluzione. Il sue fine è quello di consentire di evolvere questi polinomi affinché migliorino le loro prestazioni nel risolvere un problema specifico. Questo è, nel caso dell'approccio con classificazione binaria (par. 5.2.3), ottenere il minimo errore quadratico medio su un insieme di singole epoche. Nel caso dell'approccio *endpoint* (par. 5.2.4), si tratta invece di prendere gruppi di quattro epoche relative a stimoli in ciascuna delle quattro direzioni e mas-

```
inline float PolynomialPopulation::CoeffCrossover(float coeff1, float coeff2)
{
  // Eshelman and Schaffer — Blend Crossover BLX–alpha
  double gamma = (1.0 + 2.0 * m_BlendCoeff) * dgenrand() — m_BlendCoeff;
  return  (float)(coeff1 + gamma * (coeff2 — coeff1));
}
```

Listato A.8: Metodo CoeffCrossover

simizzare lo spostamento nella direzione voluta, minimizzando la deviazione in direzione ortogonale.

La popolazione di polinomi viene evoluta con le tecniche descritte nel paragrafo 5.2.2. Per ciascuno dei parametri del polinomio, siano essi coefficienti, indici o esponenti, viene utilizzato un *blend crossover* ed il listato A.8 mostra la sua implementazione nel caso di due coefficienti. Tramite la funzione dgenrand viene calcolato un numero casuale[1] nell'intervallo $[0, 1]$. Di conseguenza il parametro gamma può variare da $-$m_BlendCoeff a 1 + m_BlendCoeff. Se si sceglie m_BlendCoeff pari a 0.1, il coefficiente "figlio" sarà scelto casualmente nell'intervallo compreso tra i coefficienti originari, con la possibilità di sconfinare da entrambi i lati del 10% dell'ampiezza dell'intervallo stesso.

Il metodo Tournament (listato A.9) mostra come viene implementato il tournament. Se il secondo argomento viene omesso oppure è true viene effettuato un tournament positivo, se impostato a false viene effettuato un tournament negativo. Nel primo caso viene restituito l'indice dell'individuo con objective function migliore (cioè minore), nel secondo quello con objective function peggiore.

L'objective function, come già visto al paragrafo 3.5, esprime quanto l'individuo, in questo caso un polinomio, adempia al suo compito. Nel caso dell'approccio *endpoint* (par. 5.2.4) il polinomio deve dare un'uscita che:

- massimizzi la differenza tra l'uscita del classificatore relativa allo stimolo nella direzione desiderata e quella relativa allo stimolo nella direzione opposta;
- minimizzi la differenza tra le due uscite del classificatore relative alle direzioni ortogonali a quella desiderata.

[1] Nell'intero progetto, per la generazione di numeri casuali, viene utilizzato l'algoritmo Mersenne Twister realizzato da Makoto Matsumoto e Takuji Nishimura (ved. [28]).

```
uint PolynomialPopulation::Tournament(uint tournamentsize, bool best /* = true */)
{
  float objdrawn, objselected = FLT_MAX;
  uint selected, drawn;

  for(uint i = 0; i < tournamentsize; i++) {
    drawn = genrand() % m_PopulationSize;
    objdrawn = (best ? 1.0 : -1.0) * m_Objective[drawn];
    if(objdrawn < objselected) {
      objselected = m_Objective[drawn];
      selected = drawn;
    }
  }
  return selected;
}
```

Listato A.9: Metodo Tournament

Il listato A.10 mostra il metodo IndividualObjective che serve appunto per calcolare la objective function dell'individuo polinomio. Viene effettuata una scansione delle epoche disponibili al fine di raggrupparle in quartetti contenenti un'epoca per ciascuna delle quattro direzioni. Per ogni quartetto vengono calcolati rightdir, ort1dir e ort2dir. Il primo esprime lo spostamento nella direzione giusta, calcolato come differenza tra l'uscita relativa allo stimolo nella direzione desiderata e quella relativa allo stimolo nella direzione opposta. Il secondo, ort1dir, rappresenta invece lo spostamento nella direzione ortogonale a quella desiderata e l'ultimo, ort2dir è il valore assoluto di tale spostamento. La fitness del polinomio è data da un fattore di merito, ovvero rightdir medio, meno due penali. La prima di queste, il valore assoluto di ort1dir medio, fa sì che il punto finale cui si giunge non abbia un'eccessiva deviazione ortogonale. La seconda, ort2dir medio, penalizza invece le deviazioni ortogonali ad ogni passo, imponendo che, oltre che arrivare nel punto giusto, ci si arrivi andando il più dritto possibile.

Il nucleo della classe è il metodo OneStep che serve per evolvere la popolazione di una generazione. Ogni generazione è composta da un numero di cicli pari alla dimensione della popolazione. Ognuno di questi cicli ha probabilità m_CrossoverProb di essere costituito da un crossover. I due genitori vengono scelti tramite tournament e poi incrociati. Il nascituro viene posto in una posizione temporanea. Nel caso il ciclo non sia costituito da un crossover c'è una probabilità m_MutationProb che un individuo subisca una mutazione di tipo *headless chicken*, altrimenti viene copiato inalterato in una posizione temporanea. In ciascuno dei tre casi (crossover, mutazione, copia) il nuovo

```
float PolynomialPopulation::IndividualObjective(uint numindividual)
{
  double rightdir = 0;
  double ort1dir = 0;
  double ort2dir = 0;
  uint size = 0;
  for(uint p0 = 0, p1 = 0, p2 = 0, p4 = 0; ; p0++, p1++, p2++, p4++, size++) {
    while((m_VectOfTargets[p0] != 0) && (p0 < m_SizeOfSet)) p0++;
    if(p0 == m_SizeOfSet) break;
    while((m_VectOfTargets[p1] != 1) && (p1 < m_SizeOfSet)) p1++;
    if(p1 == m_SizeOfSet) break;
    while((m_VectOfTargets[p2] != 2) && (p2 < m_SizeOfSet)) p2++;
    if(p2 == m_SizeOfSet) break;
    while((m_VectOfTargets[p4] != 4) && (p4 < m_SizeOfSet)) p4++;
    if(p4 == m_SizeOfSet) break;
    double eval0 = atan(EvalIndividual(numindividual, ((void**)m_VectOfVars)[p0]));
    double eval1 = atan(EvalIndividual(numindividual, ((void**)m_VectOfVars)[p1]));
    double eval2 = atan(EvalIndividual(numindividual, ((void**)m_VectOfVars)[p2]));
    double eval4 = atan(EvalIndividual(numindividual, ((void**)m_VectOfVars)[p4]));
    rightdir += eval0 - eval2;
    ort1dir += eval1 - eval4;
    ort2dir += fabs(eval1 - eval4);
  }
  rightdir /= size;
  ort1dir /= size;
  ort2dir /= size;
  double pen1 = .2;
  double pen2 = .2;
  float fitness = (float)(rightdir - pen1 * fabs(ort1dir) - pen2 * ort2dir);
  return -fitness;
}
```

Listato A.10: Metodo IndividualObjective

individuo viene copiato dalla posizione temporanea in una nuova posizione rimpiazzando un individuo preesistente. Quest'ultimo viene scelto tramite tournament negativo. Viene infine calcolata la fitness del nuovo individuo.

A.4 R.O.C.

A.4.1 Classe CROC

La classe CROC si occupa della visualizzazione della curva R.O.C. (Receiver Operating Characteristic) [29] di un classificatore binario. La sua dichiarazione è mostrata nel listato A.12. Questa classe fornisce funzionalità ad un controllo di tipo *owner draw button* e ne gestisce gli eventi. Un controllo *owner draw button* è un'area di una finestra la cui gestione, ed in particolare il disegno, è delegato ad una classe personalaizzata. Utilizzando il *Resource Editor* si può creare un *owner draw button* nella finestra desiderata. È poi necessario chiamare il metodo SubclassDlgItem, che CROC eredita da CButton,

```
bool PolynomialPopulation::OneStep()
{
  uint newindpos;
  for(uint iv = 0; iv < m_PopulationSize; iv++) {
    if(dgenrand() < m_CrossoverProb) {
      uint par1 = Tournament(m_TournamentSize);
      uint par2 = Tournament(m_TournamentSize);
      CrossoverToTemp(par1, par2);
    } else {
      if(dgenrand() < m_MutationProb) {
        uint par = Tournament(m_TournamentSize);
        RandomFillIndividual(GetTemp());
        CrossoverToTemp(par, GetTemp());
      } else {
        CopyToTemp(Tournament(m_TournamentSizeCopy));
      }
    }
    newindpos = Tournament(m_TournamentSizeNeg, false); // negative tournament
    MoveTempTo(newindpos);
    m_Objective[newindpos] = IndividualObjective(newindpos);
  }
  m_Generation++;
  if(m_Error)
    return false;
  return true;
}
```

Listato A.11: Metodo OneStep

indicando l'ID del controllo creato ed un puntatore alla finestra in cui il controllo è contenuto. Così facendo comunichiamo al sistema operativo che sarà un oggetto di tipo CROC a gestire gli eventi relativi al controllo creato. In particolare DrawItem verrà chiamato ogni volta che il controllo deve essere ridisegnato e questo ci consente di tracciare la curva ROC al suo interno.

DrawItem disegna gli assi del grafico e, se il vettore m_ROCPoints contiene dei punti, traccia la curva che li congiunge. Per permettere a CROC di calcolare questi punti è necessario chiamare ProcessOutputsTargets la cui implementazione è mostrata nel listato A.13. È necessario passare a questo metodo un vettore con le uscite "analogiche" del classificatore, un vettore con l'uscita binaria corretta e la dimensione di questi vettori ovvero il numero di esempi su cui la ROC viene calcolata. È inoltre possibile specificare, come ulteriore parametro, targetprob ovvero la frequenza di casi positivi prevista in fase di utilizzo; il significato di questo parametro sarà chiarito più avanti.

ProcessOutputsTargets è il metodo chiave della classe, quello che calcola i punti della curva a partire dalle uscite del classificatore. Il suo funzionamento (mostrato nel listato A.13) può essere suddiviso in più fasi. Inizialmente viene

```
struct ROCPoint {
  float Output;
  float FP;
  float TP;
};

class CROC : public CButton {
protected:
  ROCPoint *m_ROCPoints;
  uint m_NumPoints;
  float m_Threshold;
  float m_TargetProb;
  virtual void DrawItem(LPDRAWITEMSTRUCT lpDIS);

public:
  CROC();
  ~CROC();

  bool ProcessOutputsTargets(float *outputs, bool *targets, uint size, float targetprob = .5f);
  void SetThreshold(float thres);
  double MutualChannelInformation(double TP, double FP, double P1);
  bool SaveToFile(const char *filename);
};
```

Listato A.12: Definizione della classe CROC

allocata memoria per il vettore dei punti. Successivamente vengono copiati i vettori di ingresso (con le uscite del classificatore e le uscite corrette) in vettori temporanei. Questi vettori vengono ordinati per valore decrescente di uscita del classificatore tramite un algoritmo di *shell sort*. Si effettua poi una scansione del vettore ordinato e si crea il vettore con i punti della ROC. In questo vettore vengono salvati il numero di veri positivi, il numero di falsi negativi e l'estremo inferiore dell'intervallo di soglie che ha quel tasso di TP e FP.

Oltre a tracciare una curva ROC statica, la classe consente anche di impostare una soglia e mostrare informazioni al riguardo. Viene evidenziato tramite un pallino rosso il punto della ROC corrispondente alla soglia scelta e mostrati il valore di TP e FP per quel punto. Considerando il classificatore come un canale di comunicazione con in ingresso la risposta corretta ed in uscita la risposta data dal classificatore, viene mostrata la percentuale di classificazioni corrette e la mutua informazione di canale quando l'ingresso ha probabilità targetprob di essere positivo. Ad esempio nel caso del Donchin Speller è noto a priori che la probabilità che la lettera si trovi nella riga o colonna in esame è 1/6 e quindi la percentuale di risposte corrette sarà $1/6 \cdot TP + 5/6 \cdot (1 - FP)$. Il metodo che consente di impostare la soglia è

```
bool CROC::ProcessOutputsTargets(float *outputs, bool *targets, uint size, float targetprob /*=
    .5f*/)
{
  if(m_ROCPoints)
    delete[] m_ROCPoints;
  float *toutputs;
  bool *ttargets;
  try {
    toutputs = new float[size];
    ttargets = new bool[size];
    m_ROCPoints = new ROCPoint[size + 2];
  } catch(std::bad_alloc) {
    m_ROCPoints = NULL;
    return false;
  }
  // Copy the vector with the classifier outputs and the vector with target binary output
  uint postarget = 0;
  for(uint m = 0; m < size; m++) {
    toutputs[m] = outputs[m];
    ttargets[m] = targets[m];
    if(ttargets[m])
      postarget++;  // Counts how many positive (true) targets
  }
  // Sorts the outputs (shell sort algorithm)
  for (uint gap = size / 2; gap > 0; gap /= 2)
    for (uint i = gap; i < size; i++)
      for (int j = i - gap; j >= 0 && toutputs[j] < toutputs[j + gap]; j -= gap) {
        float tempf = toutputs[j];
        bool tempb = ttargets[j];
        toutputs[j] = toutputs[j + gap];
        ttargets[j] = ttargets[j + gap];
        toutputs[j + gap] = tempf;
        ttargets[j + gap] = tempb;
      }
  // Creates the list of points of the ROC
  float outprec = m_ROCPoints[0].Output = 1e38f; // the first point is the bottom left angle
  uint FP = m_ROCPoints[0].FP = 0;
  uint TP = m_ROCPoints[0].TP = 0;
  m_NumPoints = 1;
  for(int k = 0; k < size; k++) {
    if(toutputs[k] != outprec) {
      m_ROCPoints[m_NumPoints].Output = toutputs[k];
      m_ROCPoints[m_NumPoints].FP = 1.0 * FP / (size - postarget);
      m_ROCPoints[m_NumPoints].TP = 1.0 * TP / postarget;
      m_NumPoints++;
      outprec = toutputs[k];
    }
    if(ttargets[k])
      TP++;
    else
      FP++;
  }
  m_ROCPoints[m_NumPoints].Output = -1e38f; // the last point is the upper right angle
  m_ROCPoints[m_NumPoints].FP = 1;
  m_ROCPoints[m_NumPoints].TP = 1;
  m_NumPoints++;
  m_TargetProb = targetprob;
  delete[] toutputs;
  delete[] ttargets;
  return true;
}
```

Listato A.13: Implementazione della funzione ProcessOutputsTargets

SetThreshold che, se chiamato ogni volta che uno *slider control* viene modificato, consente di variare la soglia interattivamente e vederne immediatamente l'effetto sullo schermo.

Bibliografia

[1] Robert M. Berne and Matthew N. Levy. *Principi di fisiologia*. Casa Editrice Ambrosiana, 2002.

[2] Andrea Brovelli. Basi fisiologiche dei segnali biologici complessi. www.univ.trieste.it/˜brain/Segnali/index.html.

[3] Tomas Hruby and Petr Marsalek. Event-related potentials - the P300 wave. *Acta Neurobiologiae Experimentalis*, 63:55–63, 2003.

[4] E. Donchin and M. G. H. Coles. Is the P300 a manifestation of context updating? *Behav. Brain Sci.*, 11:355–372, 1988.

[5] J. R. Wolpaw, N. Birbaumer, W. J. Heetderks, D. J. McFarland, P. H. Peckham, G. Schalk, E. Donchin, L. A. Quatrano, C. J. Robinson, and T. M. Vaughan. Brain-computer interface technology: a review of the first international meeting. *IEEE transactions on rehabilitation engineering*, 8(2):164–173, Jun 2000.

[6] G. Pfurtscheller, C. Neuper, C. Guger, W. Harkam, H. Ramoser, A. Schlögl, B. Obermaier, and M. Pregenzer. Current Trends in Graz Brain-Computer Interface (BCI) Research. *IEEE transactions on rehabilitation engineering*, 8(2):216–219, Jun 2000.

[7] J. R. Wolpaw, D. J. McFarland, and T. M. Vaughan. Brain-Computer Interface research at the Wadsworth Center. *IEEE transactions on rehabilitation engineering*, 8(2):222–226, Jun 2000.

[8] Niels Birbaumer, Andrea Kübler, Nimr Ghanayim, Thilo Hinterberger, Jouri Perelmouter, Jochen Kaiser, Iver Iversen, Boris Kotchoubey, Nicola Neumann, and Herta Flor. The Thought Translation Device (TTD)

for completely paralyzed patients. *IEEE transactions on rehabilitation engineering*, 8(2):190–193, Jun 2000.

[9] L. A. Farwell and E. Donchin. Talking off the top of your head: A mental prosthesis utilizing event-related brain potentials. *Electroencephalogr. Clin. Neurophysiol.*, 70:510–523, 1988.

[10] Emanuel Donchin, Kevin M. Spencer, and Ranjith Wijesinghe. The mental prosthesis: assessing the speed of a P300-based Brain-Computer Interface. *IEEE transactions on rehabilitation engineering*, 8(2):174–179, Jun 2000.

[11] P. R. Kennedy, R. A. E Bakay, M. M. Moore, K. Adams, and J. Goldwaithe. Direct control of a computer from the human central nervous system. *IEEE transactions on rehabilitation engineering*, 8(2):198–202, Jun 2000.

[12] N. Weiskopf, R. Veit, M. Erb, K. Mathiak, W. Grodd, R. Goebel, and N. Birbaumer. Physiological self-regulation of regional brain activity using real-time functional magnetic resonance imaging (fMRI): methodology and exemplary data. *NeuroImage*, 19:577–586, 2003.

[13] R. Quian Quiroga and M. Schürmann. Functions and sources of event-related EEG alpha oscillations studied with the Wavelet Transform. *Clinical Neurophysiology*, 110:643–654, 1999.

[14] Piotr Jerzy Durka. *Time-frequency analyses of EEG.* PhD thesis, Warsaw University, 1996.

[15] Mitchell Slep. Single trial analysis of eeg signals. Technical report, Princeton University, may 2003.

[16] Vasil Kolev, Tamer Demiralp, Juliana Yordanova, Ahmet Ademoglu, and Ümmühan Isoglu-Alkaç. Time-frequency analysis reveals multiple functional components during oddball P300. *NeuroReport*, 8:2061–2065, may 1997.

[17] Rodrigo Quian Quiroga. *Quantitative analysis of EEG signals: Time-frequency methods and Chaos theory.* PhD thesis, Medical University Lübeck, 1998.

[18] The MathWorks, Inc. *Wavelet Toolbox User's Guide*.

[19] Darrell Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4:65–85, 1994.

[20] Benjamin Blankertz and Gabriel Curio. Description of the algorithm used for the classification of the "Albany P300" data set IIb from the BCI Competition 03.

[21] Caterina Cinel, Riccardo Poli, and Luca Citi. Possible sources of perceptual errors in P300-based speller paradigm. *Biomedizinsche Technik*, (in press).

[22] Ron Kohavi and George H. John. Wrappers for feature subset selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

[23] B. Blankertz, K. R. Muller, G. Curio, T. M. Vaughan, G. Schalk, J. R. Wolpaw, A. Schlogl, C. Neuper, G. Pfurtscheller, T. Hinterberger, M. Schroder, and N. Birbaumer. The BCI competition 2003: progress and perspectives in detection and discrimination of EEG single trials. *IEEE Transactions on Biomedical Engineering*, 51(6):1044–1051, jun 2004.

[24] Fabrizio Beverina, Giorgio Palmas, Stefano Silvoni, Francesco Piccione, and Silvio Giove. User adaptive BCIs: SSVEP and P300 based interfaces. *PsychNology Journal*, 1(4):331–354, 2003.

[25] James B. Polikoff, H. Timothy Bunnell, and Winslow J. Borkowski Jr. Toward a p300-based computer interface.

[26] Documentation 2nd Wadsworth BCI Dataset. ida.first.fraunhofer.de/projects/bci/competition/albany_desc/albany_desc_ii.pdf.

[27] Tomas Hruby and Petr Marsalek. Interactions between P300 and passive probe responses differ in different visual cortical areas. *Acta Neurobiologiae Experimentalis*, 61:93–104, 2001.

[28] Makoto Matsumoto and Takuji Nishimura. Mersenne Twister: a 623-dimensionally equidistributed uniform pseudo-random number genera-

tor. *ACM Transactions on Modeling and Computer Simulation*, 8(1), Jan 1998.

[29] Guido Valli and Giuseppe Coppini. *Bioimmagini*. Number 8 in Collana di ingegneria biomedica. Patron Editore, 2002.

[30] Francisco Herrera, Manuel Lozano, and Jose L. Verdegay. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review*, 12(4):265–319, 1998.

[31] Riccardo Poli, Stefano Cagnoni, and Guido Valli. Genetic design of optimum linear and non-linear QRS detectors. *IEEE Transactions on Biomedical Engineering*, 42(11):1137–1141, November 1995.

[32] John Elder and Donald Brown. Induction and polynomial networks. In *Network models for control and processing*, pages 143–198. Intellect Books, 2000.

[33] Peter Meinicke, Matthias Kaper, Florian Hoppe, Manfred Heumann, and Helge Ritter. Improving transfer rates in brain computer interfacing: A case study. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 1107–1114. MIT Press, 2003.

[34] Jessica D. Bayliss. *A flexible Brain-Computer Interface*. PhD thesis, University of Rochester, 2001.

[35] Janne Lehtonen. *EEG-based Brain Computer Interfaces*. PhD thesis, Helsinki University of Technology, 2002.

[36] Jorge Baztarrica Ochoa. *EEG signal classification for Brain-Computer Interface applications*. PhD thesis, Ecole Polytechnique Federale de Lausanne, 2002.

[37] Luca Citi, Riccardo Poli, and Francisco Sepulveda. An evolutionary approach to feature selection and classification in P300-based BCI. *Biomedizinsche Technik*, (in press).