ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Dipartimento di Ingegneria dell'Energia Elettrica e
dell'Informazione "Guglielmo Marconi" - Campus di Cesena
Corso di Laurea Magistrale in Ingegneria Biomedica

IMPLEMENTAZIONE DI ALGORITMI DI MACHINE LEARNING PER LA CLASSIFICAZIONE DEL MOVIMENTO TRAMITE SEGNALI ELETTROENCEFALOGRAFICI

Tesi in: Sistemi Neurali

Presentata da:
DAVIDE MIANI

Relatore:
Prof.ssa ELISA MAGOSSO

Correlatori:
Prof.ssa SILVIA FANTOZZI
Dott. DAVIDE BORRA

ANNO ACCADEMICO 2017–2018
SESSIONE III, APPELLO II

Parole chiave

EEG
Decodifica del movimento
Machine Learning
Deep Learning
Reti neurali convoluzionali

Abstract

I recenti sviluppi nel campo dell'intelligenza artificiale hanno permesso la decodifica del movimento tramite segnale EEG. È stato dimostrato come sia possibile ottenere ottime performance di classificazione fra più task motori impiegando tecniche di Machine Learning (ML) e di Deep Learning (DL), facendo uso per quest'ultime di reti neurali convoluzionali (Convolutional Neural Networks, CNN). Tuttavia, il gold standard per questo tipo di applicazioni rimane un algoritmo di ML, il Filter Bank Common Spatial Pattern (FBCSP) con classificazione tramite Linear Discriminant Analysis (LDA), anche se simili prestazioni sono state recentemente ottenute con una CNN (Deep ConvNet). L'assenza in letteratura di uno studio che considerasse tutte le principali strategie di addestramento ha ispirato l'indagine svolta in questo elaborato. Sono stati implementati sia l'algoritmo FBCSP+LDA che l'architettura Deep ConvNet, addestrati su un dataset EEG (High-Gamma, disponibile online) ottenuto da 14 soggetti per un totale di 13484 trial motori divisi in 4 classi. L'addestramento con trial appartenenti ad un unico soggetto non ha solamente confermato i risultati precedenti, ma anche dimostrato che impiegando un miglior partizionamento del dataset le CNN sono in grado di superare significativamente le accuratezze di FBCSP+LDA (98.2% vs. 89.6%). L'addestramento con trial appartenenti a più soggetti ha ribadito la superiorità delle CNN (71.5% vs. 47.2%); il divario si amplia ulteriormente utilizzando tecniche di trasferimento della conoscenza (91.3% con transfer learning). Inoltre, addestrando i classificatori con diverse finestre temporali sui trial, si è dimostrato come le CNN siano in grado di raggiungere ugualmente buone performance (87.8% vs. 48.6%). In conclusione, questo studio evidenzia come l'uso di CNN rispetto ad altre metodologie di ML sia in grado di decodificare il segnale EEG con ottime prestazioni anche utilizzando strategie di addestramento differenti.

Indice

Notazione

Questa sezione fornisce un riferimento sintetico che descrive la notazione utilizzata nella trattazione matematica.

Numeri e vettori

a: quantità scalare
$\boldsymbol{a}$: quantità vettoriale
$\boldsymbol{A}$: matrice
$\mathbf{A}$: tensore (matrice multi-dimensionale)
A: insieme

Indicizzazione

L'indicizzazione parte da 1.
a_i: elemento i-esimo del vettore $\boldsymbol{a}$
a_{ij}: elemento in posizione (i, j) della matrice $\boldsymbol{A}$
$\boldsymbol{a}_j$: colonna j-esima della matrice $\boldsymbol{A}$
$\{a_{ij}\}$: matrice formata dagli scalari a_{ij}

Variabili aleatorie e probabilità

Le variabili aleatori si rappresentano aggiungendo una sottolineatura.
$\underline{a}$: variabile aleatoria scalare
$\underline{\boldsymbol{a}}$: vettore di variabili aleatorie
$\mathbb{E}[\underline{a}]$: valor medio di $\underline{a}$

Introduzione

Decodificare l'attività cerebrale legata al movimento volontario è fondamentale non solo per comprendere i circuiti motori del cervello umano e l'origine dei disturbi neuromotori, ma anche per lo sviluppo di dispositivi protesici per la neuroriabilitazione. Sono passati più di cinquant'anni da quando la neuroscienza riuscì per la prima volta a correlare le variazioni dello spettro di potenza del segnale elettroencefalografico con il movimento volontario, ma solo recentemente, grazie all'affermarsi dell'intelligenza artificiale, si è stati in grado di classificare con buona accuratezza i singoli task motori eseguiti da un soggetto a partire dalla decodifica del tracciato elettroencefalografico (EEG).

Fra gli algoritmi di machine learning impiegati per decodificare il movimento tramite l'analisi del segnale EEG, l'attuale gold standard è rappresentato dal Filter Bank Common Spatial Pattern (FBCSP), specificatamente progettato sulla base dei correlati funzionali descritti in letteratura, combinato a Linear Discriminant Analysis (LDA). Nonostante ciò, in questo algoritmo è fondamentale progettare opportunamente le feature da estrarre dai dati sulla base della conoscenza pregressa. Tali feature, cioè, non sono apprese automaticamente dall'algoritmo ma assegnate a priori.

Tuttavia, i più recenti successi delle reti neurali convoluzionali nel campo della computer vision hanno fatto crescere l'interesse dei ricercatori nell'utilizzare le tecniche di deep learning, che permettono di apprendere automaticamente le feature rilevanti ai fini del compito oggetto di studio, anche per la decodifica del movimento dai segnali EEG. In particolare, molto recentemente è stata introdotta una architettura di rete di deep learning denominata Deep ConvNet (Schirrmeister et al., Human Brain Mapping 2017) che si è dimostrata in grado di raggiungere le prestazioni di FBCSP+LDA nel decodificare il movimento da segnali EEG.

Sono ancora pochi gli studi presenti in letteratura riguardanti l'utilizzo delle reti neurali per la decodifica del movimento e quasi tutti si concentrano su un'unica strategia di addestramento. In virtù dell'elevata variabilità inter-soggetto del segnale EEG, si preferisce addestrare il classificatore a partire da esempi provenienti da un singolo soggetto (addestramento within-subject), producendo di fatto un classificatore specializzato nella decodifica del segnale di un solo individuo.

Molto più interessanti sono le modalità di addestramento che al contrario fanno uso di esempi provenienti da più soggetti per l'addestramento del classificatore (addestramento between-subject). Una di queste strategie di training è il transfer learning: la rete neurale può essere inizializzata sfruttando il risultato di un precedente addestramento su più soggetti ed in seguito addestrata con pochi esempi di un soggetto completamente nuovo; in questo modo, anche con un insieme dei dati ridotto, è possibile trasferire la conoscenza acquisita precedentemente su più soggetti per la decodifica del nuovo soggetto.

In questo elaborato di tesi si è svolta un'analisi approfondita delle tecniche di machine learning e di deep learning per la decodifica, a partire dai tracciati EEG, di 3 movimenti (finger tapping con mano destra e sinistra, clenching dei piedi) di 14 soggetti (9 di sesso femminile, 4 mancini, età media di 27.5 anni) risultando in una classificazione a 4 vie includendo anche il riposo. Il dataset utilizzato (High-Gamma dataset, Schirrmeister et al.) è disponibile online. Dunque si è implementato FBCSP+LDA e l'architettura Deep ConvNet. Sono state sviluppate successivamente tutte le principali strategie di training e valutata l'influenza sulle prestazioni di decodifica della scelta della finestra temporale (rispetto all'inizio del movimento) entro cui viene estratto il segnale EEG. L'intera implementazione è stata svolta in Python, utilizzando Keras come framework di deep learning.

L'elaborato è strutturato nel seguente modo. Nel Capitolo 1 si descrivono le basi neurofisiologiche dell'elettroencefalografia e i correlati funzionali del movimento per l'utilizzo delle Brain Computer Interface (BCI) movement-based. Nel Capitolo 2 si analizzano la storia e i concetti fondamentali del machine learning, nonché le tecniche di estrazione delle feature e di classificazione per la decodifica del movimento. Nel Capitolo 3 si ripercorre l'evoluzione delle metodologie di deep learning, dai primi modelli neurali alle reti convoluzionali, esponendo nel dettaglio lo stato dell'arte delle architetture e delle strategie

di addestramento. Nel Capitolo 4 si riportano le metodologie utilizzate per il raggiungimento dei risultati, che vengono esposti nel Capitolo 5. Nel Capitolo 6 vengono discussi i risultati ottenuti e infine si articolano le conclusioni finali.

Capitolo 1

Anatomo-fisiologia dei potenziali EEG per la classificazione del movimento

1.1 L'elettroencefalografia

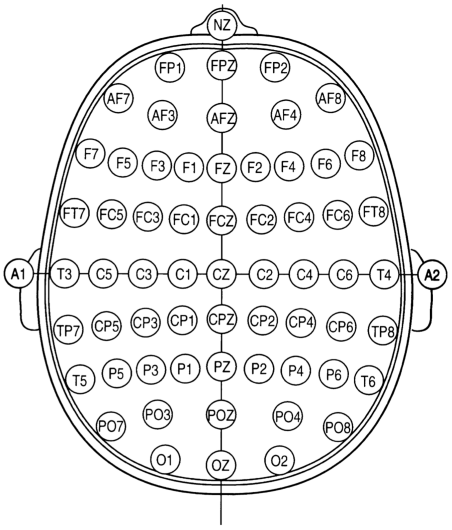
Introduzione e cenni storici.

L'elettroencefalografia (EEG) è un metodo d'indagine per l'analisi funzionale dell'attività elettrica della corteccia cerebrale. Il segnale elettroencefalografico è il risultato di potenziali postsinaptici eccitatori ed inibitori, generati dagli alberi dendritici di svariati gruppi di neuroni che lavorano in sincronia. I potenziali elettrici cerebrali possono essere acquisiti tramite l'elettroencefalografo, un sistema di acquisizione del segnale che si interfaccia con il soggetto tramite elettrodi (eventualmente premontati su cuffia) posti sullo scalpo [1].

Il pioniere delle registrazioni di attività bioelettriche cerebrali fu Richard Caton, il quale, nel 1875, riuscì ad acquisire deboli potenziali di animale. Per l'elettroencefalografia umana si dovette aspettare il 1924, quando Hans Berger osservò i pattern temporali delle onde elettriche cerebrali usando strisce metalliche poste sullo scalpo del soggetto ed un sensibile galvanometro. Dal 1924 al 1938 egli pose le basi per molte delle presenti applicazioni dell'elettroencefalografia e coniò il termine **elettroencefalogramma**, usato oggi comunemente per descrivere la registrazione dei potenziali elettrici cerebrali [2].

Elettrodi e montaggi.

Dalle strisce metalliche di Berger, nell'arco di quasi un secolo, l'elettroencefalografia ha fatto chiaramente enormi progressi. Un punto di svolta si è avuto negli anni '50, quando è stato definito il **sistema internazionale 10/20** per il montaggio degli elettrodi. Il motivo di tale nomenclatura deriva dal fatto che la distanza fra due elettrodi adiacenti è il 10/20% della lunghezza totale fra fronte e retro e fra lato destro e sinistro dello scalpo, rispettivamente [3]. Tale standard prevede l'utilizzo di 21 elettrodi, ma negli anni sono state proposte successive modifiche e migliorie, fino ad arrivare a montaggi anche da 128 elettrodi [4]. Ad oggi, fra le varie tipologie di elettrodi, i più utilizzati sono gli **elettrodi a disco** (anche detti *a coppetta*); si tratta di piccoli dischetti con un foro centrale tramite il quale viene applicato un gel elettrolitico, con la doppia funzione di abbassare l'impedenza della pelle e di collante [5]. Per semplificare di molto l'applicazione degli elettrodi, solitamente essi sono premontati su di una cuffia, che li assicura alle giuste distanze reciproche secondo i vari standard.



(a) *montaggio 10%*, una estensione dello standard 10/20 a 64 elettrodi [3].

(b) cuffia con 128 elettrodi premonatati [6].

Figura 1.1: evoluzioni del sistema 10/20 ad oggi impiegate in ricerca. Come si può vedere nello schema del pannello (a), ogni elettrodo è identificato tramite una sigla alfanumerica che si compone delle iniziali del lobo cerebrale sul quale è posto e di un numero, pari se collocato sull'emisfero destro, dispari se sul sinistro; si ha invece una *Z* se l'elettrodo si trova sull'asse centrale verticale. Si veda sezione 1.3 per una descrizione dettagliata di lobi ed emisferi cerebrali.

L'elettroencefalografo multicanale.

Il sistema che si collega agli elettrodi per la vera e propria misura del segnale EEG è l'elettroencefalografo multicanale, che permette di registrare il segnale proveniente da diversi elettrodi contemporaneamente. Ogni elettrodo è collegato con l'elettroencefalografo tramite una morsettiera (jack box), dove ogni coppia elettrodo-morsetto va a costituire un canale dell'elettroencefalogramma. Il numero di canali può variare a seconda dell'applicazione, dagli 8/16 canali per gli utilizzi più comuni sino anche a 64/128 canali per scopi di ricerca. La linea di acquisizione del segnale si compone poi dei seguenti blocchi:

- **Blocco di amplificazione.** Come vederemo in sezione 1.2, il segnale EEG ha ampiezze nell'ordine dei μV; ciò rende necessario un blocco analogico di amplificazione con specifiche particolarmente stringenti [5] (elevato guadagno differenziale, elevata resistenza di ingresso e rapporto di reiezione di modo comune).

- **Blocco di conversione analogico-digitale.** Una volta che il segnale analogico è stato opportunamente amplificato, esso raggiunge il blocco di conversione A/D, che si compone di ulteriori quattro componenti in serie: **(1) filtro anti-aliasing** (si ottiene un segnale continuo a banda limitata), **(2) campionatore** (si ottiene un segnale tempo-discreto), **(3) quantizzatore**, **(4) blocco di codifica** (si ottiene infine un segnale digitale). Nei moderni elettroencefalografi la frequenza di campionamento è in genere compresa fra i 256 Hz e i 5 kHz. Per quanto riguarda la risoluzione invece, di solito i convertitori di questi dispositivi sono a 12 o 16 bit, fino addirittura a 24 bit [6], presentando intervalli di ampiezza dell'ordine del μV.

- **Blocco di memorizzazione e signal processing.** Una volta ottenuto il segnale digitale, esso viene memorizzato, anche a valle di una ulteriore elaborazione. A questo livello vengono ad esempio applicati importanti algoritmi di filtraggio, come il **filtro notch**, il quale elimina la frequenza di rete.

Altri componenti fondamentali di un elettroencefalografo sono i seguenti: **(1) sistema di calibrazione:** identifica eventuali inadeguatezze o imprecisioni lungo la linea di acquisizione applicandovi un segnale di forma nota (solitamente un'onda quadra alla frequenza di 1 Hz); **(2) circuito di controllo delle impedenze di elettrodo:** si

occupa di misurare e mostrare a video le impedenze di ogni singolo elettrodo, così che l'operatore effettuante il montaggio possa aggiungere gel elettrolitico fino a stabilizzare le impedenze in un range standard di 5 ± 1 $k\Omega$; questa operazione è fondamentale, dal momento che un aumento di tali impedenze provoca un incremento dell'errore di interconnessione, il quale può portare il disturbo differenziale in ingresso molto al di sopra del segnale utile [5].

1.2 Il segnale elettroencefalografico

I biopotenziali elettrici cerebrali.
I biopotenziali cerebrali possono essere distinti in tre tipologie in base alla loro ampiezza e possibilità di rilevarli a livello di scalpo [2]:

1. **Oscillazioni di fondo.** Sono i potenziali cerebrali con ampiezza maggiore (1-200 μV), sono presenti e ben rilevabili a livello dello scalpo, quindi determinano in maniera sostanziale l'elettroencefalogramma. La loro ampiezza e dunque la loro potenza (alle varie frequenze) può essere alterata da eventi esterni o interni.
2. **Potenziali evocati.** Sono deflessioni che si presentano in risposta ad uno stimolo esterno (o interno) ed hanno ampiezza confrontabile con il livello di rumore dell'elettroencefalogramma; sono necessarie specifiche elaborazioni del segnale EEG per separare queste risposte dalle oscillazioni EEG di fondo.

Le oscillazioni di fondo e i potenziali evocati nascono dall'attività di estese popolazioni di neuroni, che danno origine a segnali sufficientemente elevati da essere rilevabili mediante elettrodi posizionati sullo scalpo. Al contrario si definisce infine l'ultima tipologia di biopotenziali cerebrali:

3. **Potenziali generati a livello di singoli neuroni.** Misurare l'attività elettrica di una singola cellula eccitabile cerebrale richiede l'utilizzo di elettrodi direttamente inseriti in essa.

Pertanto l'elettroencefalografia di superficie rileva le oscillazioni di fondo, le loro modificazioni, ed i potenziali evocati [2]; dal segnale EEG è poi possibile estrarre informazioni relative al movimento andando ad analizzare la banda frequenziale che va dagli 0.5 Hz fino anche ai 150 Hz [7] (si veda il prossimo paragrafo per una descrizione delle bande principali e di quelle di maggiore interesse per il movimento).

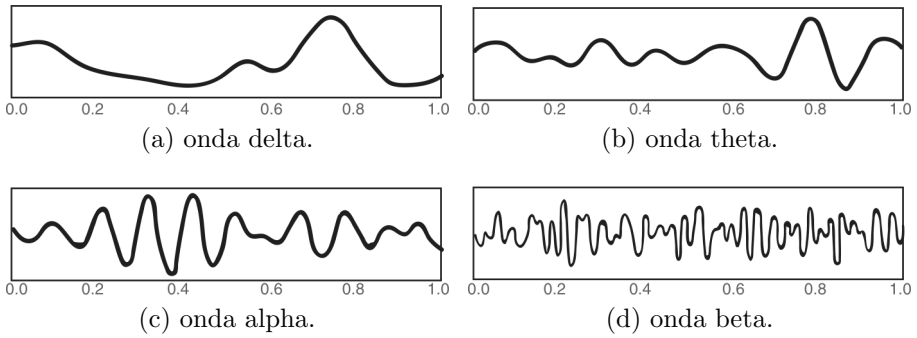
I ritmi cerebrali.

L'analisi del tracciato elettroencefalografico ha permesso di constatare che diversi gruppi neuronali in diversi distretti dell'encefalo codificano insieme per task differenti, andando a formare così delle onde ben riconoscibili e classificabili in base alla propria banda frequenziale. Canonicamente, si individuano **cinque bande principali**: in ordine crescente di frequenza e decrescente di ampiezza, si hanno le onde delta, theta, alpha, beta e gamma.

	banda (Hz)	**ampiezza (μV)**	**regione corticale**	**funzioni cerebrali correlate**
$\boldsymbol{\delta}$	0.5-4	20-200	frontale, parietale	• predominante nei neonati • sonno profondo negli adulti • task di attenzione prolungata
$\boldsymbol{\theta}$	4-8	20-100	temporale, parietale	• di maggior intensità nei bambini • assopimento in adolescenti ed adulti • coinvolto nei meccanismi di repressione di un'azione o risposta
$\boldsymbol{\alpha}$	8-14	20-50	occipitale, frontale	• relax/riflessione • molto ben evidente ad occhi chiusi • associate anch'esse all'inibizione
$\boldsymbol{\beta}$	14-25	5-30	parietale, frontale	• concentrazione • allerta e ansia
$\boldsymbol{\gamma}$	> 25	1-20	frontale	• processi sensoriali cross-modali (coinvolgimento di 2 o più sensi) • memoria a breve termine

Tabella 1.1: descrizione schematica dei ritmi cerebrali principali [8][9][10]. In letteratura si trovano bande frequenziali ed ampiezze di poco differenti a seconda dell'autore; si è dunque proceduto scegliendo i valori più spesso riportati. Le funzioni cerebrali qui descritte fanno riferimento per lo più alla sfera cognitiva, mentre ci si sofferma sui ritmi coinvolti nei task motori nel paragrafo successivo. Infine, per una definizione e disamina delle aree corticali qui indicate, si veda sezione 1.3.

Come è possibile constatare da tabella 1.1, ampiezza e frequenza, in generale, sono negativamente correlate [11]. Per esempio, il ritmo theta (4-8 Hz) ha ampiezza circa doppia rispetto al ritmo alpha (8-14 Hz); quest'ultimo ha poi ampiezze maggiori rispetto alle oscillazioni beta (14-25 Hz). Inoltre, visto che l'ampiezza delle oscillazioni è proporzionale al numero di neuroni attivi in maniera sincrona, i gruppi di neuroni associati alle oscillazioni lente contengono più cellule rispetto a quelli associati alle oscillazioni rapide [11].



(a) onda delta. (b) onda theta.

(c) onda alpha. (d) onda beta.

Figura 1.2: andamento temporale qualitativo di diversi ritmi neurali [9].

Onde cerebrali e movimento.
Fra i ritmi cerebrali principali, la banda beta è quella più correlata al movimento, mostrando evidenti oscillazioni durante l'attività contrattile muscolare [12]. Solo recentemente si sono osservate caratteristiche cinematiche del movimento eseguito nella banda delta [13]. In aggiunta ai cinque ritmi visti poi, ne sono stati individuati altri due che giocano un ruolo fondamentale nel movimento: le **onde mu** (a cui ci si riferisce anche con Rolandic wicket rhythm, 8-12 Hz) e le **alte onde gamma** (high-gamma, > 42 Hz).

- **Ritmi mu.** Si tratta di onde ben riconoscibili all'interno della banda alpha che hanno la proprietà di bloccarsi durante l'esecuzione, l'osservazione o l'immaginazione di un movimento volontario[14]. Si ritiene questo blocco fortemente correlato con l'attivazione dei **neuroni specchio** (una particolare tipologia di neuroni scoperti alla fine degli anni '80 dai ricercatori dell'Università di Parma) che popolano la corteccia motoria e che si attivano sia quando si compie un movimento che quando lo si osserva (da qui il nome di neuroni *specchio*) [15].

- **Ritmi high-gamma.** Mentre per l'indagine dei processi cognitivi la banda gamma contiene contributi informativi solo fino a frequenze relativamente basse (40 Hz circa), così non è per il movimento volontario, in cui, utilizzando particolari setup sperimentali schermati elettromagneticamente, è stato possibile osservare un incremento dell'attività dell'alta banda gamma, in particolare per frequenze fra i 60 e i 90 Hz [7].

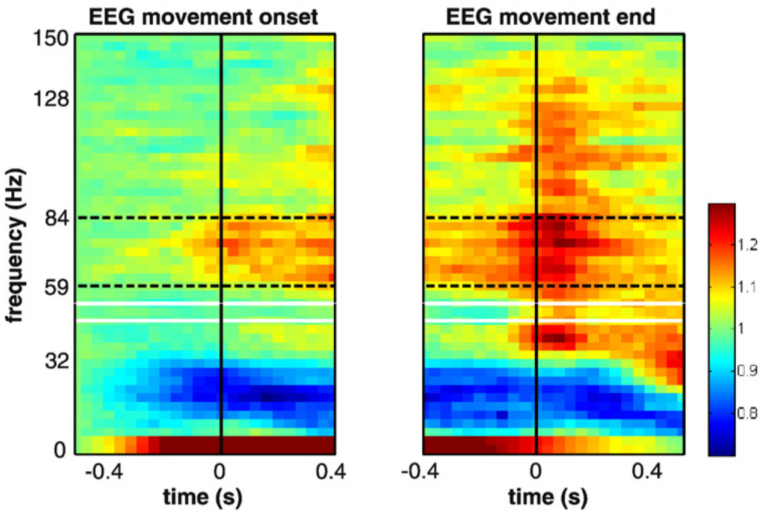


Figura 1.3: spettrogramma del segnale EEG per un task di reaching (afferrare un oggetto) in condizione di schermatura elettromagnetica. I due grafici sono relativi all'elettrodo C3, il quale si trova sopra l'area del cervello che codifica per la mano con la quale si esegue il movimento. Una potenza relativa pari ad 1 corrisponde a nessun cambiamento, valori compresi fra 0 ed 1 corrispondono ad un decremento di potenza, viceversa per i valori maggiori di 1. In generale, l'attività della banda high-gamma aumenta approssimativamente fra i 60 e gli 85 Hz. In aggiunta, la potenza dello spettro diminuisce per le frequenze più basse, in accordo con quanto detto a proposito delle onde mu [7].

1.3 Anatomia del sistema nervoso

Il sistema nervoso centrale e periferico.

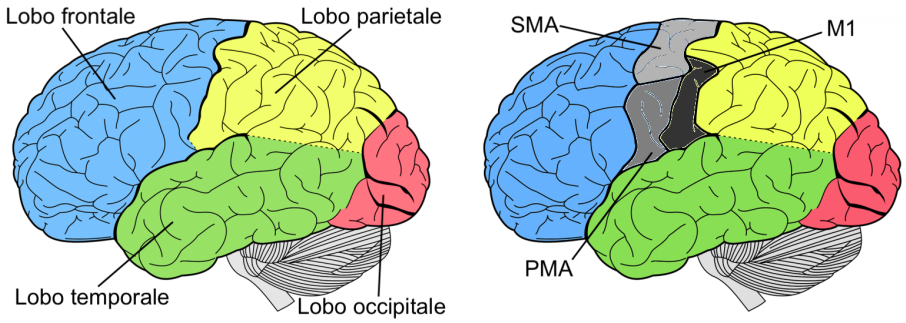
È possibile suddividere il sistema nervoso in sistema nervoso centrale (SNC) e sistema nervoso periferico (SNP); il primo è composto da **encefalo e midollo spinale**, il secondo dai **gangli nervosi e dai nervi** che interfacciano il sistema nervoso centrale con i muscoli e gli organi di senso.

L'encefalo.

Di particolare interesse per le applicazioni di questa tesi è l'encefalo. Completamente contenuto nella scatola cranica, si divide a sua volta in tronco encefalico, cervelletto e cervello. Il **tronco encefalico** è la prosecuzione craniale del midollo spinale e possiede circuiti neurali che connettono le strutture inferiori con le superiori, integrandone le informazioni. Il **cervelletto** è la porzione dell'encefalo situata nella scatola cranica posteriore ed è il principale centro della coordinazione motoria. Infine, il **cervello** è l'organo più raffinato del sistema nervoso centrale e si divide in telencefalo (porzione più superficiale) e diencefalo (porzione più interna che si collega al tronco encefalico).

Il telencefalo e la corteccia cerebrale.

Mentre il diencefalo è composto principalmente dal talamo, il telencefalo, che si articola in due emisferi differenti uniti dal corpo calloso, è costituito dalla corteccia, gangli della base, amigdala ed ippocampo.



(a) lobi cerebrali. (b) aree motorie.

Figura 1.4: schemi dei lobi cerebrali e delle aree motorie [16].

La corteccia cerebrale è lo strato più esterno ed è formato da neuroni e dalla glia; le fibre neurali che compongono la corteccia non sono mielinizzate ed è per questo che essa ha il tipico colore grigiastro (per questo viene anche detta materia grigia **materia grigia**).

Come si vedere in figura 1.4a, è possibile suddividere la corteccia a livello anatomico in quattro lobi principali: frontale, parietale, temporale e occipitale. È poi anche possibile una complessa suddivisione funzionale delle varie aree della corteccia; tuttavia, in questa tesi si considerano solamente le aree di interesse motorio. Osservando la figura 1.4b, si possono allora distinguere tre principali aree motorie:

- **Premotor Area (PMA).** La corteccia premotoria identifica i gruppi muscolari che è necessario attivare per generare uno specifico movimento. Essa prende in considerazione la posizione corrente e la postura del corpo, per questo possiede diverse afferenze dai canali sensoriali.
- **Supplemental Motor Area (SMA).** La corteccia motoria supplementare svolge un ruolo chiave nella stabilizzazione del corpo e nella coordinazione di entrambi i lati corporei, ad esempio durante operazioni bi-manuali.
- **Corteccia motoria primaria (M1).** Le prime due aree qui esposte sono spesso raggruppate sotto un un'unica area detta corteccia motoria secondaria (M2); si tratta infatti di aree integrative che afferiscono alla corteccia motoria primaria, la vera responsabile del movimento. È infatti la corteccia motoria primaria ad inviare sinapsi ai neuroni motori del midollo spinale, che innervano poi i singoli muscoli in periferia. Per fare ciò, gli assoni dei neuroni della corteccia motoria primaria si raccolgono in due vie discendenti dette **vie piramidali**, che raggiungono direttamente il midollo spinale.

Il sistema motorio.

Il sistema motorio è l'insieme di tutti gli organi del sistema nervoso centrale che sono coinvolti nella generazione e propagazione dei segnali neurali motori. Presenta un'organizzazione gerarchica estremamente versatile; infatti anche i livelli inferiori sono in grado di generare movimenti di tipo riflesso, senza l'intervento dei sistemi superiori [17].

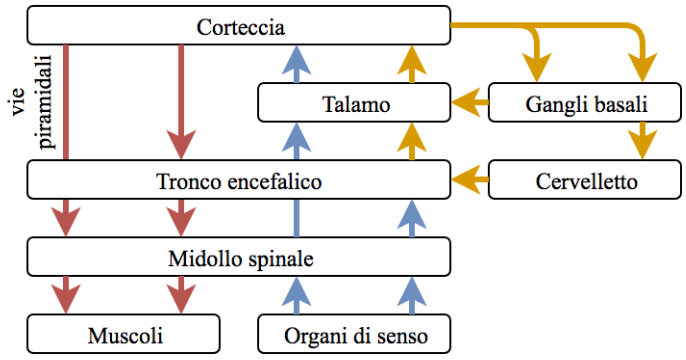


Figura 1.5: schema dei percorsi motori (in rosso), sensoriali (in blu) ed altri in feedback per equilibrio e funzioni di controllo. Appare evidente come i percorsi motori siano meno complessi, per consentire la più rapida propagazione possibile.

1.4 Correlati funzionali del movimento

Definizione di correlato funzionale.

Dopo aver visto le caratteristiche del segnale elettroencefalografico, i ritmi neurali che è possibile individuare e l'anatomia del sistema nervoso e del sistema motorio, viene adesso descritto come questo segnale va a modificarsi in corrispondenza del movimento volontario.

Mentre il controllo neurale dei movimenti riflessi inconsci ha luogo a livello spinale, il movimento volontario coinvolge le diverse strutture gerarchiche analizzate in sezione 1.3. In particolare, il fulcro del movimento volontario risiede nella corteccia motoria: qui i neuroni piramidali integrano informazione dalle regioni corticali e sub-corticali ed i loro assoni costruiscono connessioni inter ed intra-emisferiche [17]. Per questa ragione la letteratura si è arricchita negli anni di svariati studi che hanno cercato di correlare l'attività neurale in questo distretto cerebrale con il movimento volontario, ricercandone cioè i correlati funzionali. Fra quelli trovati, due sono i principali: il flusso sanguigno cerebrale distrettuale (ovvero limitato ad una particolare area dell'encefalo) ed alcuni eventi elettroencefalografici che verranno analizzati qui in seguito.

Flusso sanguigno cerebrale distrettuale.

Nel 1980 Roland et al. [18] osservarono un aumento dell'apporto sanguigno in varie regioni della corteccia motoria sia durante il movimento delle dita della mano che durante l'immaginazione dello stesso. Nel 1993 Rao et al. [19] impiegarono per la medesima indagine l'utilizzo di risonanza magnetica funzionale (fMRI), costatando lo stesso fenomeno. Contemporaneamente si è notato che anche il movimento volontario delle braccia fa variare l'apporto sanguigno, ma in misura molto più contenuta rispetto a quanto misurato per le dita: questo conferma che i segmenti motori che hanno sviluppato i controlli più fini sono anche mappati su un'area maggiormente estesa a livello della corteccia [20] (si veda il famoso homunculus corticale di Penfield, in figura 1.6). Gli stessi risultati sono confermati nel 1996 da Sadato et al. [21] con l'ausilio della tomografia ad emissione positronica (PET).

In generale, la misurazione del flusso sanguigno nelle regioni corticali tramite PET o fMRI mostra cambiamenti funzionali localizzati fra le diverse aree motorie a seconda del tipo di movimento, della complessità del task e della sequenza temporale dello stesso [17].

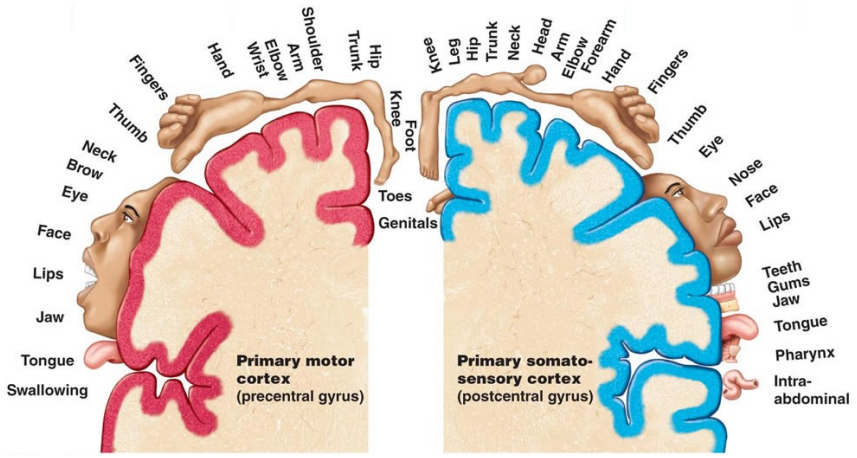


Figura 1.6: rappresentazione dell'homunculus corticale di Penfield [22]. La corteccia contrassegnata in rosso è relativa all'area motoria primaria, che si estende anatomicamente sul giro precentrale, rappresentato anche in figura 1.4b; la corteccia in blu è invece relativa all'area somato-sensoriale, che occupa il giro postcentrale, ovvero l'area direttamente posteriore all'area motoria primaria. Nonostante questo modello sia stato abbandonato da alcuni autori [23], il fatto che la corteccia motoria e sensoriale presenti un'organizzazione somatotipica è ancora una visione comune. In questo modello, le aree del corpo che sono caratterizzate da una maggiore innervazione muscolare e sensoriale sono mappate su un'area corticale più estesa. Per esempio, l'area dedicata alla mano (la quale svolge movimenti anche molto fini e dettagliati, e presente una elevata densità di recettori sensoriali) ha un'estensione maggiore rispetto a braccia piedi [17].

Biopotenziali cerebrali.

Entrambe le tecniche di imaging sopra citate (PET ed fMRI) hanno un'eccellente risoluzione spaziale ma una pessima risoluzione temporale, con frequenze di acquisizione che non permettono una distinzione fra la preparazione e l'esecuzione del movimento. Invece, tramite l'elettroencefalografia, è possibile quantificare i cambiamenti dell'attività neurale corticale in frazioni di secondo, essendo l'EEG tipicamente acquisito con risoluzione temporale dell'ordine del millisecondo [17].

Studiando come il tracciato EEG si modifica prima, durante e dopo il movimento, è possibile individuare correlati funzionali sia nel dominio del tempo che della frequenza. In entrambi i casi, l'assunto di partenza è che il segnale di ciascun canale di un elettroencefalogramma sia **stocastico non stazionario**, per cui un'analisi nel dominio del tempo prende in considerazione la variazione del valor medio, mentre nel dominio della frequenza prende in considerazione la variazione dello spettro di potenza nel tempo.

- **Analisi nel dominio temporale.** Nel 1965, Kornhber et al. [24] individuano in diversi canali EEG la tendenza del valor medio del segnale a procedere verso potenziali maggiormente negativi prima del movimento; i ricercatori chiamarono questo fenomeno **Bereitschaftspotential (BP)**, ovvero potenziale di preparazione, anche detto potenziale premotorio. In figura 1.7 è possibile osservare il fenomeno su tre canali differenti, dove il tempo $t_0 = 0$ indica l'onset del movimento (l'esatto istante temporale in cui ha inizio l'esecuzione del task). È possibile suddividere il fenomeno in tre fasi diverse: **(1) fase di early BP** (2 s pre-onset), in cui il valor medio del segnale comincia molto lentamente ad allontanarsi dalla linea di zero; **(2) fase di BP** (1.5-1.0 s pre-onset), dove la pendenza dello shift del segnale si rende evidente; **(3) fase di late BP** (400 ms pre-onset), dove la pendenza del segnale si rende estremamente negativa (indicata con *NS* in figura) [25]. A livello spaziale, inizialmente il BP si distribuisce in modo uniforme su entrambi gli emisferi; dunque, durante la fase di late BP, esso diventa preponderante sull'area moto-sensoriale controlaterale [17][25].

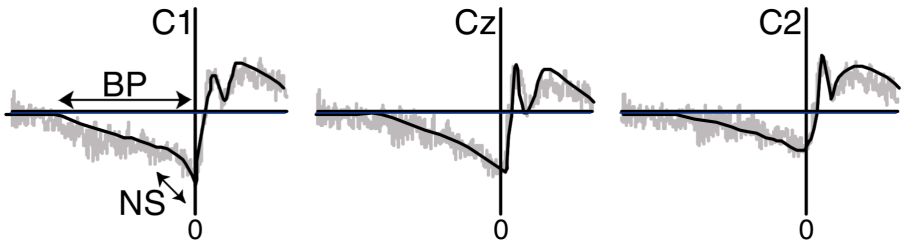


Figura 1.7: Bereitschaftspotential sugli elettrodi in posizione C1, Cz e C2 (si veda figura 1.1a per una locazione esatta di tali elettrodi) [25].

- **Analisi nel dominio frequenziale.** Nel 1977, Pfurtscheller et al. [26] descrivono un forte decremento dello spettro di potenza nella banda mu e beta. Esso avviene circa 2 s prima dell'onset nell'emisfero controlaterale, per poi spostarsi sull'ipsilaterale pochi istanti prima dell'inizio del movimento. Egli chiama questo fenomeno **Event-Related Desynchronization (ERD)** dal momento che è generato dalla desincronizzazione dei neuroni della corteccia motoria negli istanti di tempo che precedono l'onset del movimento. In seguito viene definita anche

l'**Event-Related Synchronization (ERS)**, ovvero un incremento dello spettro di potenza principalmente nelle bande mu, beta e gamma dopo l'onset del movimento, dovuto questo alla sincronizzazione dei neuroni motori [11]. In figura 1.8 è rappresentato il procedimento per il calcolo di ERD ed ERS a partire dal segnale EEG relativo ad N realizzazioni dello stesso task motorio (trial); tale procedura si compone delle seguenti fasi: (1) filtraggio passabanda di ogni trial per isolare la banda di interesse; (2) calcolo della potenza tramite elevazione al quadrato di ogni campione di ogni trial; (3) media della potenza su N trial (mediando su più realizzazioni, la stima dello potenza è più robusta); (4) media mobile per smoothing. A questo punto è possibile definire l'ERD/ERS come la percentuale di decremento/incremento della potenza rispetto ad una baseline di riferimento, secondo l'espressione $\text{ERD\%} = \frac{A-R}{R} \cdot 100$, dove R ed A sono i valori della potenza in intervalli di pre e post-evento, rispettivamente.

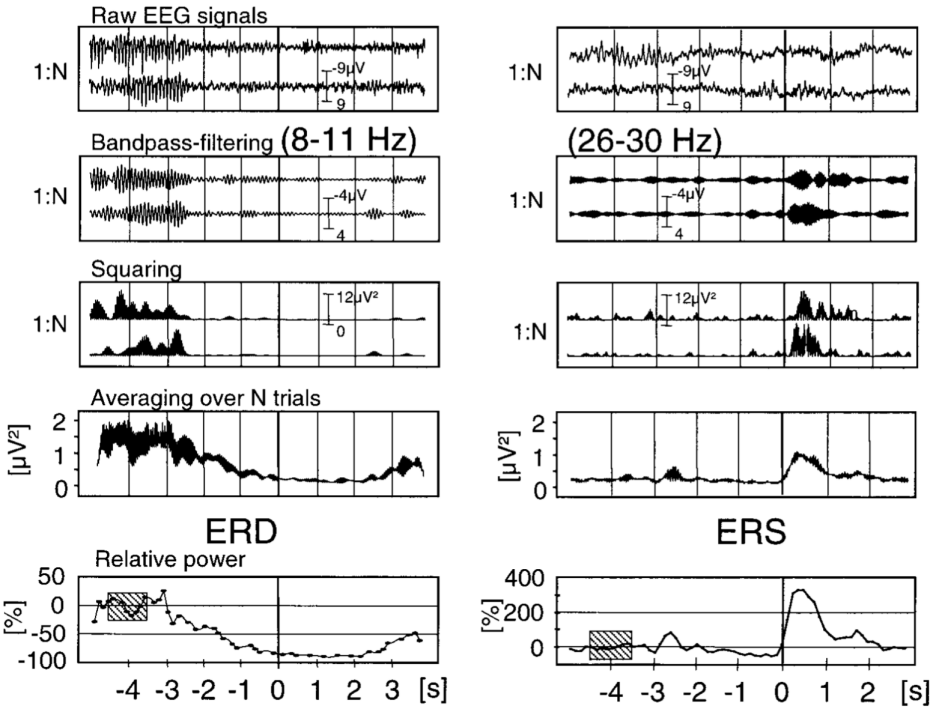


Figura 1.8: procedura per il calcolo di ERD ed ERS in due bande differenti [11].

1.5 Le interfacce neurali

Classificazione del movimento.

Se si volessero individuare i diversi istanti in cui un determinato soggetto compie un movimento, l'analisi del segnale EEG offre certamente correlati funzionali adatti allo scopo, sia nel dominio temporale che in quello frequenziale. Lo step successivo è chiedersi se sia possibile, a partire dagli stessi correlati, classificare i movimenti andando per esempio a determinare quale segmento corporeo si stia muovendo o, per una classificazione ancora più avanzata, quale tipo di movimento si stia compiendo o pianificando. Data la complessità del problema, apparentemente non risolvibile tramite le classiche tecniche di elaborazione dei segnali, allo stato dell'arte ci si affida ad algoritmi di intelligenza artificiale, fra i quali al momento **il gold standard è rappresentato da tecniche di machine learning** [27], analizzate nello specifico nel Capitolo 2. Questi algoritmi sono in grado di apprendere da esempi dei vari task forniti in ingresso in una prima fase di calibrazione; per fare ciò si va ad addestrare un classificatore, il quale poi sarà in grado di distinguere fra i diversi input in una seconda fase di utilizzo.

Definizione di interfaccia neurale.

Una **Brain Computer Interface (BCI)** può essere definita come un sistema che traduce pattern di attività cerebrale di un utente in comandi per una applicazione interattiva, in cui l'attività è misurata di solito tramite elettroencefalografia e processata dal sistema stesso [28]. Per esempio, una BCI può permettere all'utente di muovere un cursore alla sinistra o alla destra dello schermo di un computer se egli immagina di muovere la mano sinistra o la mano destra, rispettivamente.

Dal momento che le BCI rendono possibile il controllo di un computer senza alcun intervento di tipo fisico da parte dell'utente, esse promettono di rivoluzionare molte aree di applicazione: pazienti con disabilità motorie potrebbero accedere a nuovi sistemi di input per comunicare o muoversi (ad esempio tramite sedie a rotelle controllate direttamente attraverso segnali neurofisiologici acquisiti con EEG), così come sarebbe possibile progettare dispositivi riabilitativi per pazienti post-stroke basati su biofeedback elettroencefalografico [29].

Schema a blocchi di una BCI.

Come detto precedentemente, per poter utilizzare una BCI è necessario passare attraverso due fasi distinte:

1. **Fase di addestramento offline.** In questa fase, l'algoritmo di classificazione viene calibrato, andando a selezionare le caratteristiche ottimali fra i diversi canali EEG a disposizione. Dal momento che **il segnale EEG è altamente soggetto-specifico** [27], è necessario acquisire un dataset di training per ogni utente della BCI. Questo set di dati di addestramento contiene segnali EEG registrati mentre l'utente esegue ogni task di interesse più volte, secondo le istruzioni fornite. Sebbene siano attualmente in corso molti sforzi per ottenere modalità operative senza calibrazione, al momento la calibrazione offline è necessaria nella maggior parte delle BCI per ottenere un sistema affidabile [27].
2. **Fase operativa online.** In seguito alla calibrazione, il sistema può riconoscere i pattern di attività cerebrale e tradurli in comandi per il computer. Quando la BCI opera in questa seconda fase online, il sistema si trova in una configurazione a loop chiuso, come è possibile osservare in figura 1.9.

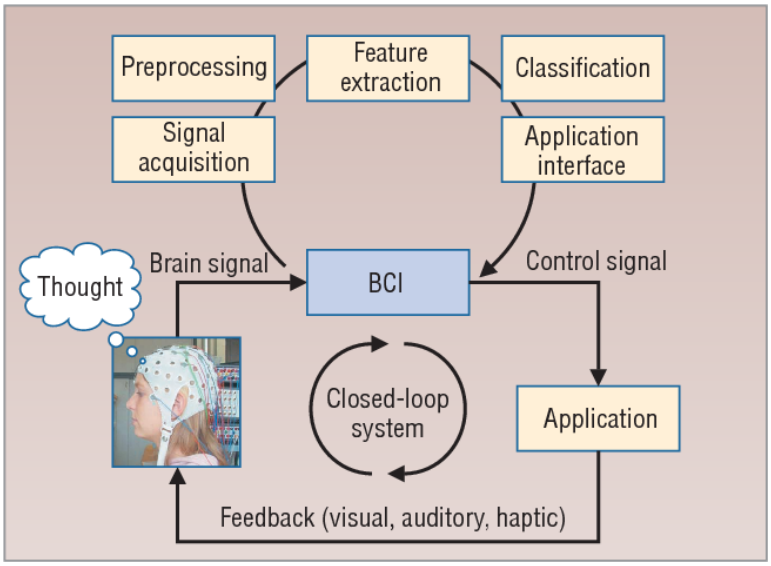


Figura 1.9: schema di una BCI operante online in configurazione a loop chiuso. La sub-routine di BCI rappresentata nel circolo superiore indica le procedure tipiche di una BCI machine learning-based, ma la stessa può essere sostituita con un qualsiasi altro algoritmo di classificazione [29].

In particolare, il segnale EEG è acquisito e processato in real-time dal blocco BCI, il cui risultato è tradotto in un segnale di controllo per una specifica applicazione, che andrà a restituire un feedback all'utente, il quale potrà così verificare se il comando mentale sia stato riconosciuto [29].

Pipeline di machine learning e deep learning a confronto.

È dunque in corrispondenza del blocco BCI di figura 1.9 che si concentra l'operato di questa tesi, andando ad implementare e confrontare differenti pipeline di machine learning (Capitolo 2) e deep learning (Capitolo 3) per la classificazione del movimento. Prima di entrare nel merito di tali algoritmi di intelligenza artificiale, i quali verranno analizzati nel seguito della trattazione, vediamo, a conclusione di questo primo capitolo, le principali differenze fra le due pipeline [4]:

- **Pipeline di Machine Learning (ML).** Tutte le linee di classificazione basate su ML applicano tecniche di estrazione/selezione delle caratteristiche per rappresentare i dati in maniera più compatta e pertinente [27]. Nel caso particolare del segnale EEG, la pipeline di ML è formata da tre blocchi in cascata:

  1. **Feature extraction.** Partendo dal singolo trial, il segnali EEG viene generalmente filtrato sia nel dominio del tempo che nel dominio dello spazio tramite l'utilizzo di filtraggi passa-banda e filtraggi spaziali; in seguito all'applicazione di tali filtri, si ottengono diverse versioni del segnale di partenza, ciascuna rappresentante le caratteristiche (feature) di una specifica banda frequenziale o area spaziale.
  2. **Feature selection.** Fra tutti i segnali estratti nella fase precedente, si vanno ad individuare quelli più informativi tramite un processo di selezione delle caratteristiche.
  3. **Classification.** Infine, l'algoritmo di classificazione vero e proprio viene applicato solamente sulle feature selezionate.

- **Pipeline di Deep Learning (DL).** Al contrario del ML, in cui è necessario implementare tecniche differenti per ciascuno dei tre blocchi sopra decritti, in una pipeline di DL il tutto "condensato" in un unico blocco dove il sistema apprende automaticamente dai dati le migliori caratteristiche per discriminare tra le classi (estrazione e selezione) ed implementa anche la classificazione.

Capitolo 2

Machine learning per la classificazione del movimento

2.1 L'intelligenza artificiale

Definizione di intelligenza artificiale.

L'intelligenza artificiale (Artificial Intelligence, AI) consiste nello studio di teorie e tecniche volte allo sviluppo di sistemi informatici in grado di svolgere compiti che normalmente necessitano dell'intelligenza umana [30]. La vita quotidiana di un individuo richiede un'immensa conoscenza del mondo. Gran parte di questa conoscenza è soggettiva ed intuitiva, e quindi difficile da articolare in modo formale [31]. Il tipico esempio che si fa in questo caso è la classificazione cane-gatto: già da bambini siamo in grado di estrapolare le caratteristiche dell'una e dell'altra classe, riconoscendo correttamente i due animali; tuttavia, insegnare questo semplice compito ad un computer ha richiesto decenni di studi [31][32]. Per fare ciò, infatti, gli algoritmi tradizionali, basati cioè sull'esecuzione consecutiva di istruzioni formulate ad hoc, non sono sufficienti: per quanto si possano aggiungere nuove variabili e condizioni all'algoritmo, non si è in grado di tradurre in una serie di comandi tutte le caratteristiche che distinguono un cane da un gatto. Ancora più arduo sarebbe poi generalizzare lo stesso compito all'intero regno animale [32]. I computer devono essere quindi in grado di catturare le stesse conoscenze che noi estrapoliamo dal mondo per comportarsi in modo intelligente; è per questa ragione che si parla di intelligenza artificiale [31].

Cenni storici.

Storicamente, la nascita dell'AI come campo di studio si fa coincidere con una rivoluzionaria conferenza del 1956 presso il Dartmouth College [33], dove professori e studenti esposero i propri risultati definendoli "stupefacenti": i computer erano in grado di apprendere le strategie degli scacchisti, risolvere complessi problemi algebrici letterali, dimostrare teoremi logici e perfino imparare a parlare l'inglese [30].

Nel quarantennio seguente, l'AI conobbe ondate di straordinario successo, seguite poi da periodi di quasi totale inattività, definiti successivamente "inverni AI" [33]; fu necessario infatti aspettare la fine degli anni '90 e l'inizio del nuovo secolo per osservare le prime applicazioni pratiche dell'AI al di fuori della ricerca accademica, con implementazioni nella logistica, nella scienza dei dati, nella diagnostica medica e in molte altre aree [30]. Questo successo è dovuto principalmente all'aumento esponenziale del potere computazionale (come predetto dalla legge di Moore), che rese finalmente possibile ai computer elaborare la quantità di dati necessaria ad ottenere risultati soddisfacenti in tempistiche accettabili [30].

Lo stato dell'arte.

Dagli anni dei primi importanti successi, l'impiego dell'AI è stato sempre più pervasivo: nel 2015 Google (Google LLC, Mountain View, California, Stati Uniti) ha dichiarato di avere all'attivo più di 2700 progetti di AI [34]; nel 2017 Apple (Apple Inc., Cupertino, California, Stati Uniti) ha rilasciato sul mercato il primo smartphone con Face ID, un algoritmo di riconoscimento facciale basato su AI [35]; nel 2018 NVIDIA (NVIDIA Corporation, Santa Clara, California, Stati Uniti) ha reso disponibili le prime schede video dotate di Tensor Core, unità computazionali specializzate nel calcolo tensoriale per accelerare gli algoritmi di AI sia in campo scientifico [36] che videoludico [37], permettendo così di implementare una nuova tecnologia di anti-aliasing video, il DLSS (Deep Learning Super Sampling), che basa il suo funzionamento sull'AI [37].

Ad oggi dunque, "intelligenza artificiale" è senza dubbio il binomio attorno al quale gravita la maggior parte delle nuove tecnologie, della ricerca e del marketing delle grandi aziende, nonché delle speranze per le future rivoluzioni scientifico-tecniche.

2.2 Approcci di intelligenza artificiale

Sistemi basati su regole.
Nell'oltre mezzo secolo del suo sviluppo, si sono susseguiti differenti approcci all'AI. I primi sistemi di AI hanno cercato di descrivere il mondo tramite l'utilizzo di linguaggi formali, grazie ai quali un computer può ragionare automaticamente usando le regole dell'inferenza logica. Sistemi di questo tipo sono detti **sistemi basati sulla conoscenza** (Knowledge Based Systems, KBS), oppure **sistemi basati su regole** (Rule Based System, RBS) se tale conoscenza viene memorizzata sotto forma di un insieme di regole [31]. Tuttavia, nessuno di questi progetti ha avuto successo: è già stato commentato in sezione 2.1 quanto questi approcci, del tutto assimilabili agli algoritmi tradizionali, non siano sufficienti a dotare le macchine dell'intelligenza necessaria neppure a distinguere un cane da un gatto [32].

Machine learning.
Il fallimento di tali approcci ha suggerito che i sistemi di AI dovessero acquisire la propria conoscenza direttamente dai dati attraverso un processo di apprendimento (**Machine Learning, ML**). In particolare, si può parlare di ML quando è rispettata la seguente definizione:

> **Definizione** (di programma di machine learning).
> *Un programma si dice che apprende con l'esperienza E rispetto ad un task assegnato T ed una metrica di prestazione P se la sua prestazione nel task T misurata dalla metrica P migliora con l'esperienza E* [38].

Riassumendo, si può dire che **un algoritmo di ML è tale se esso è in grado di apprendere dai dati**. Si vede ora nello specifico una descrizione di T, P ed E, utile anche per introdurre il lessico dell'ML.

- **Il Task *T*.** Nello specifico di questa tesi, il task T che si desidera che la macchina sappia risolvere è un **problema di classificazione**: il computer deve cioè accettare in ingresso una determinata rappresentazione dei dati a disposizione (esempio: il tracciato EEG di un soggetto) ed **etichettare** correttamente in uscita tali dati, scegliendo una delle categorie possibili (esempio: movimento della mano destra rispetto alla sinistra).

- **La metrica di performance *P*.** La prestazione di un modello di Machine Learning è misurata da una metrica di performance P [38]. Questa deve essere scelta ad hoc sulla base del task T da risolvere e, nel caso della classificazione, può essere per esempio l'**accuratezza**, che indica la frazione di esempi correttamente classificati (equazione 2.1):

$$\text{accuratezza} = \frac{\text{n. esempi correttamente classificati}}{\text{n. esempi classificati}} \tag{2.1}$$

- **L'esperienza *E*.** Gli algoritmi di ML apprendono con l'esperienza su una collezione di dati, chiamata **dataset**. Si distingue quindi fra **machine learning supervised** (supervisionato) e **machine learning unsupervised** (non supervisionato) sulla base di come l'algoritmo accumula l'esperienza E sul dataset durante il processo di apprendimento [31].

  - **ML supervisionato.** La macchina può apprendere la funzione desiderata ingresso-uscita (ovvero imparare a predire l'etichetta corretta) cercando di ridurre il grado di disaccordo tra l'etichetta predetta e quella vera. Questo è reso possibile grazie alla disponibilità nel dataset delle etichette per ciascun ingresso (dataset etichettato) e quindi il termine "supervisionato" origina dalla **presenza di un maestro** che fornisce alla macchina l'uscita corretta per un determinato ingresso [31].
  - **ML non supervisionato.** Quando il dataset non è etichettato, la macchina può limitarsi ad apprendere delle proprietà circa la struttura stessa del dataset. Ad esempio, alcuni algoritmi di ML non supervisionato sono in grado di operare il clustering, che consiste nel suddividere il dataset in gruppi (cluster) di esempi simili [31]. Tuttavia in questo caso, in mancanza di etichette, non è possibile valutarne le performance [30].

Da qui in avanti, quando si parlerà di ML, si intenderà un algoritmo di classificazione (task di classificazione, apprendimento supervisionato) che è l'oggetto di questo elaborato.

Representation learning.

Le prestazioni degli algoritmi di ML dipendono in gran parte dalla rappresentazione dei dati che vengono forniti. Ad esempio, quando si utilizza un classificatore per raccomandare il parto cesareo, il sistema AI non esamina direttamente la paziente, ma è necessario che il medico specifichi alla macchina tutte le informazioni rilevanti, come la presenza o l'assenza di una cicatrice uterina. Ogni pezzo di informazione incluso nella rappresentazione della paziente è noto come **feature (caratteristica)**. L'algoritmo di ML apprende come ognuna di queste caratteristiche della paziente sia correlata a diversi esiti, ma non può influenzare il modo in cui sono definite, essendo decise a priori dal medico. Se infatti si volesse fornire al classificatore una immagine di risonanza magnetica della paziente invece che il rapporto formalizzato dal medico, esso non sarebbe assolutamente in grado di fare previsioni utili; infatti, i pixel individuali in una risonanza hanno una correlazione trascurabile con qualsiasi complicazione che potrebbe verificarsi durante il parto [31]. Molte attività di AI possono essere risolte progettando il giusto set di feature da estrarre per quel task e da presentare all'algoritmo di ML (si parla infatti di **estrazione delle feature**). Tuttavia, per diversi task risulta complicato sapere quale feature dovrebbe essere estratta. Una soluzione a questo problema è usare l'ML non solo per scoprire la mappatura dalle feature all'output, ma anche la rappresentazione delle feature stesse. Questo approccio è noto come **Representation Learning (RL)** [31].

Deep learning.

Una delle difficoltà maggiori in molte applicazioni di AI è l'estrema variabilità dei fattori che condizionano il singolo dato che si è in grado di osservare. Per esempio l'immagine di una macchina rossa può sembrare più scura durante la notte, oppure potrebbe avere una sagoma completamente differente cambiando l'angolo di visione (si vedano altri esempi in figura 2.1). La maggior parte delle applicazioni di AI ci richiede di riuscire a separare i fattori di variabilità e scartare quelli che non ci interessano. Tale compito è estremamente complesso: molti di questi fattori di variazione, come l'accento nel parlato, possono essere identificati solo avendo una comprensione dei dati sofisticata, quasi a livello umano. Quando è così difficile ottenere una rappresentazione che risolva il problema originale, l'RL non sembra, a prima vista, ottenere risultati soddisfacenti [31].

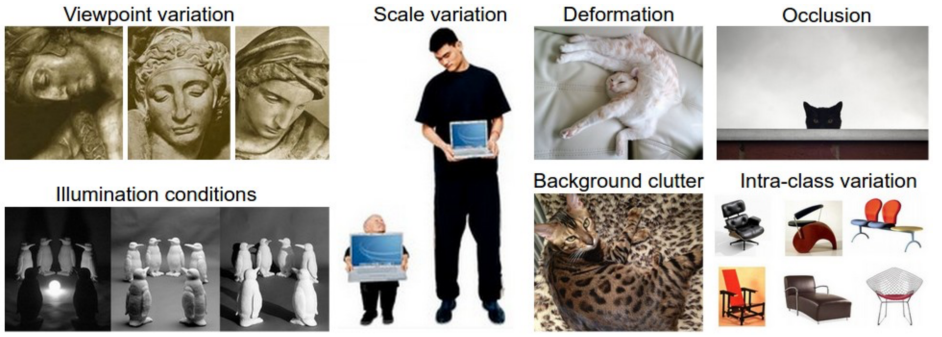


Figura 2.1: esempio dei possibili fattori di variabilità nella classificazione di immagini [32].

Il **Deep Learning (DL)** risolve queste problematiche rappresentando il mondo esterno attraverso una gerarchia di concetti, in cui ogni concetto è definito in funzione di altri concetti più semplici. I modelli matematici in grado di compiere tali semplificazioni si ottengono a partire da semplici **reti neurali feed-forward**, andando ad aggiungere nuovi strati di neuroni fra lo strato di input e di output; gli strati intermedi sono detti **strati nascosti** (hidden layers) e ci si riferisce ai modelli così ottenuti come **modelli profondi**, da cui appunto l'aggettivo "deep". Ad esempio, nella computer vision (insieme di tecniche che mirano a rendere i computer in grado di comprendere le immagini) i primi strati estraggono i bordi dell'immagine, quindi gli angoli e i contorni degli oggetti, dunque porzioni di essi, confluendo poi sui neuroni di uscita con la reale classificazione. I concetti appresi, quindi, hanno una complessità crescente lungo la rete neurale [31].

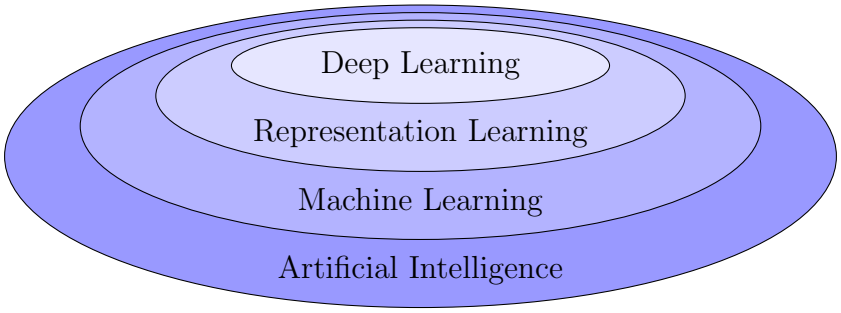


Figura 2.2: diagramma di Venn che mostra come il deep learning sia una tipologia di representation learning, che a sua volta è una tipologia di machine learning, che è l'approccio più utilizzato fra i sistemi di intelligenza artificiale [31].

2.3 Fondamenti di machine learning

Training set, test set e validation set.
Quando si addestra un classificatore, lo si fa con l'obbiettivo di ottenere la miglior performance possibile con gli esempi a disposizione, in modo poi da poter utilizzare tale modello per la predizione con input nuovi e mai visti prima. Il dataset viene dunque suddiviso in un sottoinsieme di addestramento, detto **training set**, e in un sottoinsieme di test, detto **test set**. Inoltre, come si vedrà nel corso della trattazione, si farà anche uso di una ulteriore suddivisione del training set detta **validation set**, utile per testare anche l'influenza di singoli parametri del classificatore durante l'addestramento [31].

Generalizzazione, overfitting, underfitting e capacità.
La capacità di classificare correttamente esempi precedentemente non osservati è detta **generalizzazione**. Dal momento che il test set è formato da esempi che il classificatore non ha avuto modo si osservare durante l'addestramento, allora l'errore di generalizzazione coincide con l'errore commesso dal classificatore nell'etichettare il test set [31].

Detto questo, i fattori che determinano il buon funzionamento di un algoritmo di ML sono la sua capacità di: **(1) minimizzare l'errore di training** e **(2) minimizzare il divario tra l'errore di training e l'errore di test** (o generalizzazione). Si faccia ora riferimento a figura 2.3: quando un classificatore riesce ad operare entrambe queste minimizzazioni, si trova nel suo punto di lavoro ideale, indicato dalla linea rossa verticale. Spostandosi alla sinistra ci si trova in **regime di underfitting**, che si verifica quando il modello non è in grado di ottenere un errore sufficientemente basso sul training set; alla destra invece si è in **regime di overfitting**, ovvero quando il divario tra l'errore training e di test è troppo grande [31].

Siamo in grado di controllare se un modello ha maggiori probabilità di essere in regione di overfitting o di underfitting modificando la sua capacità. **La capacità di un modello** può essere definita come la sua capacità di descrivere un'ampia varietà di funzioni. I modelli con una capacità bassa hanno difficoltà ad adattarsi al training set (andranno cioè a lavorare prevalentemente in regione di underfitting); al contrario, i modelli con un'elevata capacità possono "imparare a memoria" le proprietà del training set, con il rischio di non utilizzare correttamente questa conoscenza sul test set [31].

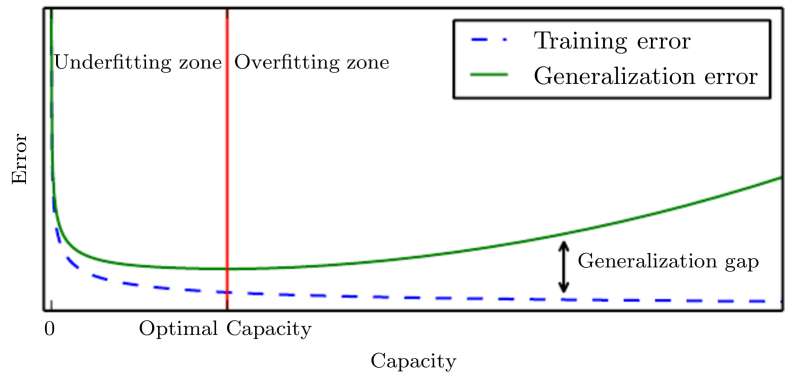


Figura 2.3: relazione tipica tra capacità ed errore. L'errore di training e l'errore di test (che come si è detto misura l'errore di generalizzazione) si comportano diversamente. All'estremità sinistra del grafico, l'errore di training e quello di generalizzazione sono entrambi alti: ci si trova dunque in regime di underfitting. Aumentando la capacità, l'errore di training diminuisce, ma aumenta il divario con l'errore di generalizzazione, entrando dunque in regime di overfitting [31].

Regolarizzazione.

Quando un modello possiede una scarsa generalizzazione (tende cioè all'overfitting), è possibile utilizzare tecniche di regolarizzazione per modificare l'algoritmo affinché si riduca l'errore di generalizzazione mantenendo però invariato l'errore sul set di addestramento. Queste tecniche ottengono il risultato desiderato esprimendo delle preferenze sulle funzioni che il modello deve descrivere, limitando quindi la capacità complessiva del modello originario e spostando il punto di lavoro del modello dalla regione di overfitting verso il punto di lavoro ottimale [31].

Gli iper-parametri.

Il comportamento della maggior parte degli algoritmi di ML può essere controllato andando a modificare particolari parametri; questi, dal momento che influenzano spesso in modo significativo l'apprendimento del modello, vengono detti iper-parametri (ad esempio nel caso delle reti neurali profonde sono iper-parametri il numero di neuroni per strato, le funzioni di attivazione usate, il numero di epoche di addestramento, ecc.). Gli iper-parametri definiscono uno spazio, detto spazio degli iper-parametri (di dimensionalità elevata nel caso delle reti neurali, poiché esistono molti iper-parametri) e per ogni possibile valore di questi si definisce un punto corrispondente alla configurazione del modello scelto.

Un modello di ML è rappresentato da parametri interni che vengono modificati con l'avanzare dell'addestramento, detti parametri addestrabili (ad esempio, sempre guardando alle reti neurali, sono parametri addestrabili i pesi che connettono due strati successivi). Questi parametri vengono addestrati sfruttando il training set. Non è opportuno invece addestrare gli iper-parametri sullo stesso set di addestramento: infatti, se si suppone che esista un iper-parametro tramite il quale la capacità del sistema viene modificata, viene da sé che addestrandolo sul training set esso tenderà ad assumere valori per i quali la capacità del modello è massimizzata; di conseguenza, tenderebbe a portare il modello in regione di overfitting [31]. Infine, è scorretto settare i valori degli iper-parametri basandosi sui dati di test poiché questi dati devono essere accessibili solo nell'ultima fase di valutazione della capacità di generalizzazione del modello e non devono essere in alcun modo impiegati per selezionare il modello.

Quindi, per il **tuning degli iper-parametri** è opportuno allora utilizzare un set di dati separato da quello di training e da quello di test, ovvero l'**insieme di validazione** (validation set). In genere, si utilizza una percentuale del training set variabile dal 10-20% come validation set e la percentuale rimanente viene riassegnata al training set. Infine, una volta completata l'ottimizzazione dell'iper-parametro, l'errore di generalizzazione può essere stimato utilizzando il test set [31].

La cross-validazione.

Dividere il dataset in un training set e un test set fissi può essere problematico se il set di test risulta ridotto. Un piccolo test set implica incertezza statistica circa l'errore medio stimato, rendendo difficile affermare che l'algoritmo A funzioni meglio dell'algoritmo B sul task per il quale sono stati addestrati [31]. Avendo a che fare con dataset ridotti, si possono adoperare procedure alternative che consentono di utilizzare tutti gli esempi nella stima dell'errore di test medio, al prezzo di un aumento del costo computazionale. Queste procedure si basano sull'idea di ripetere l'addestramento ed il successivo test su diverse suddivisioni del dataset originale. La più comune di queste è la **procedura di cross-validation a *k*-fold**, in cui si vanno a formare k partizioni diverse del dataset, dove ogni volta il test set è rappresentato da esempi differenti (si veda figura 2.4). L'errore sul test set può quindi essere stimato come l'errore di test medio tra k prove [31].

Una particolare procedura che sarà utilizzata in questa tesi è la **cross-validazione stratificata**. In questo caso, si vanno a creare le partizioni in funzione delle etichette degli esempi di test e ci si assicura che i test set di ogni fold contengano un numero bilanciato di esempi appartenenti alle varie classi. In questo modo è possibile creare partizioni più bilanciate e raggiungere prestazioni in generale migliori [31].

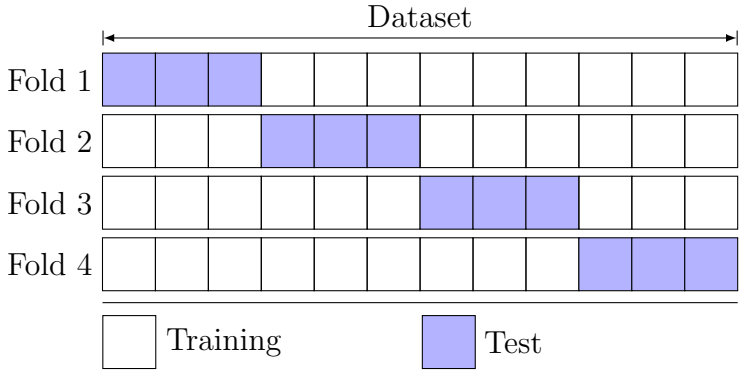


Figura 2.4: esempio suddivisione di un dataset composto da soli 12 esempi per cross-validazione a 4 fold. Ogni fold sarà allora composta da un test set da 3 esempi, mentre il resto dei dati comporrà il training set. Nel caso in cui si voglia utilizzare anche un validation set, di solito si vanno a selezionare per la validazione gli ultimi esempi che compongono il training set [31].

Una visione d'insieme.

Cioè che è stato esposto fin ad ora a proposito della terminologia, delle tecniche di base e dei parametri che descrivono un algoritmo di ML è valido anche per il DL, essendo quest'ultimo un sotto-insieme del primo. Ciò che differenzia il DL da un normale algoritmo di ML è che, lo ricordiamo, il DL è una tecnica di RL, ed è quindi in grado di estrarre anche le caratteristiche con le quali si effettua la classificazione.

Di qui in avanti per tutto il resto del Capitolo 2 si parlerà di ML e, nello specifico, del Filter Bank Common Spatial Pattern (FBCSP) con classificazione tramite Linear Discriminant Analysis (LDA), l'attuale gold standard per le applicazioni di BCI movement-based [27]. L'FBCSP è una tecnica di estrazione delle caratteristiche, che verrà esposta in sezione 2.4. L'LDA è invece un algoritmo di classificazione, esposto a seguire in sezione 2.5.

2.4 Estrazione delle feature per la decodifica del movimento

Introduzione.

Nel primo capitolo di questa tesi, si è avuto modo di osservare come il moto volontario provochi un'attenuazione o un incremento dell'attività ritmica neurale in determinate bande frequenziali; tali fluttuazioni dei biopotenziali cerebrali sono registrate tramite l'elettroencefalografo ed è quindi possibile applicare al segnale prodotto diversi algoritmi di classificazione basati sull'apprendimento automatico [39][40][41].

Ad oggi, il metodo maggiormente impiegato per l'estrazione delle feature è il **Filter Bank Common Spatial Pattern (FBCSP)** [4]. Esso si basa sui correlati funzionali precedentemente analizzati (in particolare ERD/ERS [42]) e si compone di due macro-blocchi in cascata:

1. **Blocco dei filtraggi temporali (parte FB).** Vengono applicati al segnale EEG svariati filtraggi di tipo passa-banda, andando ad ottenere diverse versioni del segnale originale, ognuna delle quali contenente le sole informazioni armoniche della banda associata al filtro applicato; per fare ciò, si va a progettare un **banco di filtri**, ovvero un insieme di filtri digitali con banda passante di una decina di Hertz che coprono tutto lo spettro del segnale, da poco sopra la continua fino alla frequenza di Nyquist (da qui la prima parte del nome, "Filter Bank") [39][43].

2. **Blocco dei filtraggi spaziali (parte CSP).** Gli eventi ritmici cerebrali di interesse per la classificazione del movimento avvengono principalmente nei pressi dell'area motoria e sono dunque registrasti da un certo gruppo di elettrodi. Fra questi, i canali più prossimi all'evento capteranno inevitabilmente un segnale più informativo, mentre il resto dell'informazione è distribuita spazialmente nell'intorno [42]. Tramite filtri spaziali creati ad hoc, è possibile convogliare la maggior parte di questa informazione all'interno di nuovi segnali tramite il metodo dei Common Spatial Patterns (da cui la seconda parte del nome, CSP) [44].

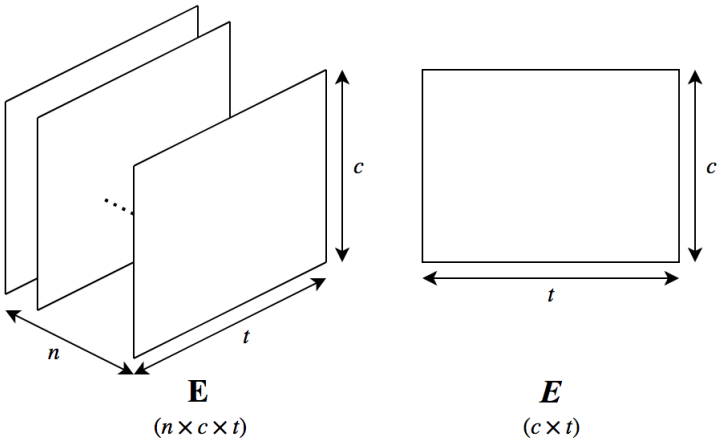


Figura 2.5: rappresentazione grafica di un generico dataset EEG. Per ogni soggetto, si può pensare al dataset come ad un tensore **E** (matrice multi-dimensionale) di dimensione $c \times t \times n$, con n numero di trial, c numero di canali e t numero di campioni temporali. Ci si riferisce invece ai dati del singolo trial con la notazione $\boldsymbol{E}$, che sarà dunque una semplice matrice $c \times t$.

Filtraggi temporali.

Il primo step dell'estrazione delle feature è il filtraggio temporale tramite l'applicazione, per ogni trial $\boldsymbol{E}$ componente il dataset, dell'intero banco di filtri. Se si è progettato un banco composto da b filtri passa-banda differenti, allora si otterranno in uscita b versioni diverse del segnale originale, dove la k-esima uscita $\boldsymbol{E}_{(k)}$ avrà banda limitata dal filtro k-esimo corrispondente [4].

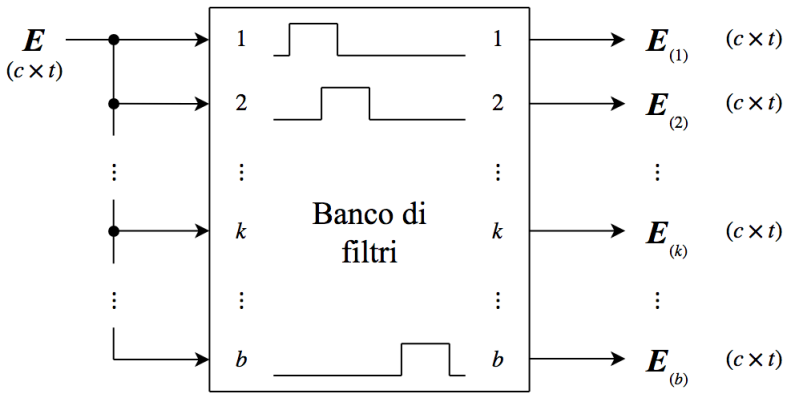


Figura 2.6: diagramma a blocchi input/output per un banco di filtri formato da b passa-banda differenti.

Quando si progetta un banco di filtri, è possibile variare i seguenti parametri: **(1) tipologia di filtro digitale**; **(2) ordine del filtro**; **(3) frequenza minima** del banco; **(4) frequenza massima** del banco; **(5) larghezza della banda passante**; **(6) larghezza della banda di overlap** fra due filtri successivi [4]. A titolo d'esempio, si consideri il banco di filtri utilizzato in questa tesi (si veda Capitolo 4); tale banco è stato ottenuto affiancando due banchi separati, entrambi formati con filtri di Butterworth del terzo ordine. Il primo banco si estende da 0.5 Hz fino ai 12 Hz, con larghezza di banda passante di 6 Hz ed overlap di 3 Hz; il secondo banco si estende invece dai 10 Hz ai 122 Hz, con larghezza di banda passante di 8 Hz ed overlap di 4 Hz. Il banco risultante è allora composto dalle seguenti bande in ordine crescente: 0.5-6 Hz, 3-9 Hz, 6-12 Hz, 10-18 Hz, 14-22 Hz, e così via sino a 114-122 Hz, per un totale di 30 filtri.

Filtraggi spaziali.
L'estrazione delle feature tramite Common Spatial Patterns è senza dubbio uno degli approcci più popolari in contesti BCI movement-based [45]. Tale algoritmo viene applicato a due classi per volta, calcolando i filtri spaziali che massimizzano la varianza di una classe e minimizzano la varianza dell'altra, e viceversa [45].

Prima di procedere con i passaggi matematici per il calcolo di tali filtri spaziali, è doveroso riprendere alcuni concetti matematici importanti:

- **Teorema spettrale.** Sia $\boldsymbol{A}$ una matrice quadrata $n \times n$; se essa è (1) simmetrica o (2) ammette esattamente n autovalori distinti, allora $\boldsymbol{A}$ è diagonalizzabile (equazione (2.2)):

$$\boldsymbol{\Lambda} = \boldsymbol{\Phi}^{-1}\boldsymbol{A}\boldsymbol{\Phi} \iff \boldsymbol{A} = \boldsymbol{\Phi}\boldsymbol{\Lambda}\boldsymbol{\Phi}^{-1} \tag{2.2}$$

  con $\boldsymbol{\Phi} = [\boldsymbol{\varphi}_1, \boldsymbol{\varphi}_2, \ldots, \boldsymbol{\varphi}_n]$ la matrice che ha come colonne gli autovettori $\boldsymbol{\varphi}_i$ di $\boldsymbol{A}$ e $\boldsymbol{\Lambda}$ la matrice diagonale che ha sulla diagonale principale gli autovalori λ_i di $\boldsymbol{A}$ [46].

- **Trasformazione di sbiancamento.** Sia $\underline{\boldsymbol{X}} = \{\underline{x}_{ij}\}$ una matrice $n \times m$ di variabili aleatorie correlate a valor medio nullo. Si definisce allora trasformazione di sbiancamento (o di whitening) una trasformazione lineare $\underline{\boldsymbol{Y}} = \boldsymbol{Q}\underline{\boldsymbol{X}}$ che trasforma $\underline{\boldsymbol{X}}$ in un set di n variabili $\underline{\boldsymbol{Y}}$ scorrelate e tutte a varianza unitaria, dove $\boldsymbol{Q}$ è una matrice $n \times n$ detta matrice di sbiancamento [47].

Sia dunque $\boldsymbol{E}$ la matrice $c \times t$ associata ad un generico trial. Si definisce **matrice di covarianza normalizzata** $\boldsymbol{\Sigma}$ la quantità (equazione 2.3) [44][45]:

$$\boldsymbol{\Sigma} = \frac{\boldsymbol{E}\boldsymbol{E}^T}{\mathrm{tr}(\boldsymbol{E}\boldsymbol{E}^T)}. \tag{2.3}$$

Se il training set contiene n_j trial per la classe ω_j, la matrice di covarianza $\boldsymbol{C}$ (normalmente calcolata, per variabili aleatorie opportunamente centrate, usando il valor medio $\boldsymbol{C} = \mathbb{E}[\underline{\boldsymbol{X}}\underline{\boldsymbol{X}}^T]$) può essere stimata per la j-esima classe tramite la media campionaria delle singole matrici di covarianza normalizzate (equazione (2.4)) [44][45]:

$$\boldsymbol{C}_j = \frac{1}{n_j} \sum_{\substack{i=1 \\ \boldsymbol{\Sigma} \in \Omega_j}}^{n_j} \boldsymbol{\Sigma}_i \tag{2.4}$$

dove Ω_j è l'insieme di tutti gli esempi $\boldsymbol{E}$ appartenenti alla classe j-esima ω_j. Si definisce inoltre **matrice di covarianza spaziale composita** $\boldsymbol{C}_c$ la quantità data dalla somma delle matrici di covarianza delle due classi che vogliamo separare (equazione (2.5)) [44]:

$$\boldsymbol{C}_c = \boldsymbol{C}_1 + \boldsymbol{C}_2. \tag{2.5}$$

Dal momento che la matrice di covarianza è simmetrica per costruzione, anche la matrice di covarianza composita $\boldsymbol{C}_c$ lo è. Valendo il teorema spettrale, è dunque certo che esista una fattorizzazione diagonalizzante per $\boldsymbol{C}_c$, a cui associamo le due matrici $\boldsymbol{\Phi}$ e $\boldsymbol{\Lambda}$. Una possibile trasformazione di sbiancamento è quella che fa uso proprio di tali matrici spettrali per la definizione di $\boldsymbol{Q}$ (equazione (2.6)) [44][47]:

$$\boldsymbol{Q} = \boldsymbol{\Lambda}^{-\frac{1}{2}} \boldsymbol{\Phi}^T. \tag{2.6}$$

A questo punto, è facile dimostrare che la fattorizzazione $\boldsymbol{Q}\boldsymbol{C}_c\boldsymbol{Q}^T$ corrisponde all'identità (equazione (2.7)):

$$\begin{aligned} \boldsymbol{Q}\boldsymbol{C}_c\boldsymbol{Q}^T &= \boldsymbol{\Lambda}^{-\frac{1}{2}} \boldsymbol{\Phi}^T \underbrace{\boldsymbol{C}_c \boldsymbol{\Phi}}_{\boldsymbol{\Phi}\Lambda} \boldsymbol{\Lambda}^{-\frac{1}{2}} = \boldsymbol{\Lambda}^{-\frac{1}{2}} \underbrace{\boldsymbol{\Phi}^T \boldsymbol{\Phi}}_{I} \boldsymbol{\Lambda}\boldsymbol{\Lambda}^{-\frac{1}{2}} = \\ &= \boldsymbol{\Lambda}^{-\frac{1}{2}} \boldsymbol{\Lambda}\boldsymbol{\Lambda}^{-\frac{1}{2}} = \boldsymbol{I}. \end{aligned} \tag{2.7}$$

Dunque, mettendo a sistema tale fattorizzazione con la definizione di matrice di coovarianza spaziale composita (2.5), è possibile isolare due nuove matrici $\boldsymbol{S}_1$ ed $\boldsymbol{S}_2$ in funzione di $\boldsymbol{C}_1$ e $\boldsymbol{C}_2$ (equazione (2.8)):

$$\begin{cases} \boldsymbol{Q}\boldsymbol{C}_c\boldsymbol{Q}^T = \boldsymbol{I} \\ \boldsymbol{C}_c = \boldsymbol{C}_1 + \boldsymbol{C}_2 \end{cases} \implies \boldsymbol{Q}(\boldsymbol{C}_1 + \boldsymbol{C}_2)\boldsymbol{Q}^T = \boldsymbol{I}$$

$$\underbrace{\boldsymbol{Q}\boldsymbol{C}_1\boldsymbol{Q}^T}_{\boldsymbol{S}_1} + \underbrace{\boldsymbol{Q}\boldsymbol{C}_2\boldsymbol{Q}^T}_{\boldsymbol{S}_2} = \boldsymbol{I}. \tag{2.8}$$

Le nuove matrici così definite condividono gli stessi autovettori, ma hanno autovalori diversi [44]. Siano $\boldsymbol{\Psi}$ la matrice degli autovettori e $\boldsymbol{\Lambda}_1$, $\boldsymbol{\Lambda}_2$ le matrici diagonali degli autovalori; possiamo trovare una relazione fra quest'ultime due partendo dalla (2.8) e fattorizzando $\boldsymbol{S}_1$ ed $\boldsymbol{S}_2$ con l'ausilio del teorema spettrale (equazione 2.9):

$$\begin{cases} \boldsymbol{S}_1 + \boldsymbol{S}_2 = \boldsymbol{I} \\ \boldsymbol{S}_1 = \boldsymbol{\Psi}\boldsymbol{\Lambda}_1\boldsymbol{\Psi}^T \\ \boldsymbol{S}_2 = \boldsymbol{\Psi}\boldsymbol{\Lambda}_2\boldsymbol{\Psi}^T \end{cases} \implies \boldsymbol{\Psi}(\boldsymbol{\Lambda}_1 + \boldsymbol{\Lambda}_2)\boldsymbol{\Psi}^T = \boldsymbol{I}$$

$$\boldsymbol{\Lambda}_1 + \boldsymbol{\Lambda}_2 = \boldsymbol{I}. \tag{2.9}$$

Si è dunque ottenuto l'importante risultato per cui la somma di due autovalori corrispondenti è sempre unitaria [44]. Ciò comporta che l'autovettore associato all'autovalore massimo in $\boldsymbol{S}_1$ è associato al minimo in $\boldsymbol{S}_2$, e viceversa [44]. Tale proprietà consente di utilizzare gli autovettori $\boldsymbol{\Psi}$ per separare le due classi: la proiezione del segnale EEG sbiancato ($\boldsymbol{Q}\boldsymbol{E}$) lungo le direzioni individuate dalle colonne di $\boldsymbol{\Psi}$ (gli autovettori del nuovo spazio) restituisce i migliori vettori delle feature $\boldsymbol{Z}$ per discriminare spazialmente fra le due classi in esame (equazione (2.10)) [44]:

$$\boldsymbol{Z} = \boldsymbol{W}^T\boldsymbol{E}, \quad \text{con} \quad \boldsymbol{W} = \boldsymbol{\Psi}^T\boldsymbol{Q} \tag{2.10}$$

dove $\boldsymbol{W}$ è detta **matrice di proiezione totale** [44]. Le colonne di tale matrice sono i Common Spatial Patterns propriamente detti e possono essere pensati come i vettori tempo-invarianti della distribuzione spaziale delle sorgenti EEG [44].

In conclusione, è bene notare che questo algoritmo utilizza le etichette per suddividere i trial nelle due popolazioni; ciò non è un problema durante il training, dove è necessario conoscere le reali classi per guidare l'addestramento; passando al test invece, non è consentito all'algoritmo osservare le etichette, perciò è necessario utilizzare la stessa matrice $\boldsymbol{W}$ ottenuta per il training anche in questa fase [4].

L'estrazione delle feature.

Al fine di ridurre ulteriormente la dimensione dello spazio delle feature, molto spesso non si utilizza l'intera matrice di proiezione totale, ma si vanno ad isolare solamente m colonne; nello specifico, in seguito al calcolo di $\boldsymbol{W}$ si pongono le colonne in ordine decrescente rispetto agli autovalori associati ad $\boldsymbol{S}_1$; in questo modo, per quanto ricavato nell'equazione (2.9), le prime $m/2$ colonne saranno i pattern spaziali principali per la prima classe, viceversa per le ultime $m/2$ [4][44].

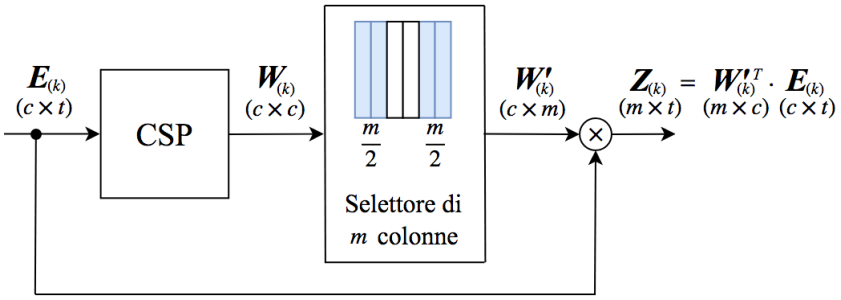


Figura 2.7: applicazione dell'algoritmo CSP sul k-esimo filtrato $\boldsymbol{E}_{(k)}$. La matrice di proiezione totale $\boldsymbol{W}_{(k)}$ ad esso associata è ridotta a $\boldsymbol{W}'_{(k)}$ selezionando solamente le m colonne più significative; quindi si proietta il segnale di partenza $\boldsymbol{E}_{(k)}$ nel nuovo spazio individuato dagli m pattern spaziali ricavati, ottenendo $\boldsymbol{Z}_{(k)}$.

Sul segnale in uscita dal blocco precedente viene poi calcolata la varianza temporale di ogni riga, riducendo enormemente la dimensionalità del problema, che passa da una rappresentazione matriciale ad una vettoriale [44][45]. Infine, per approssimare una distribuzione normale dei dati, si calcola il logaritmo della varianza normalizzata rispetto alla media [44], come si osserva in figura 2.8.

Per ogni filtro del banco si ottengono allora m feature, per un totale di $b \times m$ feature scalari totali. È poi possibile ridurre ulteriormente tale numero andando a compiere una selezione anche sui filtri [4].

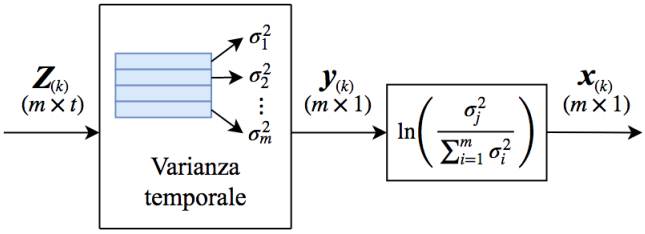


Figura 2.8: calcolo delle feature $\boldsymbol{x}_{(k)}$ a partire da $\boldsymbol{Z}_{(k)}$.

2.5 Classificatori per la decodifica del movimento

Classificazione e mapping dei dati.

L'estrazione delle caratteristiche, come si è visto, produce una nuova rappresentazione di un esempio del dataset originario. Ripetendo l'estrazione delle feature per ciascun esempio a disposizione, si va a rimappare lo spazio degli esempi in uno nuovo, detto **spazio dei pattern**, tramite i quali si implementa la classificazione.

A valle dell'operazione di estrazione delle feature si trova allora il vero e proprio classificatore. L'algoritmo di classificazione è in grado, in base ai pattern che riceve in ingresso e alle loro etichette, di imparare da essi, così da sfruttare la conoscenza appresa nelle successive fasi di test [31][48].

Per fare ciò, **il classificatore deve apprendere una qualche funzione capace di eseguire il mapping dallo spazio dei pattern allo spazio delle classi** [48]. Una pipeline di ML opera allora due differenti mappature dei dati: la prima dallo spazio degli esempi a quello dei pattern, la seconda dallo spazio dei pattern a quello delle classi.

Nel caso le classi possibili siano solamente due, il problema di classificazione è detto binario (si parla cioè di **binary classification**); con più di due classi da assegnare si parla invece di classificazione multi classe (**multi-class classification**) [31][48].

La Linear Discriminant Analysis.

Il letteratura, esistono svariati algoritmi di classificazione: fra i più noti si citano il classificatore bayesiano, il Nearest Neighbor, le Support Vector Machines e molti altri [31][48]. Di seguito ci si occuperà nello specifico del classificatore utilizzato in questa tesi, ovvero della **Linear Discriminant Analysis (LDA).**

Nonostante possa essere utilizzato come classificatore, la LDA nasce come algoritmo di riduzione della dimensionalità. In particolare, è in grado di operare una trasformazione lineare $\mathbb{R}^n \to \mathbb{R}^m$ (con $m < n$) in maniera supervisionata (osservando cioè sia gli ingressi $\boldsymbol{x}_i$ che le etichette y_i) andando a privilegiare quelle dimensioni che discriminano meglio i pattern in ingresso [48].

Si faccia ora riferimento all'uscita dell'algoritmo FBCSP analizzato nella sezione precedente. L'estrazione delle feature, a partire da ogni trial in ingresso, restituisce un vettore $\boldsymbol{x}_{(k)}$ per ogni k-esimo filtro del banco. Dal momento che il banco è formato da b filtri passa banda diversi, si saranno perciò estratte f feature, con $f = m \cdot b$.

Si consideri ora di riunire tutti gli f valori in un unico vettore $\boldsymbol{x}_i$. Questa operazione andrà ripetuta per ogni trial i-esimo che compone il dataset originario, formato da n esempi $\boldsymbol{E}_1, \boldsymbol{E}_2, \ldots, \boldsymbol{E}_n$. Si mapperà così lo spazio degli n esempi di ingresso in uno di n pattern $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$. Si supponga ora di voler addestrare un classificatore LDA in modo che sia in grado di classificare gli n pattern ricavati in ω classi diverse. Sarà necessario, per un algoritmo di ML con supervisore, conoscere le n etichette $y_1, y_2, \ldots, y_n$ degli n pattern. Tali etichette assoceranno perciò il pattern i-esimo $\boldsymbol{x}_i$ alla classe j-esima ω_j.

Sia n_j il numero di pattern che sono etichettati dal supervisore con la classe j-esima. È possibile calcolare il vettore medio dei pattern $\overline{\boldsymbol{x}}_j$ per la classe j-esima tramite la 2.11:

$$\overline{\boldsymbol{x}}_j = \frac{1}{n_j} \sum_{\substack{i=1 \\ \boldsymbol{x}_i \in \Omega_j}}^{n_j} \boldsymbol{x}_i \tag{2.11}$$

dove Ω_j è l'insieme degli esempi (e quindi dei pattern) appartenenti alla classe j-esima ω_j. Per formulare il **criterio di ottimizzazione di massima separazione tra le classi**, si definiscono due particolari matrici di covarianza, dette **matrici di scattering** [48]:

- **Matrice di scattering within-class $\boldsymbol{S}_w$.** Indica come i vettori sono dispersi rispetto al centro delle classi (ciascuno rispetto alla propria classe, equazione (2.12)):

$$\boldsymbol{S}_w = \sum_{j=1}^{\omega} \boldsymbol{S}_j, \quad \text{con:} \quad \boldsymbol{S}_j = \sum_{\substack{i=1 \\ \boldsymbol{x}_i \in \Omega_j}}^{n_j} (\boldsymbol{x}_i - \overline{\boldsymbol{x}}_j)(\boldsymbol{x}_i - \overline{\boldsymbol{x}}_j)^T. \tag{2.12}$$

- **Matrice di scattering between-class $\boldsymbol{S}_b$.** Indica come i centri delle classi sono dispersi rispetto al centro della distribuzione (ovvero quanto le classi sono disperse, equazione (2.13)):

$$\boldsymbol{S}_b = \sum_{j=1}^{\omega} n_j (\overline{\boldsymbol{x}}_j - \overline{\boldsymbol{x}}_0)(\overline{\boldsymbol{x}}_j - \overline{\boldsymbol{x}}_0)^T, \quad \text{con:} \quad \overline{\boldsymbol{x}}_0 = \frac{1}{n} \sum_{j=1}^{\omega} n_j \overline{\boldsymbol{x}}_j. \tag{2.13}$$

Tra i criteri di ottimizzazione possibili quello più frequentemente utilizzato è la massimizzazione della quantità j_1 (equazione 2.14) [48]:

$$j_1 = \mathrm{tr}(\boldsymbol{S}_w^{-1}\boldsymbol{S}_b) \tag{2.14}$$

Il criterio è intuitivo, in quanto cerca di massimizzare la dispersione tra le classi (quindi la matrice di scattering between-class $\boldsymbol{S}_b$), minimizzando al contempo la dispersione interna a ciascuna classe (quindi l'inversa della matrice di scattering within-class $\boldsymbol{S}_w^{-1}$) [48].

Si dimostra che, data la massimizzazione di j_1, il sottospazio LDA in $\mathbb{R}^m$ è individuato dagli autovettori associati ai primi m autovalori della matrice $\boldsymbol{S}_w^{-1}\boldsymbol{S}_b$; sia $\boldsymbol{V}$ la matrice $n \times m$ che ha come colonne tali m autovettori: la proiezione del pattern $\boldsymbol{x}_i$ dallo spazio di partenza allo spazio LDA sarà [47][48]:

$$\boldsymbol{x}_i' = \boldsymbol{V}^T\boldsymbol{x}_i \tag{2.15}$$

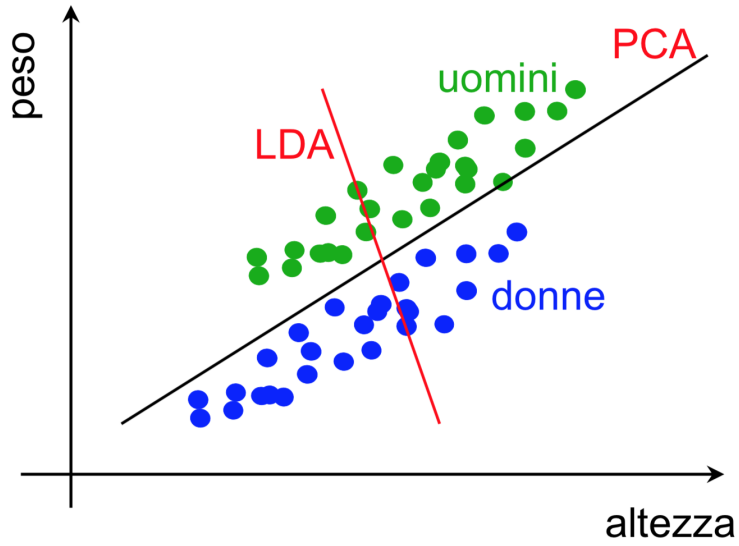


Figura 2.9: in figura sono confrontati due esempi di riduzione di dimensionalità da $\mathbb{R}^2$ ad $\mathbb{R}^1$: PCA (Principal Component Analysis) ed LDA. Il segmento nero, che identifica la soluzione PCA, è l'iperpiano sul quale, proiettando i pattern (indipendentemente dalla loro classe), si conserva al massimo l'informazione [47]. Il segmento rosso, che identifica invece la soluzione LDA, è l'iperpiano sul quale, proiettando i pattern, si è in grado di distinguere al meglio le due classi (pattern verdi contro pattern blu). Entrambi sono mapping lineari, ma la soluzione monodimensionale (retta) è profondamente diversa. Un'altra differenza fra i due algoritmi è che mentre l'LDA è supervisionata (utilizza le etichette per la massimizzazione di j_1), la PCA è non supervisionata [48].

Regolarizzazione tramite shrinkage LDA.

È possibile modificare l'algoritmo introducendo una regolarizzazione delle matrici di covarianza associate alle classi. Infatti, le matrici di covarianza stimate da pochi dati tendono ad avere autovalori che si discostano molto rispetto a quelli della vera distribuzione dei dati, portando ad una scarsa stima di tali matrici. Questo problema può essere risolto "contraendo" le matrici di covarianza (equazione (2.16)):

$$\hat{\boldsymbol{S}} = \boldsymbol{S} - \lambda \boldsymbol{I} \tag{2.16}$$

dove I è la matrice identità e λ è un parametro di regolarizzazione. Il metodo così definito prende il nome di shrinkage LDA (sLDA) ed in letteratura sono riportati risultati migliori rispetto alla standard LDA per applicazioni BCI [27]. Anche in questo elaborato di tesi si utilizza tale classificazione regolarizzata, a valle dell'estrazione delle feature operata da FBCSP.

LDA per classificatore binario.

Il classificatore più semplice che è possibile implementare sulla Linear Discriminant Analysis è quello binario che mappa lo spazio delle feature in un sottospazio monodimensionale ($m = 1$, fa uso quindi solamente del primo autovettore). Usando ora una notazione binaria per le classi (esistono solo la classe 0 e la classe 1), se $\boldsymbol{x}'_i$ è negativo la classe predetta sarà la classe 0, e viceversa per $\boldsymbol{x}'_i$ positivo [4].

LDA per classificatore multiclasse.

Per estendere la classificazione a più di due classi, è possibile utilizzare un approccio "pair-wise" in cui il classificatore viene addestrato per discriminare ogni paia di classi possibili (es. classe 1 vs. classe 2, classe 1 vs. classe 3, ecc.) e la classe finale viene scelta sulla base di uno schema di votazione maggioritario basato sulle etichette predette dai classificatori binari [49]. In questa tesi la classificazione è stata implementata su 4 classi, di conseguenza si è utilizzato l'approccio qui descritto.

Capitolo 3

Deep learning per la classificazione del movimento

3.1 Il modello di neurone

L'ispirazione biologica dell'apprendimento profondo.
Il cervello umano contiene circa 100 miliardi di neuroni, ciascuno dei quali è connesso mediamente con altri 1000 neuroni, formando un'intricata rete di almeno 10^{14} sinapsi [50]. Poiché questa complessa struttura è al momento l'unica conosciuta in grado di dar vita al comportamento intelligente [31], l'intelligenza artificiale ha cercato fin dalla sua nascita di produrre algoritmi in grado di apprendere dai dati basandosi sui processi biologici cerebrali: si parte allora dai singoli modelli neuronali, i quali opportunamente semplificati e raggruppati insieme formano uno strato; più strati sovrapposti e collegati danno vita ad una rete neurale; infine, come si è anticipato nel secondo capitolo, se la rete è formata da molteplici strati (in particolare più di tre), questa viene denominata "profonda" ed il suo utilizzo in un applicazioni di apprendimento automatico caratterizza il DL [31].

Il neurone biologico.
Il neurone è l'unità funzionale del sistema nervoso, generando i segnali elettrici che permettono al cervello di elaborare ed immagazzinare informazioni. La sua struttura può essere schematizzata in quattro parti: **(1) il soma**, il corpo cellulare contenente il nucleo; **(2) i dendriti**, fibre che ramificano dal soma per raccogliere gli input dai neuroni afferenti; **(3) l'assone**, unica fibra di trasmissione, si allontana dal soma

per portare ad altri neuroni (anche distanti) l'output; **(4) i terminali presinaptici**, detti anche bottoni presinaptici, sono strutture dilatate al termine degli assoni che contengono il neurotrasmettitore [47][50].

Quando un neurone, opportunamente stimolato dai vicini o da altri eventi, sviluppa un potenziale d'azione, la depolarizzazione viaggia per tutto l'assone e raggiunge il suo terminale presinaptico; questo provoca la fuoriuscita del neurotrasmettitore, che si lega a particolari canali chemio-dipendenti dei neuroni postsinaptici, provocandone l'apertura. La sinapsi può essere eccitatoria o inibitoria a seconda del neurotrasmettitore rilasciato: se è eccitatoria, il canale aperto farà entrare una specie chimica in grado di innalzare il potenziale del neurone postsinaptico, e viceversa. La frequenza e la quantità di sinapsi eccitatorie determineranno poi il raggiungimento di un potenziale di soglia nel neurone postsinaptico, raggiunto il quale esso produrrà a sua volta un potenziale d'azione [47][50].

I processi di apprendimento e memorizzazione sono legati alla plasticità sinaptica cerebrale, in accordo con la **regola di Hebb**: *se due neuroni, tra loro connessi da una o più sinapsi, sono ripetutamente attivati simultaneamente allora le sinapsi che li connettono sono rinforzate* [47][48]. La velocità di formazione di nuove sinapsi è strabiliante nei neonati (oltre mezzo milione al secondo tra i 2 e 4 mesi di età), ma prosegue anche in età adulta [48].

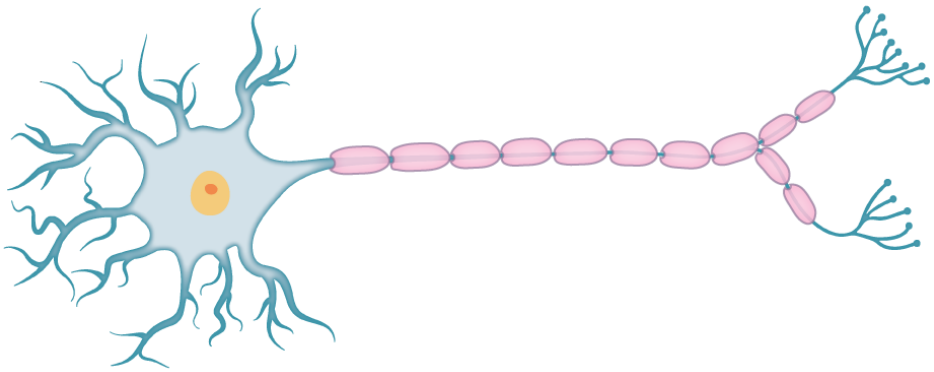


Figura 3.1: rappresentazione di un singolo neurone. Da sinistra verso destra si trovano i dendriti, il soma, l'assone e i terminali presinaptici. L'assone in figura è ricoperto dalla guaina mielica, che consente la propagazione del segnale elettrico molto più rapidamente rispetto a quella che avviene in un assone non mielinizzato (100 m/s contro 1 m/s) [51].

Il neurone artificiale.

Diversi modelli del singolo neurone e dei processi sinaptici sono stati proposti già nel decennio precedente alla nascita dell'intelligenza artificiale [31][48]. In letteratura è possibile trovare modelli anche molto sofisticati, come quello del 1952 ad opera di Hodgkin e Huxley [52], in grado di simulare l'andamento temporale del potenziale d'azione; tuttavia, collegare insieme centinaia di migliaia di neuroni artificiali di tale tipologia in un modello di rete neurale risulta computazionalmente oneroso [47].

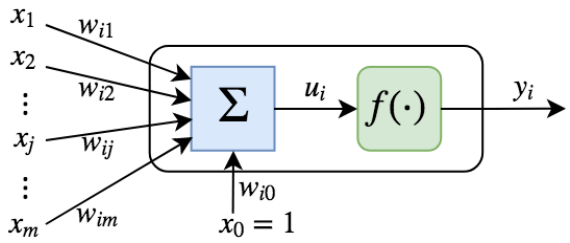


Figura 3.2: modello dell'i-esimo neurone di una rete.

Sono stati dunque proposti negli anni successive semplificazioni, fino ad arrivare al modello "spiking neuron", in cui l'informazione viene codificata dalla sola frequenza di scarica del neurone. Come ulteriore semplificazione del modello, dal momento che la dinamica temporale non viene considerata nei modelli profondi per DL, ci si limita a considerare il neurone in condizioni stazionarie ed ai valori medi. La schematizzazione complessiva del modello semplificato di neurone impiegato nelle moderne reti neurali è rappresentato in figura 3.2; tale figura sintetizza graficamente gli elementi che descrivono il neurone i-esimo di una rete [47][48]:

- x_j: **ingresso** che il neurone postsinaptico i-esimo riceve dal neurone presinaptico j-esimo.
- w_{ij}: **peso della sinapsi** fra il neurone postsinaptico i-esimo e il neurone presinaptico j-esimo; l'addestramento della rete andrà ad agire proprio su questi pesi, incrementandone il valore per simularne il rafforzamento e decrementandolo viceversa.
- w_{i0}: **bias** del neurone postsinaptico; è un ulteriore peso che si considera collegato ad un input fittizio con valore sempre unitario, utile per impostare il punto di lavoro ottimale del neurone.

- u_i: **livello di eccitazione globale** del neurone (equazione (3.1)):

$$u_i = \sum_{j=0}^{m} w_{ij} x_j. \tag{3.1}$$

- $f(\cdot)$: è una funzione scalare detta **funzione di attivazione** che determina il comportamento del neurone, ovvero come esso genera l'uscita y_i in funzione dello stato interno u_i (equazione (3.2)):

$$y_i = f(u_i). \tag{3.2}$$

La funzione di attivazione nella modellazione proposta è **non lineare, continua e differenziabile** [47][48]. Si può dimostrare che la non linearità è indispensabile affinché la rete riesca ad eseguire correttamente il mapping input-output dell'informazione [32][48]; la continuità e la differenziabilità sono invece condizioni necessarie per la back-propagation (si veda sezione 3.2) [32][47][48].

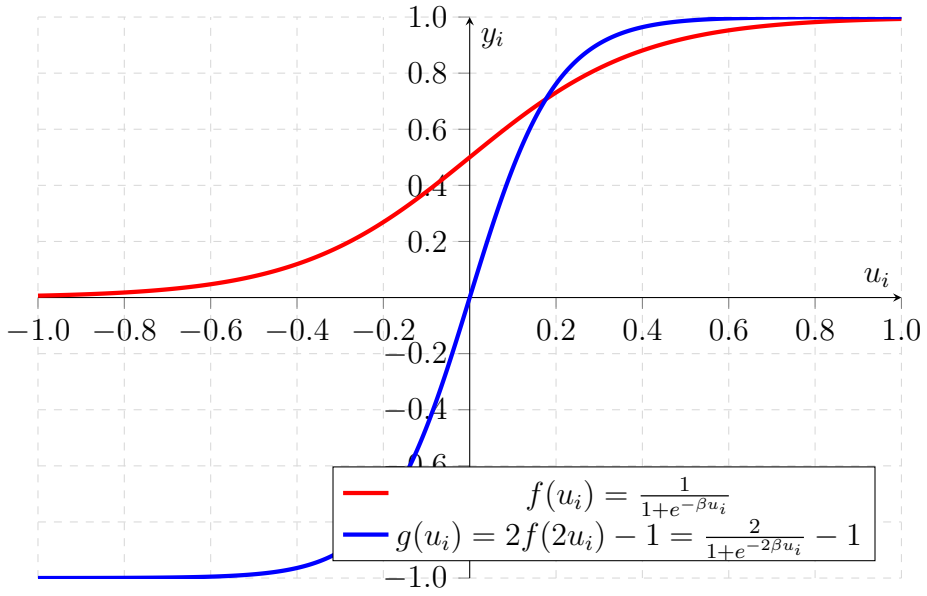


Figura 3.3: le due funzioni di attivazione più utilizzate in modelli di questo tipo: in rosso la funzione logistica (sigmoide), in blu la tangente iperbolica (una sigmoide opportunamente scalata e traslata).

3.2 Le reti neurali

Il percettrone di Rosenblatt.
La storia delle reti neurali comincia nel 1943 con il modello McCulloch-Pitts [53], una **rete feed-forward** (cioè senza retroazioni) con un neurone in uscita ed attivazione lineare. Tale semplice modello si dimostrò già in grado di operare classificazione binaria: ad uscita negativa corrispondeva classe 0 e viceversa; tuttavia, i pesi $\boldsymbol{w}$ non erano addestrabili e dovevano essere impostati correttamente a priori.

Il primo modello con pesi addestrabili fu il **percettrone di Rosenblatt**, proposto a cavallo fra gli anni '50 e '60 [54][55]. Anch'esso prevedeva uno strato di neuroni di ingresso $\boldsymbol{x}$ ed un solo neurone in uscita y (per cui l'uso del pedice i può essere omesso); tuttavia, a differenza del modello McCulloch-Pitts, il vettore dei pesi $\boldsymbol{w}$ poteva essere modificato di una quantità $\Delta\boldsymbol{w}$ applicando una **regola di addestramento**, definita nel seguito.

Si consideri di avere a disposizione p pattern diversi e di conoscere, per ciascun pattern k-esimo, l'uscita desiderata d^k. È allora possibile definire una **funzione errore** e qualsiasi, per esempio **l'errore quadratico medio** (**Mean Square Error, MSE**, equazione (3.3)):

$$e = \frac{1}{p}\sum_{k=1}^{p} e^k, \quad \text{con:} \quad e^k = \frac{1}{2}\left(d^k - y^k\right)^2 \tag{3.3}$$

dove con e^k si è evidenziato l'**errore quadratico in presenza del pattern k-esimo**, in cui si è arbitrariamente introdotto il termine $1/2$ dal momento che risulterà utile per una successiva semplificazione.

Per ognuno dei p pattern disponibili, il k-esimo è presentato in ingresso alla rete; ciò significa che il valore delle uscite dello strato di input è fatto coincidere con il valore del pattern k-esimo (si scriverà allora $\boldsymbol{x}^k$). A questo punto è possibile **propagare in avanti** il valore dell'ingresso e calcolare l'uscita y^k per il pattern k-esimo utilizzando la (3.1) e la (3.2) (equazione (3.4)):

$$y^k = f(u^k) = f\left(\sum_{j=0}^{m} w_j x_j^k\right). \tag{3.4}$$

Considerando sia $\boldsymbol{w}$ che $\boldsymbol{x}$ dei vettori colonna $m \times 1$, la (3.4) può anche essere espressa in forma vettoriale (equazione (3.5)):

$$y^k = f(u^k) = f(\boldsymbol{w}^T \cdot \boldsymbol{x}). \tag{3.5}$$

Dal momento che, come si evince dalla 3.4 e dalla 3.5, l'uscita y^k è funzione dei pesi $\boldsymbol{w}$ della rete, anche l'errore quadratico e^k lo sarà; risulta quindi possibile calcolare la derivata m-dimensionale (gradiente) della funzione errore e^k rispetto ai pesi $\boldsymbol{w}$. Il vettore risultante punterà nella direzione che massimizza la funzione errore; dunque, volendo minimizzare tale funzione, i pesi della rete andranno modificati puntando alla direzione opposta. Si può scrivere allora:

forma vettoriale		**forma scalare**	
$\Delta \boldsymbol{w} = -\gamma \nabla e^k$	(3.6)	$\Delta w_j = -\gamma \frac{\partial e^k}{\partial w_j}$	(3.7)

dove γ è detta costante di apprendimento (**learning rate**); un aumento di tale valore determinerà una maggiore modifica dei pesi, e viceversa. Definita tale regola di addestramento, si tratta ora di esprimerla nelle variabili che caratterizzano la rete; per fare ciò, possiamo utilizzare la regola della catena (equazione (3.8)):

$$\left.\begin{aligned} \Delta w_j &= -\gamma \tfrac{\partial e^k}{\partial w_j} \\ e^k &= e^k\left(y^k\left(u^k\left(\boldsymbol{x}^k; \boldsymbol{w}\right)\right)\right) \end{aligned}\right\} \implies \Delta w_j = -\gamma \frac{\partial e^k}{\partial y^k}\frac{\partial y^k}{\partial u^k}\frac{\partial u^k}{\partial w_j} \tag{3.8}$$

usando la (3.4) si ottiene infine l'equazione (3.9):

$$\Delta w_j = \gamma (d^k - y^k) f'(u^k) x_j^k. \tag{3.9}$$

Dopo aver presentato tutti i p pattern disponibili, si dice che è passata un'**epoca di addestramento**. Per addestrare una rete neurale sono necessarie diverse epoche, a volte anche centinaia o migliaia a seconda della complessità del problema [47].

Il successo di questa rete è legato al **teorema di convergenza del percettrone**. Tale teorema rappresentò la prima dimostrazione che una rete, composta unicamente da neuroni (quindi senza altri meccanismi computazionali al di fuori del calcolo eseguito dai neuroni) potesse risolvere un importante problema concreto:

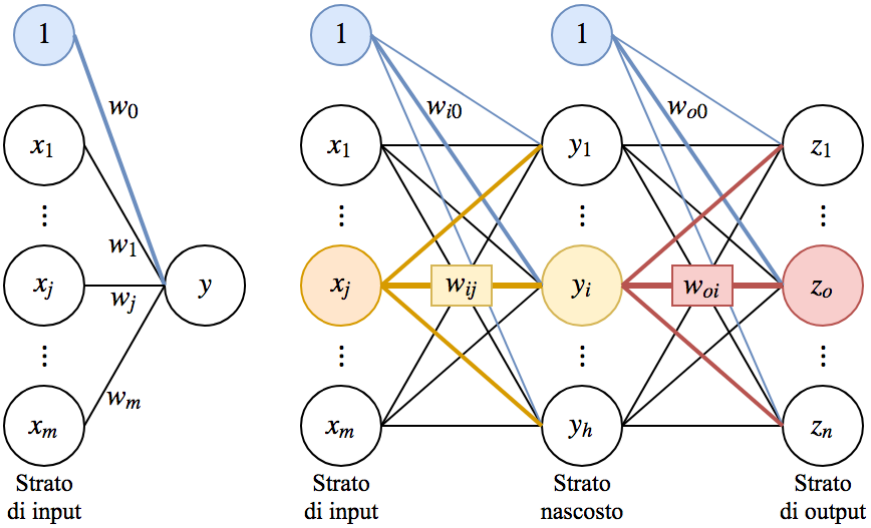
> **Teorema** (di convergenza del percettrone).
> *Si supponga che i pattern di addestramento di una rete neurale provengano da due classi linearmente separabili. Allora il percettrone, con un opportuno algoritmo di addestramento, è in grado di trovare, in un numero finito di passi, una soluzione che separa le due classi* [47].

Il percettrone multi-strato.

Il lavoro di Rosenblatt e i successivi suscitarono grande interesse nella comunità scientifica, mostrando la possibilità di risolvere importanti problemi (quali il classificatore lineare o l'interpolazione lineare) attraverso meccanismi esclusivamente neurali. Nel decennio successivo, tuttavia, alcune importanti critiche furono mosse a tali reti, soprattutto da parte dei cultori dell'intelligenza artificiale classica [31][47].

Fra le critiche più interessanti, meritano particolare attenzione quelle descritte da Minsky e Papert nel libro *The Percepton*, pubblicato nel 1969 [56]. Attraverso una serie di esempi tratti da problemi cognitivi, gli autori mostravano come la rete implementata da Rosenblatt non fosse in grado di risolvere semplici problemi cognitivi, e quindi fosse sterile per le applicazioni tipiche dell'AI [47].

I due però riconobbero anche che sarebbe stato teoricamente possibile addestrare una rete neurale per approssimare qualsiasi funzione utilizzando un percettrone multistrato (**Multilayer Perceptron, MLP**), ovvero un percettrone modificato in modo che presentasse degli strati nascosti fra quello di input e quello di output [31][32][47][48]. Un esempio con m ingressi, n uscite ed h neuroni nascosti è rappresentato in figura 3.4b; in questo caso, i pesi fra uno strato e il successivo saranno allora organizzati in matrici (genericamente indicate con $\boldsymbol{W}$).



(a) percettrone. (b) percettrone multi-strato.

Figura 3.4: rappresentazione grafica del percettrone di Rosenblatt (a) e di un MLP con un solo strato nascosto (b).

Solo più tardi, nel 1989, arrivò la formalizzazione matematica delle intuizioni di Minsky e Papert, grazie ad un teorema dovuto a Cybenko [57] e noto come **teorema dell'approssimazione universale**. Tale teorema assicurava che:

> **Teorema** (dell'approssimazione universale).
> *Una rete neurale feed-forward con uno strato di uscita lineare ed almeno uno strato nascosto con una qualsiasi funzione di attivazione che restituisce un'uscita in un range fissato può approssimare ogni funzione continua su un sottoinsieme chiuso e limitato di $\mathbb{R}^n$ con un errore desiderato diverso da 0, purché la rete sia progettata con un numero sufficiente di unità nascoste* [31].

Guardando alla rete di figura 3.4b, conoscendo le uscite desiderate d_o^k è possibile addestrare le sinapsi w_{oi} fra lo strato nascosto e lo strato d'uscita con lo stesso procedimento visto per il percettrone di Rosenblatt. Tuttavia, per poter utilizzare l'MLP per ogni tipo di problema, occorre poter addestrare anche le sinapsi w_{ij} che vanno dallo strato di input allo strato nascosto. Dal momento che non è possibile conoscere le uscite desiderate d_i^k per lo strato nascosto, Minsky e Papert decretarono che non fosse possibile utilizzare reti neurali per gli algoritmi di intelligenza artificiale [31][56].

Tale critica e l'avvento del primo inverno AI portò ad un'enorme riduzione dei fondi e delle attività di ricerca sulle reti neurali per tutti gli anni '70. Soltanto negli anni '80, con la scoperta dell'algoritmo di back-propagation, è stato possibile addestrare le reti neurali multistrato [47].

La back-propagation.
Introdotto nel 1988 da Rumelhart, Hinton e Williams [58], l'algoritmo back-propagation consiste in realtà nell'applicazione **della regola di derivazione a catena** per propagare all'indietro l'errore di classificazione commesso; in questo modo diventò possibile addestrare anche reti neurali multi-strato [31][47][48]. Guardando a figura 3.4b si può già scrivere:

strato z		**strato y**	
$z_o^k = f(v_o^k)$	(3.10)	$y_i^k = f(u_i^k)$	(3.12)
$v_o^k = \sum_{i=0}^{h} w_{oi} y_i^k$	(3.11)	$u_i^k = \sum_{j=0}^{m} w_{ij} x_j^k$	(3.13)

dove si è impiegato il vettore $\boldsymbol{v}$ per rappresentare il livello di eccitazione globale dello strato di uscita. Volendo definire la stessa funzione errore impiegata per il percettrone di Rosenblatt, sarà necessario estendere l'errore quadratico per il pattern k-esimo e^k a tutti i neuroni dello strato $\boldsymbol{z}$ di output:

$$e^k = \frac{1}{2}\sum_{o=1}^{n}\left(d_o^k - z_o^k\right)^2. \tag{3.14}$$

Usando la legge di addestramento (3.7), si dovrà avere:

$$\Delta w_{oi} = -\gamma\frac{\partial e^k}{\partial z_o^k}\frac{\partial z_o^k}{\partial v_o^k}\frac{\partial v_o^k}{\partial w_{oi}^k} \tag{3.15}$$

$$\Delta w_{ij} = -\gamma\frac{\partial e^k}{\partial y_i^k}\frac{\partial y_i^k}{\partial u_i^k}\frac{\partial u_i^k}{\partial w_{ij}^k}. \tag{3.16}$$

Usando la (3.11) e la (3.13):

$$\Delta w_{oi} = \gamma\delta_o^k y_i^k \tag{3.17}$$

$$\Delta w_{ij} = \gamma\delta_i^k x_j^k \tag{3.18}$$

dove si è posto:

$$\delta_o^k = -\frac{\partial e^k}{\partial z_o^k}\frac{\partial z_o^k}{\partial v_o^k} \tag{3.19}$$

$$\delta_i^k = -\frac{\partial e^k}{\partial y_i^k}\frac{\partial y_i^k}{\partial u_i^k}. \tag{3.20}$$

Usando poi la (3.14), la (3.10) e la (3.12):

$$\delta_o^k = (d_o^k - z_o^k)f'(v_o^k) \tag{3.21}$$

$$\delta_i^k = -\tfrac{\partial e^k}{\partial y_i^k}f'(u_i^k). \tag{3.22}$$

È rimasto quindi un unico elemento incognito ($\frac{\partial e^k}{\partial y_i^k}$) nella (3.22); è proprio tale quantità che ha interdetto lo sviluppo delle reti neurali per quasi quindici anni [47]. Si procede dunque a questo punto propagando l'errore e^k all'indietro considerando le sue dipendenze:

$$\begin{aligned} e^k &= \frac{1}{2}\sum_{o=1}^{n}\left(d_o^k - z_o^k\right)^2 \\ e^k &= e^k\left(\boldsymbol{z}^k\left(\boldsymbol{v}^k\left(\boldsymbol{y}^k, \boldsymbol{W}\right)\right)\right) \end{aligned} \quad \Longrightarrow \quad -\frac{\partial e^k}{\partial y_i^k} = \sum_{o=1}^{n} -\frac{\partial e^k}{\partial z_o^k}\frac{\partial z_o^k}{\partial v_o^k}\frac{\partial v_o^k}{\partial y_i^k}.$$

Infine, impiegando la 3.19 e la 3.11 si ottiene l'equazione (3.23):

$$-\frac{\partial e^k}{\partial y_i^k} = \sum_{o=1}^{n}\delta_o^k w_{oi}. \tag{3.23}$$

Si è dunque dimostrato che propagando all'indietro l'errore è possibile addestrare anche un percettrone con uno strato nascosto. Tale procedura si può estendere anche al caso di una rete con un numero strati nascosti grande a piacere [31][32].

Addestramento di un MLP a tre strati.

Di seguito lo pseudo-codice per l'addestramento di un MLP con un solo strato nascosto, procedura scalabile ad un modello profondo. L'addestramento prosegue fino a che non sono trascorse le epoche desiderate n_{epoche} o la **funzione costo** (loss function, si veda sezione 3.5) non scende sotto una certa soglia th_{loss}; qui si utilizza l'MSE e. $\boldsymbol{w}_i$ e $\boldsymbol{w}_o$ sono le righe i-esime e o-esime delle rispettive matrici dei pesi.

```
- # carico il training set
- X ← training set;    d ← etichette
-
- # inizializzo variabili
- inizializzo h, γ, th_loss, n_epoche
- inizializzo {w_ij}, {w_oi} (random)
- loss ← ∞;    epoca ← 1
-
- # addestramento
- while (loss ≥ th_loss & epoca ≤ n_epoche) do:
-     # inizializzo l'errore
-     e ← 0
-
-     # presento ogni pattern
-     for each x^k in X do:
-         # forward-propagation
-         y_i^k ← w_i · x^k,    i = 1, 2, ..., h
-         z_o^k ← w_o · y^k,    o = 1, 2, ..., n
-
-         # aggiorno l'errore
-         e ← e + 1/2 Σ_o (d_o^k − z_o^k)^2
-
-         # backward-propagation
-         δ_o^k ← f'(v_o^k) · (d_o^k − z_o^k),    o = 1, 2, ..., n
-         δ_i^k ← f'(u_i^k) Σ_o δ_o^k · w_oi,    i = 1, 2, ..., h
-         w_oi ← w_oi + γ · δ_o^k · y_i^k
-         w_ij ← w_ij + γ · δ_i^k · x_j^k
-
-     # aggiorno loss ed epoca
-     loss ← e/p;    epoca ← epoca + 1
```

Modelli profondi.

Con il termine **Deep Neural Networks (DNN)** si denotano reti "profonde" composte da molti strati (almeno 2 nascosti), l'uno di seguito all'altro [31][32].

Le implicazioni del teorema di approssimazione universale e la difficoltà di addestrare reti con molti livelli hanno portato per lungo tempo a focalizzarsi su reti con un solo strato nascosto [31]. Tuttavia, l'**esistenza di soluzioni non implica efficienza**: esistono funzioni computabili con complessità polinomiale operando su k livelli che richiedono invece complessità esponenziale se si opera su $k-1$ [59].

Le DNN ottengono già buone prestazioni al termine degli anni '90 su problemi di piccole dimensioni (es. riconoscimento caratteri, riconoscimento oggetti a bassa risoluzione), ma bisogna attendere il 2012 con la rete AlexNet [60] per un radicale cambiamento [48]. AlexNet non introduce rilevanti innovazioni rispetto alle reti dei decenni precedenti, ma alcune condizione al contorno sono nel frattempo cambiate:

- **Big data.** La superiorità delle tecniche di DL rispetto ad altri approcci di ML si manifesta quando sono disponibili grandi quantità di dati di training (figura 3.5).

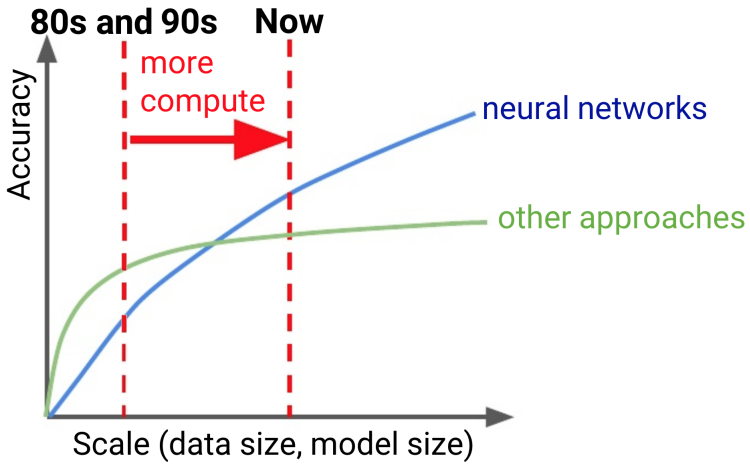
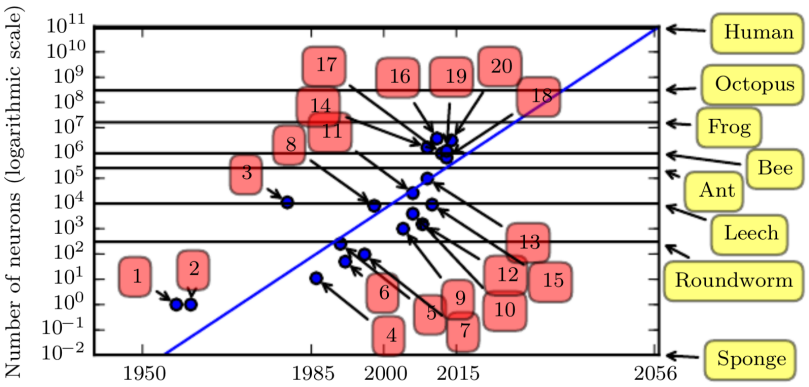
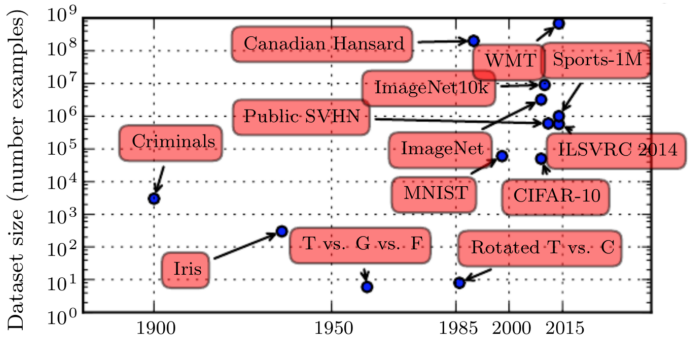


Figura 3.5: grafico di confronto fra l'accuratezza media degli algoritmi di DL (curva blu) e quella ottenibile con altri approcci di intelligenza artificiale (curva verde). Mentre con pochi dati a disposizione la curva blu rimane al di sotto di quella verde, il trend si inverte aumentando la dimensione del dataset. Questa immagine è in grado di spiegare l'odierna esplosione delle tecniche di DL e delle difficoltà incontrate invece in passato [61].

- **GPU computing.** Il training di modelli complessi (profondi e con molti pesi e connessioni) richiede un'elevata potenza computazionale. La disponibilità di schede video (GPUs) con migliaia di core e GB di memoria dedicata ha consentito di ridurre drasticamente i tempi di training: da mesi a giorni [32][48].
- **Attivazione ReLU.** La back-propagation è problematica con la sigmoide come attivazione negli strati nascosti; risultati migliori si ottengono con attivazione ReLU (si veda 3.4) [31][48].



(a) variazione del numero di neuroni nei modelli neurali negli anni [31].



(b) variazione delle dimensioni dei dataset negli anni [31].

Figura 3.6: alla numerazione del pannello (a) corrispondono le seguenti reti: **(1)** percettrone [54][55]; **(2)** ADALINE [62]; **(3)** Neurocognitron [63]; **(4)** early back-propagation network [64]; **(5)** RNN for speech recognotion [65]; **(6)** MLP for speech recognition [66]; **(7)** mean field sigmoid belief network [67]; **(8)** LeNet-5 [68]; **(9)** echo state network [69]; **(10)** deep belief network [70]; **(11)** GPU-accelerated convolutional network [71]; **(12)** Deep Boltzmann machine [72]; **(13)** GPU-accelerated deep belief network [73]; **(14)** unsupervised convolutional network [74]; **(15)** GPU-accelerated multilayer perceptron [75]; **(16)** OMP-1 network [76]; **(17)** distributed autoencoder [68]; **(18)** Multi-GPU convolutional network [60]; **(19)** COTS HPC unsupervised convolutional network [77]; **(20)** GoogLeNet [78].

3.3 Reti neurali convoluzionali

Dalle reti fully-connected alle convoluzionali.

Le reti viste finora sono in grado di accettare in ingresso solamente oggetti mono-dimensionali: i pattern utilizzati sono vettori $\boldsymbol{x}$ dove ciascun elemento x_j è collegato con tutti i neuroni dello strato successivo; per questo motivo, reti di questo tipo sono anche dette **Fully-Connected (FC)**. Con reti del genere, per processare ingressi 2-D o 3-D, come ad esempio immagini o, più coerentemente con l'oggetto di questa tesi, il segnale EEG $\boldsymbol{E}$ di un task motorio (si rammenti figura 2.5), sarebbe necessario ridimensionare la matrice o il tensore in modo da ottenerne una rappresentazione vettoriale (operazione chiamata "flattening") [31][32].

Tuttavia, esistono reti specializzate per questa tipologia di dati: le **Convolutional Neural Networks (CNN)**. Si tratta di reti feed-forward in cui, almeno per uno strato, viene utilizzata l'operazione lineare di convoluzione invece della generica propagazione in avanti tramite moltiplicazione matriciale ($\boldsymbol{y} = \{w_{ij}\}\,\boldsymbol{x}$) [31][32][48].

L'operazione di convoluzione.

Si consideri una rete con uno strato di ingresso matriciale $\boldsymbol{X}$ di dimensione qualsiasi, non monodimensionale come visto sin ora per le reti FC. Sia poi $\boldsymbol{K}$ una matrice, anch'essa di dimensione qualsiasi, detta **kernel** o, più genericamente, filtro. Nel caso particolare delle CNN, il kernel ha entrambe le dimensioni (righe e colonne) minori rispetto a quelle dello strato di ingresso $\boldsymbol{X}$; ciò comporterà dei notevoli vantaggi, come spiegato poco più avanti.

L'operazione di **convoluzione discreta 2-D** consiste del traslare il kernel $\boldsymbol{K}$ sull'intera estensione dello strato $\boldsymbol{X}$, calcolando per ogni posizione di $\boldsymbol{K}$ la somma del prodotto elemento per elemento fra $\boldsymbol{K}$ e la sotto-matrice $\boldsymbol{X'}$ ottenuta dalla sovrapposizione delle due matrici (si veda l'esempio grafico di figura 3.7). Come risultato si ottiene una nuova matrice $\boldsymbol{Y}$ in cui l'elemento in posizione (ij) y_{ij} è dato dalla (3.24):

$$y_{ij} = (\boldsymbol{X} * \boldsymbol{K})_{ij} = \sum_n \sum_m x_{i-m,j-n} k_{mn} \tag{3.24}$$

dove con la notazione $(\boldsymbol{K} * \boldsymbol{X})_{ij}$ si indica l'elemento (ij) della matrice risultante dalla convoluzione [31].

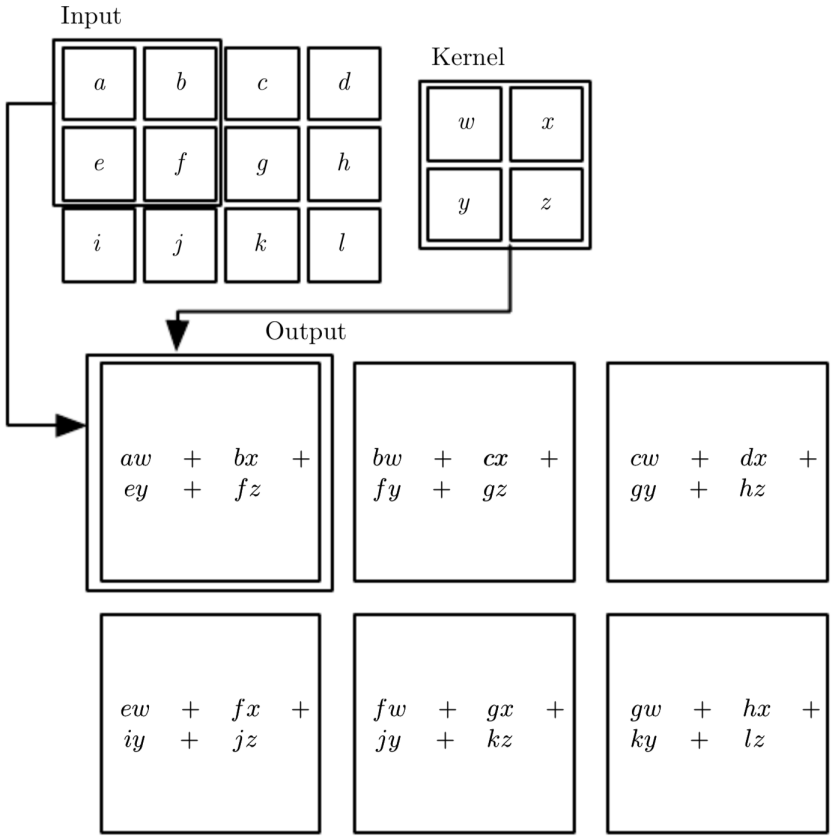


Figura 3.7: un esempio grafico di convoluzione 2-D. Si è qui limitato il calcolo alle sole posizioni in cui il kernel giace interamente all'interno della matrice di input, ma esistono metodologie per estendere il kernel anche oltre i bordi di esso. Nell'immagine è possibile seguire il cammino delle frecce per comprendere come venga formato l'elemento in alto a sinistra della matrice di uscita, applicando il kernel alla corrispondente regione superiore sinistra di quella di ingresso.

Ispirazione e giustificazione biologica.

L'ispirazione per le CNN arriva ancora una volta dal mondo biologico, in particolare dallo studio dal **campo recettivo** dei neuroni che compongono la **via ottica principale**. Con campo recettivo si intende la regione dell'area visiva in cui deve cadere uno stimolo per attivare un neurone; la via ottica principale invece identifica i vari gruppi di neuroni che portano l'informazione visiva dalla retina alla corteccia. Essa è formata da tre strati principali: **(1) cellule gangliari retiniche**; **(2) nucleo genicolato laterale**; **(3) corteccia visiva** [79].

Gli esperimenti di Hubel e Wiesel.
A partire dalla fine degli anni '50 e per tutto il decennio successivo, David Hubel e Torsten Wiesel studiarono le strutture neurali coinvolte nella visione, prima nel gatto [80][81], quindi nella scimmia [82]. Dopo aver inserito degli elettrodi in diversi punti del cervello degli animali, mostrarono loro alcuni stimoli luminosi con l'ausilio di uno schermo. Osservando le zone in cui si verificava una maggior attivazione neurale, fu loro possibile stimare il campo recettivo dei vari strati della via ottica principale, deducendo due importanti conclusioni:

- **Mappatura topografica della corteccia.** Cellule vicine rappresentano regioni vicine nel campo visivo [32].
- **Struttura gerarchica della via ottica principale.** Le cellule appartenenti al primo strato del sistema visivo sono eccitate in modo diretto dagli stimoli luminosi e i loro campi recettivi sono determinati esclusivamente dal modo in cui tali cellule interagiscono tra loro sullo stesso strato. Le cellule degli strati successivi, invece, sono eccitate dagli stimoli luminosi solo per via indiretta: lo stimolo viene dunque alterato da elaborazioni compiute dagli strati precedenti [79]. Per questo motivo il campo recettivo si modifica in maniera gerarchica lungo gli strati: se le cellule gangliari retiniche hanno un campo circolare, in quanto condizionato uniformemente dallo stimolo visivo, gli strati più profondi hanno un campo recettivo sempre più complesso e sono in grado di estrarre dall'immagine feature particolari, come bordi, orientamento e direzione dello stimolo luminoso [32].

L'esplosione della computer vision.
L'organizzazione gerarchica della via ottica e la profonda interazione fra le cellule dello stesso strato mise in discussione l'utilizzo di reti neurali FC, in cui non sono presenti meccanismi di integrazione dell'informazione proveniente da neuroni vicini fra loro [32].

I primi tentativi di sopperire a tale mancanza risalgono al 1980 con la rete "Neurocognitron" [63], nel 1998 ci fu la prima applicazione della back-propagation su strati convoluzionali (LeCun et al. [68]) mentre il primo esempio di CNN nella concezione contemporanea comparve nel 2012 con la rete "AlexNet" [60], in grado di raggiungere accuratezze molto più elevate rispetto al passato nella classificazione delle immagini. Da quel momento ad oggi, la computer vision conobbe un miglioramento esponenziale, dovuto proprio alle CNN [32].

Applicazione di strati convoluzionali.

Nel seguito si dà la definizione di **strato convoluzionale** di una rete neurale nel caso più generico possibile, ovvero quando si hanno strati e kernel multi-dimensionali (si farà uso cioè di variabili tensoriali). Nei problemi di computer vision, ad esempio, si ha spesso a che fare con immagini a tre canali (R, G, B); lo strato di ingresso della rete $\mathbf{X}$ dovrà allora disporre i neuroni sia in altezza che in larghezza che in profondità, per cui si avrà uno strato di ingresso di dimensione $h \times w \times d$ (con $d = 3$ nel caso di immagine a tre canali).

Definizione (di strato convoluzionale di una rete neurale)
Quando si convolve un filtro $\mathbf{K}$ *con l'ingresso* $\mathbf{X}$ *di una rete neurale ottenendo l'uscita* $\mathbf{Y}$*, si dice allora che tale strato è uno strato convoluzionale.*

Quando si utilizza uno strato convoluzionale in una rete neurale si desidera estrarre dall'input più feature possibili; per farlo, si può convolvere lo stesso input con diversi filtri ed ottenere altrettante versioni del segnale filtrato in uscita; dunque un kernel 3-D $\mathbf{K}$ non è altro che l'insieme di più filtri $\boldsymbol{K}$ raggruppati in un unico tensore, che verranno però utilizzati uno alla volta; ognuno di questi genererà un'uscita 2-D $\boldsymbol{Y}$, che si possono a loro volta raggruppare in uno strato di uscita 3-D $\mathbf{Y}$ (si veda figura 3.8).

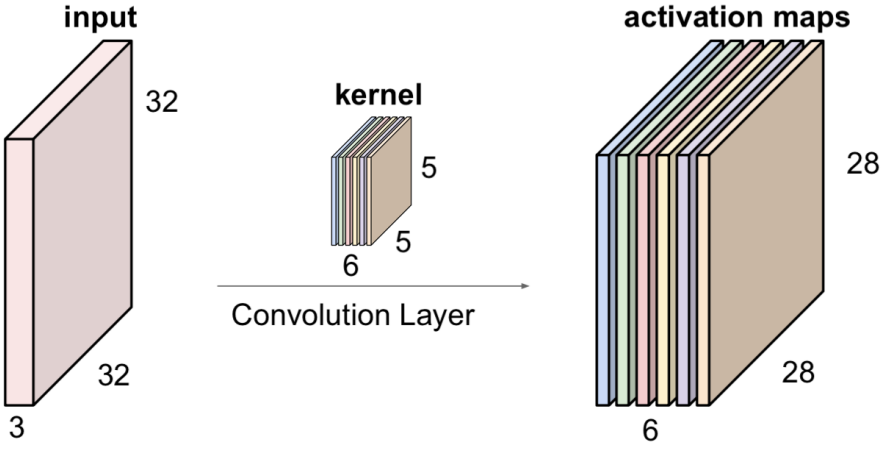


Figura 3.8: convoluzione su un ingresso $32 \times 32 \times 3$ con un kernel $5 \times 5 \times 6$ [32].

La singola uscita di ogni filtro $\boldsymbol{Y}$ è detta **mappa di attivazione** (Activation Map, AM). Dal punto di vista neuroscientifico, ogni elemento della AM può essere interpretato come l'uscita di un neurone che osserva una piccola regione dell'ingresso rilevandone particolari caratteristiche e condivide i parametri con tutti i neuroni spazialmente vicini [32].

Definizione di modello, strato denso e di parametro.
Quando si progetta una rete neurale, ciò che si fa è concatenare diversi strati andando a formare **l'architettura della rete**.

Una CNN tipicamente è costituita da un blocco convoluzionale ed un blocco fully-connected, entrambi caratterizzati da più strati (per una disamina degli altri possibili strati, si veda sezione 3.4). Il primo blocco è costituito da strati convoluzionali ed opportuni strati che operano su oggetti multi-dimensionali e che verranno introdotti in seguito. Il secondo blocco invece, è costituito da uno strato in cui viene applicata l'**operazione di flattening** (che trasforma una matrice o un tensore in un vettore) e da strati fully-connected, chiamati anche **strati densi**.

In seguito ci si riferisce più genericamente all'insieme degli elementi che compongono i kernel e dei pesi degli strati densi con il termine parametri. I parametri possono poi dividersi in **parametri addestrabili** (se essi si modificano in fase di training) e in **parametri non addestrabili**.

Vantaggi degli strati convoluzionali.
In uno strato denso, i parametri addestrabili sono i pesi w_{ij} che collegano ogni neurone dello strato d'uscita con tutti i neuroni dello strato di ingresso. Invece, **in uno strato convoluzionale gli unici parametri addestrabili sono quelli dei filtri**. Dal momento che si usano filtri di dimensioni contenute (in molti casi molto più contenute della dimensione dello strato di input), questo è vantaggioso sia dal punto di vista computazionale (meno operazioni da eseguire) sia per la memoria da allocare sulla macchina per creare il modello della rete.

3.4 Progetto di reti neurali

Implementazione di modelli tramite framework.

Programmare DNN da zero è possibile, ma molto complicato. È necessario infatti implementare, per ciascuno strato si voglia utilizzare, l'algoritmo di forward-propagation, di back-propagation e allo stesso è necessario parametrizzare il tutto affinché sia possibile e semplice il tuning degli iper-parametri. Inoltre, sarebbe preferibile – se non necessario – ottimizzare il codice affinché sia possibile eseguirlo su GPU (anche più di una).

Fortunatamente sono disponibili numerosi framework (spesso open-source), ovvero librerie in cui è già disponibile tutto ciò di cui si ha bisogno per creare il proprio modello di Deep Learning. I più famosi ed utilizzati sono **TensorFlow** [83] (programmato da Google), e **PyTorch** [84] (programmato da Facebook, Facebook Inc., Menlo Park, California, Stati Uniti). Entrambi questi framework sono disponibili sia in Python che in C++, mentre la parte di codice per l'accesso alle risorse computazionali della GPU è scritta in CUDA [85] (dunque eseguibile su schede video NVIDIA).

Un framework particolarmente semplice ed intuitivo da utilizzare è **Keras** [86] (programmato dal MIT, Massachusetts Institute of Technology, Stati Uniti). Scritto in Python, questo Framework permette di accedere con codice di più alto livello ad altre librerie, come Tensorflow per esempio.

Scelta degli strati per comporre un modello.

Come detto al termine della sezione precedente (3.3), nella fase di progettazione di un modello di rete neurale si vanno a concatenare, uno dopo l'altro, i diversi strati che comporranno l'architettura desiderata. Tuttavia, un metodo universalmente valido che sia di aiuto per la scelta di tali strati non esiste. Sarà dunque necessario fare diversi tentativi, modificando volta per volta l'architettura della rete o gli iper-parametri forniti in ingresso ai vari strati.

Per massimizzare le possibilità di costruire un'architettura vincente, sarà bene allora conoscere con precisione i vari strati con cui è possibile formarla. Di seguito, dunque, si riporta un elenco dei principali strati, con un'analisi dettagliata di ciascuno di essi.

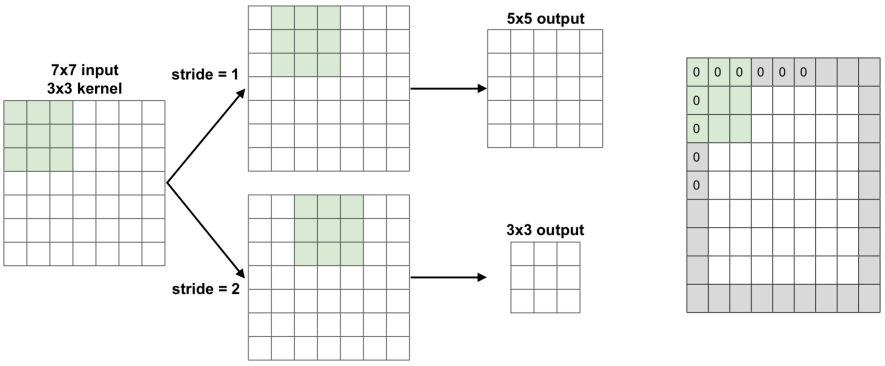
Strato di input.

Il primo strato che è necessario definire è ovviamente quello di input. Nel caso più generico, esso sarà un tensore $\mathbf{X}$ le cui dimensioni dovranno essere pari a quelle dei possibili input. A differenza del modello neurale originale (figura 3.2), i neuroni che compongono questo strato e come tutti quelli a seguire nella trattazione, non includono la funzione di attivazione; essa verrà infatti applicata da strati appositi, come spiegato poco più avanti.

Strato convoluzionale.

È possibile definire uno strato convoluzionale impostando i seguenti iper-parametri: **(1) numero dei filtri**; **(2) dimensione dei filtri**; **(3) stride**; **(4) padding**. Lo stride è il passo con cui il filtro scorre sull'ingresso (figura 3.9a). Il padding è invece una tecnica con cui si va ad aggiungere dei valori di bordo (tipicamente 0, **zero-padding**) al volume di input, rendendo così possibile posizionare il centro del filtro anche su tutti i punti dell'ingresso (figura 3.9b). Questo risulta utile qualora si desideri regolare la dimensione del tensore di uscita allo strato convoluzionale, fino a renderla pari a quella dello strato di ingresso sotto opportune condizioni dell'operazione di padding. Siano $\boldsymbol{X}$ (input) e $\boldsymbol{K}$ (kernel) due matrici quadrate e siano n_x ed n_k le rispettive dimensioni. Se lo stride è unitario, allora il numero di righe o colonne che devono essere aggiunte con padding è (equazione (3.25)):

$$p = \frac{n_k - 1}{2}. \tag{3.25}$$



(a) esempio di convoluzione con due stride diversi. (b) zero-padding.

Figura 3.9: esempi di convoluzione con differenti stride e con zero-padding [4].

Se invece si indica con s lo stride, è allora possibile calcolare la dimensione n_y dell'uscita applicando la (3.26):

$$n_y = \frac{n_x - n_k + 2p}{s} + 1. \tag{3.26}$$

Strato di pooling.
Si tratta di uno stato che opera un semplice sotto-campionamento. Può essere pensato come uno strato convoluzionale in cui il kernel ha il solo scopo di ridurre la dimensionalità dell'ingresso. La modalità di pooling più utilizzata è la variante **max pooling**, dove un filtro scorre sull'ingresso con un certo stride (esattamente come in uno strato convoluzionale), ma l'unica operazione svolta dal filtro è quella di estrarre l'elemento massimo fra quelli dell'input (figura 3.10). Un'altra variante meno utilizzata è l'**average pooling**, dove il funzionamento è lo stesso ma l'operazione svolta del filtro è questa volta la media. Gli iper-parametri con cui si crea uno strato di pooling sono infatti: **(1) pool size** (dimensione del filtro di pooling); **(2) stride**; **(3) padding**.

Strato di flattening.
Strato che compie banalmente l'operazione di flattening, in cui il tensore viene convertito in un vettore. In questo modo è possibile passare da una "logica convoluzionale" ad una "logica densa".

Strato denso.
Implementa uno strato fully-connected, costruendo cioè un nuovo strato vettoriale dove ciascun neurone è collegato a tutti i neuroni dello strato precedente. Gli iper-parametri che definiscono uno strato denso sono: **(1) numero delle unità**, che definisce di conseguenza anche il numero dei pesi; **(2) utilizzo dei bias**.

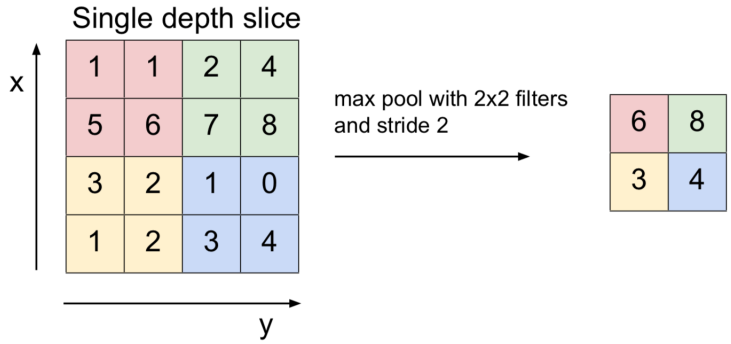
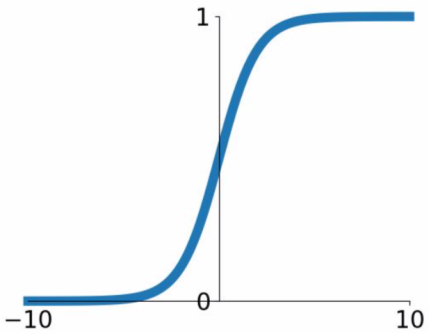


Figura 3.10: esempio di max pooling [32].

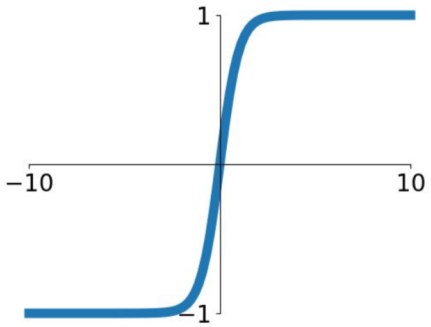
Strato di attivazione.

Come già anticipato, i neuroni degli strati visti fino ad ora non sono provvisti di funzione di attivazione. Per questa ragione, è necessario aggiungere uno strato il cui unico scopo è applicare una qualche funzione di attivazione ad ogni neurone dello strato precedente.

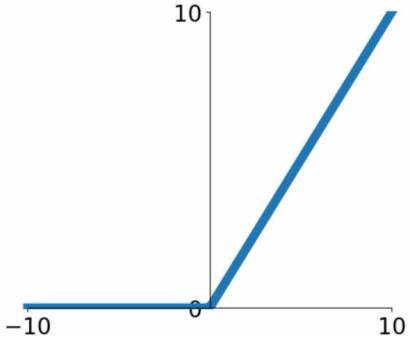
Il teorema dell'approssimazione universale (visto in sezione 3.2) garantisce che una rete neurale può approssimare con successo una qualsiasi funzione continua tramite l'utilizzo di almeno uno strato nascosto con funzione di attivazione che restituisce un'uscita in un range fissato. Per questo motivo, storicamente si sono utilizzate funzioni di attivazione che riuscissero a mappare l'input in un output limitato, come ad esempio la **sigmoide** (già vista in sezione 3.2).



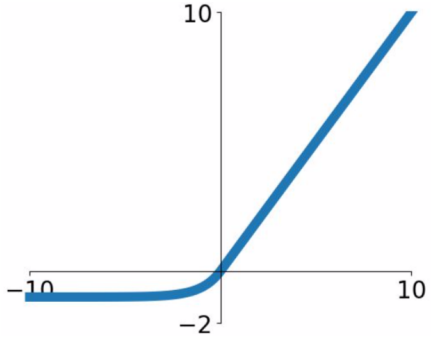
(a) sigmoide.



(b) tangente iperbolica.



(c) ReLU.



(d) ELU.

Figura 3.11: alcune funzioni di attivazione fra le più utilizzate o più storicamente rilevanti [32].

Tuttavia, la funzione di attivazione sigmoidale non si è rivelata particolarmente efficace nel training di reti neurali, per via di due problematiche ad essa legate [32]:

1. **Riduzione eccessiva del gradiente ("vanishing gradient").** Si è visto in sezione 3.2 che l'addestramento di una rete neurale si basa sulla discesa del gradiente della funzione errore lungo l'iperspazio individuato dai vettori colonna $\boldsymbol{w}_i$ della matrice dei pesi $\boldsymbol{W}$. Per calcolare la variazione dei pesi $\Delta\boldsymbol{W}$ ad ogni passo, è necessario propagare all'indietro l'errore e si è visto che $\Delta\boldsymbol{W}$ dipende linearmente dalla derivata della funzione di attivazione $f'(u)$, come indicato da equazione (3.9). Si consideri ora un neurone con livello di eccitazione globale u molto positivo o molto negativo. Come si vede in figura 3.11a, quando l'ingresso è appunto molto positivo o molto negativo, ci si trova in una regione in cui la derivata è nulla. Per questo motivo, si dice che la sigmoide *uccide il gradiente*, in quanto la variazione del peso per un neurone operante in regione di saturazione della sigmoide sarà sempre nulla [32].

2. **Complessità computazionale.** Come visto in sezione 3.2 e qui riportato nell'equazione (3.27), per ricavare l'uscita di questa funzione di attivazione è necessario calcolare l'esponenziale, computazionalmente dispendioso [32].

Di seguito si associano i pannelli di figura 3.11 con le rispettive equazioni, commentandone vantaggi e svantaggi.

- **Sigmoide.** La sigmoide è stata storicamente largamente utilizzata dal momento che rappresentava molto bene la curva di saturazione della frequenza degli spike di un neurone. Tuttavia, abbiamo già commentato gli svantaggi associati al suo utilizzo [32].

$$y(u) = \frac{1}{1 + e^{-\beta u}} \tag{3.27}$$

- **Tangente iperbolica.** Storicamente molto utilizzata per le stesse ragioni, anch'essa presenta gli stessi svantaggi [32].

$$y(u) = \frac{2}{1 + e^{-2\beta u}} - 1 \tag{3.28}$$

- **ReLU.** L'attivazione ReLU (**Rectified Linear Units**) è di gran lunga la funzione di attivazione più utilizzata [87], in quanto è in grado di risolvere entrambi i problemi legati alle precedenti [32]. Inoltre, si è visto sperimentalmente che reti neurali basate su questa attivazione convergono verso il minimo della funzione costo molto più velocemente [32].

$$y(u) = \max(0,\, \mathrm{u}) \tag{3.29}$$

  Tuttavia, è necessario prestare attenzione a quando l'ingresso è negativo. In questo caso, l'uscita sarà sempre nulla, che comporta anche in questo caso un'uccisione il gradiente. Per questo motivo è indispensabile prestare attenzione ad inizializzare i pesi della rete in modo che siano leggermente positivi [32].

- **ELU.** Sempre per far fronte al problema precedente, in aggiunta a questa accortezza legata all'inizializzazione dei pesi, sono nate negli anni alcune varianti particolari, come l'attivazione ELU (**Exponential Linear Units**) di equazione (3.30), dove la funzione di attivazione per valori negativi dell'ingresso ha andamento esponenziale monotono crescente [32].

$$y(u) = \begin{cases} u & \iff u \geq 0 \\ \alpha \cdot (e^u - 1) & \iff u < 0 \end{cases} \tag{3.30}$$

Strato di dropout.
Il dropout [88] è un metodo di regolarizzazione molto potente e computazionalmente poco costoso. Grazie a questa tecnica, viene rimosso dallo strato precedente un neurone con una certa probabilità di "drop" (drop rate) e la rete complessiva viene addestrata con una frazione di neuroni disattivati volutamente. La probabilità che un singolo neurone di una rete venga rimosso è l'iper-parametro dello strato di dropout. Per come opera questo strato, può essere posizionato unicamente a valle di uno strato d'ingresso o nascosto. L'addestramento di una rete neurale con dropout è equivalente ad addestrare un insieme di reti costituito da tutte le sotto-reti che si ottengono rimuovendo dallo strato bersaglio del dropout una certa frazione dei neuroni che lo costituiscono [31]. Si faccia riferimento a figura 3.12 di pagina seguente per un esempio pratico in cui è stato applicato uno strato di dropout a valle dello strato d'ingresso e dell'unico strato nascosto utilizzato.

La rimozione di un'unità dalla rete può esser facilmente introdotta moltiplicando il valore della sua uscita per 0. Infatti, durante l'addestramento, ogni volta che viene valutata la funzione costo per poi calcolarne il gradiente rispetto ai vari parametri, lo strato di dropout costruisce in maniera casuale una maschera binaria da applicare a tutte le unità di ingresso. Escludendo questa modifica, la propagazione in avanti e all'indietro dell'informazione procede regolarmente, così come l'aggiornamento dei parametri addestrabili [31].

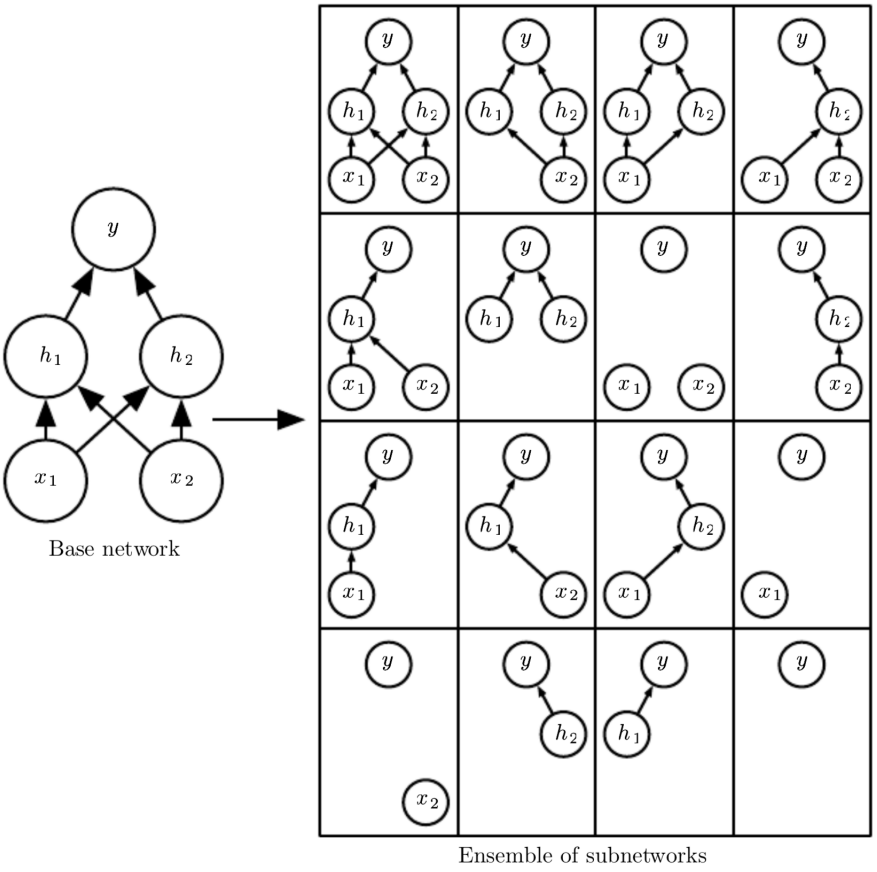


Figura 3.12: a partire da una semplice rete di base costituita da due neuroni di ingresso e due nascosti, sono qui mostrate tutte le sedici sotto-reti che è possibile ottenere rimuovendo uno, due, tre o anche tutti e quattro i neuroni suddetti. In questo piccolo esempio, una grande percentuale delle reti risultanti non ha unità di input o nessun percorso che collega l'input all'output: tali reti non sono addestrabili e quindi andranno scartate. Questo problema diventa però insignificante per le reti con livelli più ampi, dove la probabilità di eliminare tutti i percorsi possibili diventa molto più piccola [31].

Strato di batch normalization.
Come si è visto nel paragrafo relativo allo strato di attivazione, nell'utilizzare l'attivazione ReLU (o ogni sua altra variante) diventa molto importante inizializzare a dovere i parametri della rete [32]. Le possibili modalità di inizializzazione sono ancora in fase di studio, ma in generale si è osservato che inizializzando i parametri addestrabili dei vari strati in modo che abbiano distribuzione gaussiana a varianza unitaria migliora i tempi di convergenza anche sensibilmente [32].

Con l'obbiettivo di rendere meno dipendente la prestazione della rete neurale dall'inizializzazione dei parametri scelta, è stata introdotta la **batch normalization**. Si tratta di una tecnica che introduce nella rete una normalizzazione delle attivazioni appena prima degli strati di attivazione. L'utilizzo di questo tipo di stato introduce nuovi parametri addestrabili e non addestrabili [89].

3.5 Addestramento di reti neurali

Introduzione.
Come detto in sezione 3.4, lo sviluppo dell'architettura di rete è agevolato grazie alla disponibilità di dedicati framework open-source. Tuttavia, il codice che manipola i dati in ingresso, compone gli strati del modello desiderato, definisce gli iper-parametri ed imposta la procedura di training e di test deve essere accuratamente implementato, essendo fortemente problema-specifico. Lo scopo di questa sezione, allora, è quello di analizzare le varie procedure che è necessario programmare quando si va ad implementare l'addestramento di un modello profondo.

Pre-processing dei dati.
Prima di presentare i vari esempi disponibili alla rete, molto spesso è preferibile includere uno step di pre-processing dei dati. Consideriamo un dataset formato da esempi matriciali $\boldsymbol{X}$. È allora possibile operare le seguenti operazioni di pre-processing:

- **Centratura dei dati.** È la forma più comune di pre-processing [32]. Implica la sottrazione della media di ogni singolo esempio $\boldsymbol{X}$ da sé stesso. Possiamo esprimere matematicamente tale operazione attraverso la (3.31):

$$\boldsymbol{X} = \boldsymbol{X} - \mu_{\boldsymbol{X}} \tag{3.31}$$

dove $\mu_{\boldsymbol{X}}$ indica la media dei valori scalari che compongono il singolo esempio $\boldsymbol{X}$. In realtà, il risultato cambia a seconda della dimensione rispetto alla quale viene calcolata la media. Nel caso specifico di trial EEG $\boldsymbol{E}$ di dimensione $c \times t$, si sottrae ad ogni canale EEG (cioè ad ogni riga) la propria media temporale (operata cioè lungo le colonne) [4].

- **Normalizzazione.** In questo caso, i dati di ogni esempio vengono normalizzati in modo che siano tutti descritti dalla stessa scala di valori [32]. Una modalità comune è la normalizzazione rispetto alla deviazione standard (standardizzazione), in modo che tutti gli esempi siano a varianza unitaria:

$$\boldsymbol{X} = \frac{\boldsymbol{X} - \mu_{\boldsymbol{X}}}{\sigma_{\boldsymbol{X}}} \tag{3.32}$$

dove $\sigma_{\boldsymbol{X}}$ indica la deviazione standard dei valori scalari che compongono il singolo esempio $\boldsymbol{X}$. Nel caso di una matrice EEG $\boldsymbol{E}$ la normalizzazione viene applicata, come per la centratura, calcolando la deviazione standard sui segnali temporali che compongono ogni canale (viene cioè operata ancora una volta lungo le colonne) [4].

- **Exponential moving standardization.** Si tratta di una tecnica di standardizzazione che ha il vantaggio di poter essere eseguita anche online [4]. Si consideri un segnale discreto $x[n]$ che è ottenuto in real-time. Ogni nuovo valore di $x[n]$ viene standardizzato utilizzando media e deviazione standard di un numero fissato di campioni predenti n_p. Indicando tali quantità rispettivamente con μ_{n-1} e σ_{n-1}, il segnale standardizzato $x'[n]$ è dato dalla (3.33):

$$x'[n] = \frac{x[n] - \mu_n}{\sqrt{\sigma_n^2}} \tag{3.33}$$

dove le quantità μ_n e σ_n^2 sono calcolate come segue:

$$\mu_n = \frac{1}{n_p} x[n] + \frac{n_p - 1}{n_p} \mu_{n-1} \tag{3.34}$$

$$\sigma_n^2 = \frac{1}{n_p} (x[n] - \mu_n)^2 + \frac{n_p - 1}{n_p} \sigma_{n-1}^2. \tag{3.35}$$

Scelta degli iper-parametri.

Prima di scegliere il modello da utilizzare, è bene definire alcuni iper-parametri che ne influenzano la struttura e il comportamento, come ad esempio il dropout rate. Altri iper-parametri sono il learning rate, il numero di epoche e il batch size, che merita un discorso dedicato.

Scelta del batch size.

In sezione 3.2 abbiamo visto l'algoritmo di addestramento di un MLP. In quel caso, i parametri del modello erano modificati dopo aver presentato alla rete ciascun pattern del training set [47][48]. Questo processo è detto **Stochastic Gradient Descent (SGD)**, o anche online gradient descent [32][48].

Tuttavia, vi sono alcuni dataset che sono formati da milioni di esempi, ed un aggiornamento dei pesi simile sarebbe computazionalmente inefficiente [32]. Per ovviare a questo problema, si utilizza un altro tipo di discesa del gradiente, ovvero la **Mini-batch Gradient Descent (MGD)** [32][48]: il training set viene suddiviso in gruppi più piccoli detti mini-batch (o più semplicemente batch) e i valori del gradiente sono accumulati in variabili temporanee (una per ciascun peso) [48]; l'aggiornamento dei pesi avviene solo quando tutti i pattern di un mini-batch sono stati processati [32][48].

La dimensione dei batch (**batch size**) è un iper-parametro, ma è molto comune deciderlo a priori, anche in base ai vincoli imposti dalla memoria della GPU; valori tipici sono appartenenti alla potenza del due (32, 64, 128 ecc.). Se il batch size è pari al numero di esempi del training set, allora si parla di **Full-batch Gradient Descent (FGD)** [32][48]. Più il batch size è grande, meno la discesa del gradiente sarà stocastica, in quanto "mediata" su un numero grande di esempi [32].

Scelta del modello.

A questo punto si può procedere scegliendo il modello di rete che si vuole addestrare. Come detto, la rete deve essere costruita concatenando vari strati, fra cui quelli visti i sezione 3.4.

Inizializzazione dei parametri.

Come già spiegato in più punti di sezione 3.4, l'inizializzazione è un argomento delicato ancora in fase di studio [32]. Si ricorda comunque che è fortemente consigliato inizializzare i parametri addestrabili in modo che abbiano distribuzione gaussiana con valor medio nullo e varianza unitaria [32][87][89].

Scelta della funzione costo.

Un punto chiave nella progettazione di una rete neurale profonda è la scelta di una funzione costo adeguata. Si è già visto in sezione 3.2 che storicamente per l'MLP si minimizza l'errore quadratico medio e come funzione costo. In effetti, negli algoritmi di DL è comunque possibile utilizzare funzioni costo del tutto analoghe a quelle impiegate in altri altri modelli parametrici: infatti, come è stato possibile intuire parlando della discesa del gradiente, addestrare una rete neurale è un problema di minimizzazione volto alla ricerca del vettore dei parametri ottimale $\boldsymbol{\vartheta}_{opt}$, che sarà formato da tutti i parametri addestrabili della rete che garantiscono la minimizzazione della funzione costo.

In seguito si farà ricorso del termine l per indicare il valore della funzione costo totale. Esso è in genere formato un termine che tiene conto dello scostamento tra la predizione della rete neurale e l'etichetta vera (per l'apprendimento supervisionato) ed un termine di regolarizzazione pesato opportunamente. In questa tesi, tuttavia, l'unica tecnica di regolarizzazione utilizzata è il dropout. Questa tecnica, come accennato precedentemente, non viene introdotta esplicitamente nella funzione costo totale, bensì sotto forma di una modifica del processo di addestramento disattivando i neuroni con una certa probabilità. Quindi, in seguito ci si concentrerà solamente sul primo termine.

Per risolvere problemi di classificazione, è comune utilizzare la **cross-entropia** come funzione costo. Il concetto di entropia è un concetto chiave nell'ambito dalla teoria dell'informazione. L'entropia di una variabile aleatoria $\underline{\boldsymbol{x}}$ può essere interpretata come una **misura media di impredicibilità della variabile aleatoria.** Quanto più una variabile è impredicibile, tanto più è l'informazione guadagnata dalla sua osservazione [47]. Estendendo il concetto di entropia a due variabili $\underline{\boldsymbol{x}}$ ed $\underline{\boldsymbol{y}}$, si può dire che **l'entropia congiunta (cross-entropia) è una misura dell'impredicibilità legata all'osservazione delle due variabili aleatorie** [47]. Tanto più la cross-entropia sarà maggiore, quanto più, osservando la variabile $\underline{\boldsymbol{x}}$, sarà impredicibile il comportamento della variabile $\underline{\boldsymbol{y}}$.

Nel caso di un modello parametrico, la cross-entropia viene valutata tra la distribuzione target "vera" $p(\underline{\boldsymbol{x}})$ che viene fornita dal supervisore e la distribuzione stimata dal modello $q(\underline{\boldsymbol{x}}; \underline{\boldsymbol{\vartheta}})$ [32]. Dunque, il vettore dei parametri ottimo $\boldsymbol{\vartheta}_{opt}$ è quello per cui la cross-entropia fra queste due distribuzioni è minimizzata.

A partire dal valore dell'uscita i-esima $\underline{y}_i = f_i(\underline{\boldsymbol{x}}; \underline{\boldsymbol{\vartheta}})$, è possibile calcolare la probabilità che l'ingresso $\underline{\boldsymbol{x}}$ appartenga alla classe i-esima ω_i applicando la **funzione di attivazione softmax** (equazione (3.36)):

$$q_i(\underline{\boldsymbol{x}}, \underline{\boldsymbol{\vartheta}}) = \frac{\exp(f_i(\underline{\boldsymbol{x}}, \underline{\boldsymbol{\vartheta}}))}{\sum_j \exp(f_j(\underline{\boldsymbol{x}}, \underline{\boldsymbol{\vartheta}}))}. \tag{3.36}$$

La funzione softmax mappa il valore del neurone d'uscita i-esimo y_i in un nuovo valore q_i che, per la (3.36), potrà appartenere solamente all'intervallo [0;1]. Questo nuovo valore q_i rappresenta la probabilità che l'input $\boldsymbol{x}$ appartenga alla classe i-esima ω_i [32].

Si supponga ora che il modello debba classificare fra n classi (rete con n neuroni d'uscita) e si indichi con Ω_i l'insieme degli esempi $\boldsymbol{x}$ appartenenti l'i-esima classe ω_i. L'entropia congiunta per due distribuzioni discrete è data dalla (3.37):

$$h_i(p_i, q_i) = -\sum_{\boldsymbol{x} \in \Omega_1}^{\Omega_n} p_i(\underline{\boldsymbol{x}}) \ln\left[q_i(\underline{\boldsymbol{x}}, \underline{\boldsymbol{\vartheta}})\right]. \tag{3.37}$$

Tuttavia, per ogni esempio in ingresso $\boldsymbol{x}$ il supervisore conosce la distribuzione vera $p_i(\underline{\boldsymbol{x}})$ (equazione (3.38)):

$$p_i(\underline{\boldsymbol{x}}) = \begin{cases} 1 & \iff \boldsymbol{x} \in \Omega_i \\ 0 & \text{altrimenti} \end{cases}. \tag{3.38}$$

Dunque la (3.37) si riduce alla (3.39):

$$h_i(\underline{\boldsymbol{x}}, \underline{\boldsymbol{\vartheta}}) = -\ln\left[q_i(\underline{\boldsymbol{x}}, \underline{\boldsymbol{\vartheta}})\right]. \tag{3.39}$$

Si può allora esprimere la funzione costo per l'i-esima uscita l_i sostituendo la (3.36) nella (3.39), ottenendo così la (3.40):

$$l_i(\underline{\boldsymbol{x}}, \underline{\boldsymbol{\vartheta}}) = -\ln\left[\frac{\exp(f_i(\underline{\boldsymbol{x}}, \underline{\boldsymbol{\vartheta}}))}{\sum_j \exp(f_j(\underline{\boldsymbol{x}}, \underline{\boldsymbol{\vartheta}}))}\right]. \tag{3.40}$$

3.6 Reti neurali per la decodifica del movimento

Le feature estratte con reti convoluzionali.

Ad oggi, c'è un crescente interesse nell'uso delle CNN anche per l'analisi del segnale EEG. Tuttavia, poco si sa ancora a proposito di quali siano le architetture migliori e quali le caratteristiche che tali reti sono in grado di estrarre per segnali di questo tipo [4]: mentre l'FBCSP+LDA – l'attuale gold standard per BCI di decodifica del movimento [27] – è progettato per utilizzare le bande frequenziali di maggior interesse, le feature estratte dalle CNN non sono fissate a priori.

Un recente studio che approfondisce queste tematiche è quello del 2017 ad opera di Schirrmeister et al. [4]. In questo articolo si sono confrontate due pipeline di ML e DL sullo stesso dataset per la classificazione del movimento (con FBCSP+LDA e CNN rispettivamente). Inoltre, sono state studiate quali caratteristiche vengono apprese dalla rete sui dati attraverso studi di correlazione delle uscite degli strati nascosti-strato di uscita. Lo studio ha dimostrato che le reti proposte hanno imparato a selezionare le bande frequenziali alpha, beta ed high-gamma, che, come si è visto nel Capitolo 1, sono i ritmi cerebrali più fortemente correlati al movimento.

L'architettura Deep ConvNet.

Le architetture convoluzionali per la decodifica del movimento proposte in letteratura sono di dimensioni piuttosto modeste se confrontate con quelle impiegate per la computer vision: per esempio, l'architettura ResNet (rete per la classificazione di immagini) [90] è composta da ben 152 strati, mentre l'architettura DeepConvNet (uno dei modelli più profondi per la classificazione di segnale EEG) ha solo 5 strati convoluzionali ed 1 denso.

Deep ConvNet è una delle due architetture proposte da Schirrmeister et al. [4] per la decodifica del movimento tramite segnale EEG. Come si può vedere da figura 3.13, tale modello è formato da 5 blocchi principali: i primi quattro sono blocchi **Conv-ELU-Pool** (strato convoluzionale, attivazione ELU e max pooling), l'ultimo blocco è invece lo strato denso che implementa la classificazione fra 4 classi.

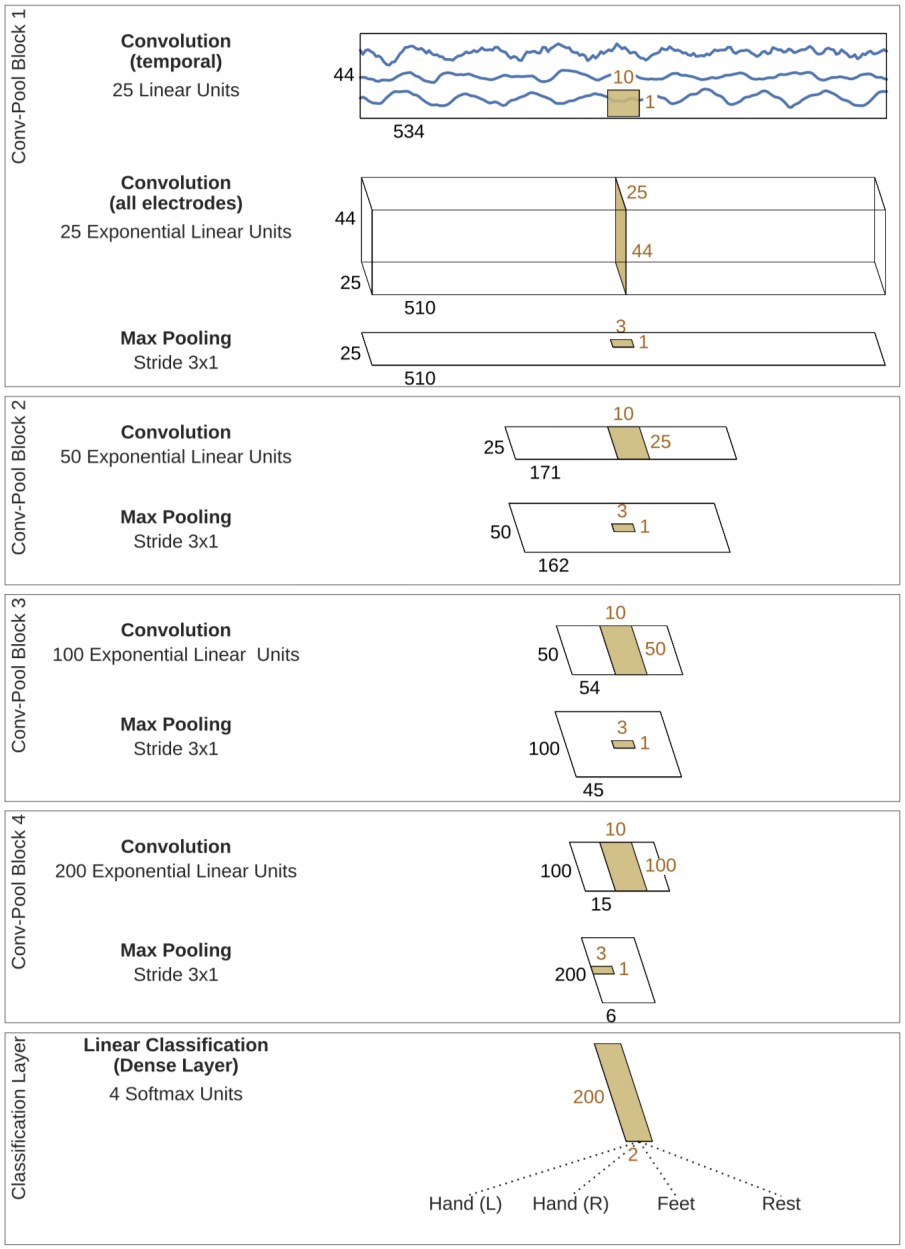


Figura 3.13: architettura Deep ConvNet. Partendo dall'alto, l'input EEG viene progressivamente trasformato, fino all'ultimo strato denso con quattro neuroni di output, uno per ciascuna delle quattro classi possibili. I cuboidi bianchi dai contorni neri rappresentano gli strati di input o le mappe di attivazione (uscite dei blocchi convoluzionali o di pooling); i cuboidi marroni rappresentano invece i kernel; le dimensioni corrispondenti sono indicate in nero e marrone, rispettivamente. In seguito ad ogni convoluzione viene indicato sulla sinistra il tipo di attivazione utilizzata (ELU o softmax). Infine, si noti che le proporzioni fra le mappe di attivazione e dei kernel sono solo approssimative [4].

I blocchi che alternano strato convoluzionale, strato di attivazione e strato di pooling (Conv-ReLU-Pool o Conv-ELU-Pool) sono molto comuni nella computer vision, in cui molte reti sono progettate andando a porre semplici blocchi di questo tipo in cascata [32]. Infatti, Schirrmeister et al. dichiarano nel loro articolo che hanno progettato questa architettura in modo che fosse semplice ma allo stesso momento in grado di estrarre molte feature [4]. L'unico elemento di questa architettura che si discosta da tale semplicità e dagli standard della computer vision classica è il primo blocco, il quale, come si vede da figura 3.13, è costituito da due strati convoluzionali in cascata:

- **Primo strato convoluzionale.** Si considerino le dimensioni dello stato di input: esso sarà una matrice $c \times t$, con c numero di canali e t numero di campioni temporali (si rammenti figura 2.5). Lo strato di input ha quindi dimensione $c \times t \times 1$. Inoltre si utilizzeranno di seguito le dimensioni espresse numericamente, facendo riferimento a figura 3.13; per cui si dirà, per esempio, che lo strato di input ha dimensione $44 \times 534 \times 1$.

  Fatte queste premesse, si considerino i 25 kernel $1 \times 10 \times 1$ che vengono convoluti all'ingresso, con convoluzione 2-D, in corrispondenza del primo strato convoluzionale. Si nota subito che l'altezza di ogni kernel è unitaria, quindi esso scorre lungo tutta la dimensione temporale (in orizzontale) di un unico canale; il risultato della convoluzione di un unico kernel sarà allora un filtraggio temporale dei singoli canali. Al termine dello scorrimento di tutti e 25 i kernel si ottiene una mappa di attivazione di dimensione $44 \times 510 \times 25$, ovvero 25 filtraggi temporali differenti. Questa operazione è del tutto analoga all'applicare un banco di 25 filtri al segnale originale, con la differenza che nel caso dell'FBCSP i parametri del filtro sono degli iper-parametri (quindi impostati a priori), mentre in questo caso sono dei parametri addestrabili, coincidendo con i coefficienti dei 25 kernel.

- **Secondo strato convoluzionale.** Il kernel di questo strato ha dimensioni $44 \times 1 \times 25$, dunque si estende "perpendicolarmente" alla dimensione temporale. Perciò scorrerà sul tensore in direzione temporale, operando il filtraggio spaziale ricercando caratteristiche tra i diversi canali e nella mappa di attivazione calcolata precedentemente. In questo modo, anche l'algoritmo CSP viene emulato.

Per completare le osservazioni a proposito di questa architettura di rete, rimangono da considerare alcuni strati che non sono esplicitamente rappresentati in figura 3.13, ovvero gli strati di batch normalization e di dropout. Come abbiamo visto in sezione 3.4 infatti, questi ed altri strati (come quelli di attivazione) non sono effettivamente formati da neuroni, quindi non introducono nuovi parametri addestrabili (batch normalization a parte, si veda il paragrafo specifico in sezione 3.4). Per questa ragione, nei diagrammi che rappresentano le architetture di rete vengono spesso trascurati. In particolare, viene applicata la batch normalization prima di ogni strato di attivazione (come da prassi per questa tecnica), strato denso escluso (sul quale invece viene applicata la funzione softmax). Inoltre, a valle di ogni pooling è posto uno strato di dropout. Si può osservare l'esatta sequenza degli strati utilizzati nella tabella 4.5 di Capitolo 4.

Strategie di training.

Oltre all'architettura di rete da utilizzare, un'altra importante scelta da fare riguarda le modalità di addestramento. Le strategie di training sono fortemente influenzate dalla natura dei dati con cui si ha a che fare; nel caso del **segnale EEG**, si è già detto nel Capitolo 1 che tale segnale è **altamente soggetto-specifico**. Questo è sufficiente a spiegare perché, nella maggior parte dei paper scientifici sull'argomento, il classificatore (sia esso di ML o di DL) venga addestrato con un approccio **Within-Subject (WS)** (ovvero ripetendo l'addestramento separatamente per ogni soggetto, così da avere un classificatore addestrato in modo specifico per il singolo soggetto).

Per quanto riguarda i modelli di ML, si è già visto che il gold standard di FBCSP+LDA viene addestrato utilizzando un approccio **trial-wise** (addestrando cioè il modello sul ogni trial disponibile utilizzato come una matrice di input). La carenza di linee guida circa la migliore strategia di addestramento da implementare con algoritmi di DL ha portato allo studio in questo elaborato di tesi di 5 differenti tipologie di training:

1. **Trial-wise WS.** Come detto poco sopra, si tratta dell'addestramento più semplice, in cui un classificatore è addestrato con i trial provenienti da un unico soggetto. È anche la modalità di training più diffusa in letteratura e quella che, in virtù dell'alta soggetto-specificità del segnale EEG, consente di ottenere i risultati migliori.

2. **Trial-wise WS con differenti finestre temporali.** Del tutto analogo al precedente, ne differisce solamente per il fatto di andare a finestrare i vari trial per addestrare il classificatore solo su un intervallo temporale più ristretto rispetto al precedente. In questo modo è possibile, ripetendo l'addestramento, individuare quali siano gli intervalli temporali più rilevanti per addestrare una rete neurale nel compito oggetto di studio con buone prestazioni.

3. **Cropped training WS.** Questa strategia di addestramento esegue **data augmentation**, un metodo di regolarizzazione particolarmente adatto a piccoli dataset. Quando la disponibilità di esempi per il training è ridotta, il modello tende velocemente a lavorare in regime di overfitting. Per regolarizzare sistemi del genere è utile allora introdurre nuovi esempi andando a campionare e modificare in un qualche modo quelli già a disposizione [32].

   Nel caso delle immagini, ad esempio, quelle già disponibili possono essere ruotate, filtrate, tagliate, occluse, ecc. Nel caso del segnale EEG, a partire da trial di lunghezza temporale pari a t si vanno a creare tante versioni del segnale con lunghezza t' minore di t (**crop**) [4].

   È possibile formare crop del segnale originale in molte modalità diverse, tuttavia qui di seguito si definirà quella implementata in questa tesi. Il numero di crop che è possibile estrarre da un singolo trial è stato fatto dipendere da due iper-parametri: il **crop size** e il **crop stride**. Il crop size definisce il numero di campioni temporali di cui ciascun crop sarà composto; il crop stride definisce invece il passo (in termine di campioni temporali) con cui si estrae un nuovo crop. Il numero di crop ottenibili sarà dato allora dalla (3.41):

$$\text{n. crop} = \text{ceil}\left(\frac{\text{n. campioni} - \text{crop size} + 1}{\text{crop stride}}\right) \tag{3.41}$$

   dove ceil è la funzione che implementa l'arrotondamento per eccesso e dove n. campioni indica il numero di campioni temporali di cui è composto un trial.

4. **Cross-subject learning (CL).** Si tratta di una strategia di addestramento **Between-Subject (BS)**, cioè che utilizza dati provenienti da soggetti diversi. In particolare, il classificatore è addestrato con la totalità degli esempi provenienti da tutti i soggetti tranne uno, i cui dati verranno utilizzati interamente come test set. Ripetendo tale addestramento isolando volta per volta un soggetto diverso, è possibile stimare la performance media del classificatore addestrato su tutti i soggetti a disposizione quando deve classificare un soggetto mai visto prima [91].

5. **Transfer learning (TL).** È un metodo di addestramento che va di pari passo con il precedente. Una volta addestrato un classificatore su tutti i soggetti disponibili tranne uno ed aver stimato la performance sul soggetto non considerato, si può inizializzare il modello con i parametri addestrabili appresi in quest'ultimo e proseguire l'addestramento con pochi esempi del nuovo soggetto. In questo modo si riesce ad osservare se e di quanto migliora la performance rispetto alla baseline di CL mostrando alla rete neurale un numero limitato di dati del nuovo soggetto [48]. Si parla di "transfer learning" perché si tenta di trasferire la conoscenza appresa in precedenza su un nuovo dataset, costituito tipicamente da un numero limitato di dati [32].

   Inoltre, nelle applicazioni classiche di computer vision è pratica comune **"congelare" alcuni strati**, partendo dallo strato di input in avanti (il numero di strati congelati è un nuovo iperparametro). Congelare uno strato significa rendere impossibile la modifica dei suoi parametri, preservando intatte le feature apprese sui dati osservati prima della nuova procedura di training. Gli strati che rimangono addestrabili saranno gli unici che potranno apprendere nuove feature [32].

In questo elaborato di tesi si sono sviluppate e confrontate tutte queste modalità di addestramento, cosa che ha permesso di valutare le applicazioni più promettenti dell'utilizzo del segnale EEG per la decodifica del movimento. Questo lavoro fornisce una visione d'insieme più ampia di quanto non fosse stato fatto dal paper di riferimento di Schirrmeister et al. [4]. Rispetto a quest'ultimo, si sono infatti sviluppati tutti gli addestramenti between-subject (CL e TL), utilizzando inoltre differenti finestre temporali.

Capitolo 4

Materiali e metodi

4.1 High-Gamma dataset

Il dataset.

High-Gamma dataset (Schirrmeister et al.[4]) è un dataset acquisito in un laboratorio ottimizzato per la registrazione non invasiva delle componenti EEG ad alta frequenza (banda high-gamma), le quali, come visto nel Capitolo 1, hanno una forte correlazione con il movimento volontario [4][7][92][93][94].

Il dataset è acquisito tramite 128 elettrodi, tutti riferiti a Cz, con frequenza di acquisizione di 5 kHz. I dati provengono da 14 soggetti sani (di cui 9 di sesso femminile, 4 mancini, età media di 27.5 anni) e per ognuno di essi sono stati acquisiti 1000 trial motori circa (si veda tabella 4.1 per il numero esatto per ogni soggetto e per ogni classe).

I task motori si dividono in quattro classi: **(1) riposo** (nessun movimento); **(2) movimento dei piedi**; **(3) movimento della mano destra**; **(4) movimento della mano sinistra**. Si veda il prossimo paragrafo per una descrizione più dettagliata dei task motori.

Nel lavoro originale, il dataset di ciascun soggetto è stato suddiviso in modo che gli ultimi 160 trial formassero il test set, i rimanenti il training set. Tuttavia, in questo lavoro di tesi i due insiemi di test e di training sono stati ricongiunti per poter eseguire cross-validazione a k fold.

	classi				
subj	**rest**	**piedi**	**mano dx**	**mano sx**	**totale**
1	120	120	120	120	480
2	243	244	244	242	973
3	260	260	260	260	1040
4	264	265	264	264	1057
5	220	220	220	220	880
6	260	260	260	260	1040
7	260	260	260	260	1040
8	203	203	204	204	814
9	260	260	260	260	1040
10	260	260	260	260	1040
11	260	260	260	260	1040
12	260	260	260	260	1040
13	240	240	240	240	960
14	260	260	260	260	1040
totale	3370	3372	3372	3370	13484

Tabella 4.1: numero di trial per ogni soggetto e classe nell'High-Gamma dataset. L'abbreviazione "subj" sta per subject ed indica l'identificativo del soggetto.

Modalità di acquisizione.
I soggetti sono stati fatti sedere in maniera confortevole nel mezzo di una stanza poco illuminata ed elettromagneticamente isolata. Le impedenze di elettrodo sono state ridotte al di sotto dei 5 $k\Omega$ usando gel elettrolitico applicato mediante una cannulla non affilata.

I segnali visivi per indicare ai soggetti quale task motorio eseguire sono stati presentati usando un monitor collocato al di fuori della stanza, visibile attraverso una finestra elettromagneticamente schermata. Il soggetto è stato istruito affinché la distanza dei proprio occhi dal display fosse mantenuta a circa 1 m. Un punto fisso è stato posto al centro dello schermo. Il soggetto doveva fissare tale punto ed eseguire il task motorio mantenendo il capo e il resto del corpo il più fermi possibile. Il task da eseguire veniva indicato sullo schermo tramite una freccia grigia mostrata su sfondo nero. Ad ogni direzione della freccia era associato uno dei quattro task possibili: **(1) se la freccia puntava verso l'alto**, il soggetto doveva rilassarsi rimanendo immobile; **(2) se la freccia puntava verso il basso**, il soggetto doveva stringere ripetutamente le punte delle dita dei piedi; **(3) se la freccia puntava verso destra o (4) verso sinistra**, il soggetto doveva premere ripetutamente con le dita della mano destra o sinistra, rispettivamente (finger tapping).

I task motori sono stati scelti in modo da richiedere la minor attività muscolare possibile durante il movimento. Ogni trial aveva una durata di 4 secondi ed il soggetto era istruito affinché eseguisse il movimento al proprio ritmo finché la freccia era mostrata sul display. L'ordine di presentazione dei segnali era pseudo-random, con tutte e quattro le classi eseguite ogni quattro trial.

4.2 Materiali

Software.
Il codice di questa tesi è stato interamente scritto in Python [95], utilizzando Keras [86] come framework di DL e Braindecode per caricare ed operare alcune operazioni di pre-processing sul dataset di ciascun soggetto. Braindecode è un framework per maneggiare i segnali EEG sviluppato da Schirrmeister et al. [4]. Il software così implementato è stato processato da un sistema basato su Linux Ubuntu Desktop ed adeguatamente impostato per sfruttare l'accelerazione grafica fornita dalle istruzioni di basso livello del pacchetto CUDA di NVIDIA [85]. Il codice è organizzato in una repository su GitHub sotto il nome di **hgdecode**. La repository è liberamente accessibile al link seguente: `https://github.com/davidemiani/hgdecode`.

sofware	**versione**
Ubuntu Desktop	18.04.2
CUDA	9.0
Python	3.6
Tensorflow-gpu	1.12.0
Keras	2.1.6
Braindecode	0.4.7

(a) versione dei sofware utilizzati. Fare riferimento a tali versioni se si volessero replicare i test di questa tesi con hgdecode.

	modello	**specifiche principali**
CPU	Intel(R) i5-8500	6/6 core/thread @4.0 GHz
MB	Asus H370	16 linee pci-e
RAM	XPG ADATA (4x)	32 GB DDR4 @2600 MHz
GPU	NVIDIA RTX 2080	8 GB GDDR6 @7000 MHz

(b) specifiche tecniche principali del personal computer utilizzato.

Tabella 4.2: software e hardware utilizzati per questo elaborato di tesi.

Hardware.

Gli algoritmi sviluppati sono stati eseguiti su un personal computer le cui specifiche sono riportate in tabella 4.2b. Per quanto l'FBCSP+LDA non sia particolarmente oneroso a livello computazionale, le DNN necessitano di hardware potente – soprattutto lato grafico – e soluzioni di dissipazione congeniali ad una macchina che deve eseguire calcoli per giorni interi. Per queste ragioni, si è optato per un processore di fascia media affiancandolo ed una delle più recenti schede grafiche NVIDIA, così che fosse possibile utilizzare l'accelerazione hardware dei Tensor Core, dei quali si è già parlato nel Capitolo 2.

Una problematica affrontata in corso d'opera è stata la gestione della memoria RAM per eseguire il CL. Come si è visto nel Capitolo 3, per questa metodologia di training è necessario addestrare il classificatore su tutti i dati di tutti i soggetti tranne uno, che verrà utilizzato come test set. L'intero dataset High-Gamma è formato da circa 25 GB di dati EEG e dunque, compatibilmente con la RAM disponibile, è stato necessario implementare codice computazionalmente efficiente nella gestione della memoria.

4.3 Metodi

Algoritmi implementati.

In questa tesi sono state implementate tutte le strategie di training viste nel Capitolo 3, sia per pipeline di ML (TL escluso) che per pipeline di DL. Tali metodologie sono poi state utilizzate per l'addestramento dell'FBCSP+LDA e dell'architettura Deep ConvNet, rispettivamente.

Svantaggi del cropped training e relativo abbandono.

Dal momento che dividere in crop aumenta di molto la dimensione del dataset, aumentano anche i tempi di addestramento. Nell'articolo di Schirrmeister et al. [4], la rete è stata addestrata sia con approccio trial-wise che facendo uso di crop su due bande differenti. Tuttavia, non si è osservato un beneficio significativo del cropped training sulla banda utilizzata in questo elaborato. Perciò, aumentando il tempo macchina senza effettivi vantaggi, si è deciso di non utilizzare questa modalità di training.

Iper-parametri globali.

Per iper-parametri globali, si intendono quegli iper-parametri che sono rimasti inalterati in tutte le prove effettuate.

Iper-parametri globali comuni fra ML e DL.

- **Canali EEG:** dei 128 canali disponibili, sono stati utilizzati solo i 44 più prossimi alle aree motorie.
- **Frequenza di ricampionamento:** il segnale è stato sottocampionato 250 Hz per questioni di efficienza computazionale.

Iper-parametri globali di FBCSP+LDA.

- **Iper-parametri del blocco FB:** si veda tabella 4.3.
- **Numero di feature spaziali:** ne sono state utilizzate 10, le prime e le ultime 5 della matrice dei pattern spaziali.
- **Tipologia di pre-processing:** centratura dei dati.

	banco 1	**banco 2**
f_{min} **(**Hz**)**	0.5	10
f_{max} **(**Hz**)**	12	122
banda passante (Hz**)**	6	8
overlap (Hz**)**	3	4
tipo di filtro	Butterworth	
ordine del filtro	3	

Tabella 4.3: iper-parametri globali del blocco FB.

Iper-parametri globali di Deep ConvNet.

Oltre a quelli indicati in tabella 4.4, si è utilizzata l'exponential moving standardization con finestra di 1000 campioni come pre-processing.

frazione di validazione	0.1
kernel convoluzionale 1	$1 \times 10 \times 25$
kernel convoluzionale 2	$44 \times 1 \times 25$
convoluzionale 3	$1 \times 10 \times 50$
convoluzionale 4	$1 \times 10 \times 100$
convoluzionale 5	$1 \times 10 \times 200$
attivazione strati convoluzionali	ELU
attivazione strato denso	softmax
funzione costo	cross-entropia

Tabella 4.4: iper-parametri globali per l'addestramento del modello DNN.

A chiusura di questo paragrafo si riporta che, in accordo con quanto osservato da Schirrmeister et al, le performance peggiorano significativamente – sia per FBCSP+LDA che per Deep ConvNet – utilizzando tecniche di pre-processing diverse da quelle qui indicate.

blocco	**tipo di strato**	**dimensione output**	**n. parametri**
1	input	$44 \times 1125 \times 1$	0
1	conv 2-D	$44 \times 1116 \times 25$	275
1	conv 2-D	$1 \times 1116 \times 25$	27525
1	batch norm	$1 \times 1116 \times 25$	100
1	attivazione	$1 \times 1116 \times 25$	0
1	max pooling	$1 \times 372 \times 25$	0
1	dropout	$1 \times 372 \times 25$	0
2	conv 2-D	$1 \times 372 \times 25$	12550
2	batch norm	$1 \times 363 \times 50$	200
2	attivazione	$1 \times 363 \times 50$	0
2	max pooling	$1 \times 121 \times 50$	0
2	dropout	$1 \times 121 \times 50$	0
3	conv 2-D	$1 \times 112 \times 100$	50100
3	batch norm	$1 \times 112 \times 100$	400
3	attivazione	$1 \times 112 \times 100$	0
3	max pooling	$1 \times 37 \times 100$	0
3	dropout	$1 \times 37 \times 100$	0
4	conv 2-D	$1 \times 28 \times 200$	200200
4	batch norm	$1 \times 28 \times 200$	800
4	attivazione	$1 \times 29 \times 200$	0
4	max pooling	$1 \times 9 \times 200$	0
4	dropout	$1 \times 9 \times 200$	0
5	flatten	1800	0
5	denso	4	7204
5	attivazione	4	0
Parametri totali: 299'354 **Parametri addestrabili:** 298'604 **Parametri non addestrabili:** 750			

Tabella 4.5: tabella riassuntiva degli strati, dimensioni d'uscita e numero dei parametri dell'architettura Deep ConvNet ricreata in questo lavoro di tesi. Rispetto al modello originale di figura 3.13 si hanno delle dimensioni di output da ciascuno strato differenti: questo è dovuto al fatto che nell'articolo originale è stata utilizzata una strategia cropped-training, con ingresso $44 \times 534 \times 1$. Una modifica allo strato di input comporta una modifica in cascata anche a tutti gli strati inferiori. Tuttavia, l'architettura rimane la stessa.

Iper-parametri specifici.

Per iper-parametri specifici si intendono quegli iper-parametri che variano con le strategie di training utilizzate.

	finestra (*ms*)	dropout rate	frozen index	learning rate	batch size	epoche	fold/ fold size
WS	-500÷4000	0.5	0	1e-4	32	1000	2/4/6/8/10/12
WS	-1000÷1000	0.5	0	1e-4	64	1000	12
WS	-1500÷500	0.5	0	2e-5	64	1000	12
CL	-500÷4000	0.5	0	1e-4	32	800	-
CL	-1000÷1000	0.5	0	1e-4	64	800	-
TL	-500÷4000	0.6	0	2e-5	4	100	4
TL	-500÷4000	0.6	0	2e-5	8	200	8
TL	-500÷4000	0.6	0	2e-5	16	500	16
TL	-500÷4000	0.6	0	2e-5	32	500	32
TL	-500÷4000	0.6	0	2e-5	64	500	64/128
TL	-1000÷1000	0.6	0	2e-5	4	100	4
TL	-1000÷1000	0.6	0	2e-5	8	100	8
TL	-1000÷1000	0.6	0	2e-5	16	150	16
TL	-1000÷1000	0.6	0	2e-5	32	250	32
TL	-1000÷1000	0.6	0	2e-5	64	400	64/128
TL	-500÷4000	0.6	1	2e-5	64	200	128
TL	-500÷4000	0.6	2	2e-5	64	200	128
TL	-500÷4000	0.6	3	2e-5	64	200	128
TL	-500÷4000	0.6	4	2e-5	64	200	128
TL	-500÷4000	0.6	5	2e-5	64	200	128
TL	-500÷4000	0.6	-1	2e-5	64	200	128
TL	-500÷4000	0.6	-2	2e-5	64	200	128
TL	-500÷4000	0.6	-3	2e-5	64	200	128
TL	-500÷4000	0.6	-4	2e-5	64	200	128
TL	-500÷4000	0.6	-5	2e-5	64	200	128
TL	-500÷4000	0.6	6/-6	2e-5	64	200	128

Tabella 4.6: addestramenti eseguiti con i relativi iper-parametri specifici. Per WS, l'ultima colonna riporta il numero di fold con cui si è eseguita la cross-validazione; per TL riporta invece la dimensione della fold di training. Infatti nel TL la cross-validazione risulta invertita: la partizione più grande va al test set, la più piccola al train. Nel TL, le fold contengono esattamente lo stesso numero di trial per ogni classe. Per quanto riguarda il significato di **indice di congelamento (frozen index)** si veda il risultato R9 di Capitolo 5.

Capitolo 5

Risultati

Indicizzazione dei risultati.

I risultati ottenuti sono indicizzati tramite un identificativo formato dalla lettera "R" seguita da un codice numerico progressivo.

Metrica di performance utilizzata.

La metrica di performance riportata nei vari risultati è l'accuratezza, espressa in percentuale. Anche i valori medi e le deviazioni standard sono sempre riferiti all'accuratezza percentuale.

Definizione di curva di apprendimento.

Le performance di un algoritmo di ML possono variare, anche considerevolmente, con l'insieme dei dati che si utilizza per addestrarlo. Più dati si forniscono all'algoritmo in fase di training, maggiori saranno le probabilità di raggiungere performance più elevate. Chiaramente, la capacità di un algoritmo di migliorarsi va incontro ad un inevitabile limite superiore, imposto in parte dalla logica stessa dell'algoritmo, in parte dagli iper-parametri utilizzati, in parte dalla bontà dei dati di training.

Una modo per caratterizzare l'andamento della metrica di performance al variare della numerosità è tracciarne la **curva di apprendimento**: si ripete l'addestramento registrando la metrica di performance ottenuta in funzione del numero di esempi di training.

Ordine di esposizione.

L'ordine con cui i risultati sono esposti ne rispecchia lo sviluppo:

R1. Si è partiti da una cross-validazione a 8 fold non stratificata per verificare che l'insieme degli iper-parametri scelti – sia per FBCSP+LDA che per Deep ConvNet – garantisse un'accuratezza comparabile ai risultati ottenuti da Schirrmeister et al. [4]. Infatti, il numero di esempi che confluiscono nel training set quando lo si divide in 8 partizioni, è confrontabile con quello utilizzato nell'articolo di riferimento. Si veda Capitolo 6 per un confronto approfondito.

R2. Superata questa prima fase di verifica, si è introdotta la stratificazione, che ha consentito di raggiungere performance migliori. Perciò, dal risultato R2 in avanti si è sempre utilizzata cross-validazione stratificata.

R3. Si è dunque valutata la curva di apprendimento mediante cross-validazione con 2, 4, 6, 8, 10 e 12 fold. Quest'ultima suddivisione dei dati si è rivelata la preferibile.

R4. Si riportano i risultati ottenuti con la partizione migliore (12 fold).

R5/R6. Sono stati condotti alcuni esperimenti utilizzando finestre temporali sempre più spostate verso intervalli precedenti all'inizio del movimento, con l'intento di valutare se i classificatori fossero in grado di estrarre ed utilizzare con successo caratteristiche dei segnali appartenenti ad intervalli diversi.

R7/R8. Si riportano i risultati ottenuti con CL e TL, utilizzando un numero crescente di campioni di training (4, 8, 16, 32, 64 e 128) selezionando le finestre temporali dei segnali compatibilmente con i risultati ottenuti precedentemente.

R9. Si è ripetuto il TL sulla finestra principale ($-500 \div 4000\ ms$) andando congelare progressivamente strati diversi.

R10. Si riportano i tempi macchina di ogni test.

R11. Confronto globale fra FBCSP+LDA e Deep ConvNet con paired t-test.

R1 Within-subject, -500÷4000 *ms*. Cross-validazione non stratificata a 8 fold.

FBCSP+LDA. In tabella 5.1, i risultati relativi ad addestramento trial-wise within-subject con cross-validazione **non** stratificata a 8 fold sulla finestra temporale $-500 \div 4000$ ms.

	fold									
subj	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**mean**	**std**
1	90.0	93.3	98.3	98.3	86.7	85.0	85.0	88.3	90.6	5.13
2	68.0	70.5	83.6	70.2	74.4	69.4	73.6	71.9	72.7	4.56
3	92.3	93.1	93.1	93.1	93.1	91.5	97.7	90.0	93.0	2.05
4	98.5	97.7	94.7	88.6	97.0	97.7	97.0	97.7	96.1	3.02
5	89.1	96.4	99.1	98.2	98.2	97.3	96.4	98.2	96.6	2.97
6	86.9	91.5	92.2	89.9	90.7	87.6	93.8	87.6	90.0	2.32
7	85.0	81.9	89.8	95.3	81.1	93.7	95.3	87.4	88.7	5.38
8	86.3	91.2	89.2	83.3	90.2	89.2	93.1	75.2	87.2	5.32
9	77.7	86.9	82.3	90.8	91.5	91.5	96.2	88.5	88.2	5.47
10	86.9	84.4	85.2	83.6	66.1	83.5	84.3	79.3	81.7	6.21
11	89.2	94.6	91.5	96.9	94.6	94.6	89.2	94.6	93.2	2.65
12	86.2	93.8	94.6	96.2	89.2	92.3	95.4	87.6	91.9	3.54
13	81.7	88.3	71.7	83.3	90.0	85.0	75.0	81.7	82.1	5.82
14	96.9	90.0	98.5	98.5	99.2	98.5	93.1	53.1	91.0	14.6
							media totale		88.8	4.93

Tabella 5.1: risultati FBCSP+LDA.

Deep ConvNet. In tabella 5.2, i risultati relativi ad addestramento trial-wise within-subject con cross-validazione **non** stratificata a 8 fold sulla finestra temporale $-500 \div 4000$ ms.

	fold									
subj	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**mean**	**std**
1	93.3	91.7	98.3	96.7	98.3	96.7	91.7	90.0	94.6	3.09
2	91.0	91.0	85.2	91.7	89.3	91.7	87.6	83.5	88.9	2.95
3	95.4	90.0	99.2	93.1	97.7	96.2	94.6	92.3	94.8	2.79
4	93.2	94.7	97.7	97.0	95.5	96.2	94.7	96.9	95.7	1.41
5	94.5	93.6	95.5	90.9	92.7	90.0	95.5	99.1	94.0	2.69
6	88.5	85.3	81.4	89.9	86.0	87.6	77.5	89.9	85.8	4.07
7	87.4	87.4	89.8	92.9	90.6	86.6	92.1	90.6	89.7	2.17
8	90.2	83.3	88.2	88.2	91.2	85.3	85.1	86.1	87.2	2.52
9	97.7	97.7	96.2	96.9	92.3	93.8	93.1	93.1	95.1	2.10
10	90.2	94.3	93.4	94.3	93.4	93.4	92.6	95.0	93.3	1.38
11	90.0	96.2	91.5	96.2	90.8	96.2	92.3	86.2	92.4	3.36
12	95.4	94.6	94.6	96.2	97.7	97.7	96.9	91.5	95.6	1.93
13	95.8	92.5	92.5	91.7	92.5	92.5	94.2	95.0	93.3	1.38
14	96.2	53.8	90.8	73.1	90.0	69.2	58.5	94.6	78.3	15.7
							media totale		91.3	3.40

Tabella 5.2: risultati Deep ConvNet.

R2 Within-subject, -500÷4000 *ms*. Cross-validazione stratificata a 8 fold.

FBCSP+LDA. In tabella 5.3, i risultati relativi ad addestramento trial-wise within-subject con cross-validazione stratificata a 8 fold sulla finestra temporale $-500 \div 4000\ ms$.

	fold									
subj	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**mean**	**std**
1	83.3	91.7	90.0	85.0	93.3	98.3	88.3	91.7	90.2	4.44
2	75.8	72.6	73.2	66.7	72.5	80.8	69.2	70.0	72.6	4.06
3	96.2	93.2	95.5	93.9	95.3	93.0	91.4	92.2	93.8	1.59
4	97.7	97.0	97.0	96.2	98.5	97.7	93.9	96.2	96.8	1.30
5	96.4	96.4	94.6	98.2	98.1	98.1	97.2	97.2	97.1	1.14
6	93.2	90.9	90.7	92.2	88.3	90.6	92.2	93.0	91.4	1.50
7	94.5	88.3	90.6	88.3	89.1	92.2	88.7	83.9	89.4	2.94
8	88.5	95.2	93.3	78.4	87.0	91.0	94.0	87.0	89.3	5.06
9	93.2	89.4	82.6	86.4	89.8	88.3	87.5	89.1	88.3	2.85
10	83.1	80.5	82.8	87.7	70.2	75.0	78.3	85.0	80.3	5.29
11	90.2	92.4	95.5	96.2	93.8	93.8	95.3	92.2	93.7	1.89
12	93.2	93.9	97.0	98.5	94.5	89.8	92.2	89.1	93.5	3.02
13	87.5	85.0	83.3	80.0	84.2	80.0	77.5	85.0	82.8	3.12
14	96.2	94.7	95.5	96.2	99.2	97.7	96.1	98.4	96.7	1.44
							media totale		89.7	2.83

Tabella 5.3: risultati FBCSP+LDA.

Deep ConvNet. In tabella 5.4, i risultati relativi ad addestramento trial-wise within-subject con cross-validazione stratificata a 8 fold sulla finestra temporale $-500 \div 4000\ ms$.

	fold									
subj	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**mean**	**std**
1	93.3	96.7	96.7	90.0	96.7	96.7	93.3	93.3	94.6	2.32
2	98.4	100	100	95.8	98.3	95.0	100	98.3	98.2	1.79
3	100	98.5	97.7	100	98.4	99.2	99.2	98.4	98.9	0.75
4	98.5	99.2	100	99.2	98.5	99.2	99.2	98.5	99.1	0.50
5	97.3	97.3	99.1	98.2	96.3	97.2	98.1	99.1	97.8	0.91
6	97.7	98.5	100	98.4	99.2	96.1	100	97.7	98.5	1.23
7	98.4	100	96.9	96.9	96.9	93.8	97.6	95.2	96.9	1.78
8	99.0	98.1	100	99.0	98.0	99.0	100	99.0	99.0	0.69
9	100	100	98.5	100	99.2	99.2	99.2	99.2	99.4	0.50
10	97.6	99.2	96.7	97.5	99.2	96.7	98.3	96.7	97.7	0.99
11	94.7	94.7	89.4	96.2	100	97.7	96.9	96.1	95.7	2.87
12	99.2	100	97.7	99.2	99.2	98.4	99.2	97.7	98.8	0.77
13	99.2	96.7	95.8	99.2	96.7	94.2	95.0	98.3	96.9	1.76
14	93.2	93.2	93.9	97.0	93.8	98.4	94.5	95.3	94.9	1.78
							media totale		97.6	1.33

Tabella 5.4: risultati Deep ConvNet.

R3 Within-subject, -500÷4000 *ms*. Curve di apprendimento.

Ogni punto dei grafici di figura 5.1 e 5.2 rappresenta l'accuratezza (media ± deviazione standard) tra tutti i soggetti.

FBCSP+LDA. In figura 5.1, è riportata la curva di apprendimento per addestramento trial-wise within-subject con cross-validazione stratificata a k fold sulla finestra temporale $-500 \div 4000\ ms$.

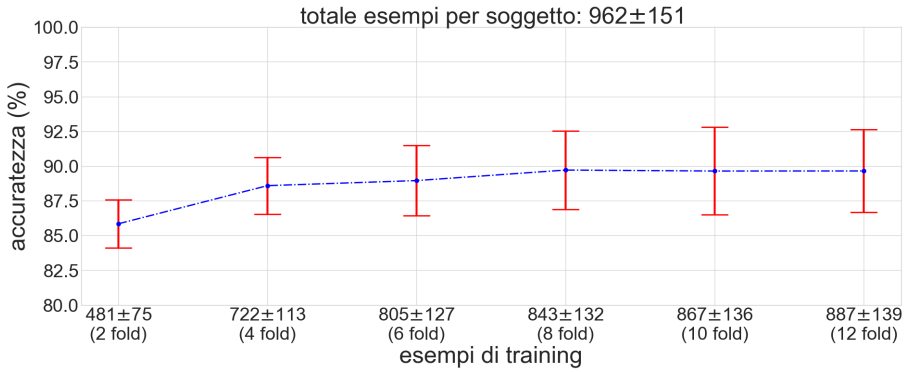


Figura 5.1: curva di apprendimento per FBCSP+LDA.

Deep ConvNet. In figura 5.2, è riportata la curva di apprendimento per addestramento trial-wise within-subject con cross-validazione stratificata a k fold sulla finestra temporale $-500 \div 4000\ ms$.

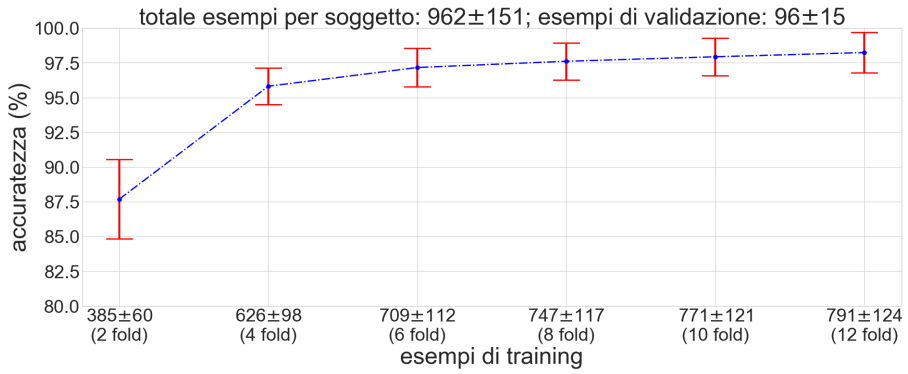


Figura 5.2: curva di apprendimento per Deep ConvNet.

R4 Within-subject, -500÷4000 *ms*. Cross-validazione stratificata a 12 fold.

FBCSP+LDA e Deep ConvNet. In tabella 5.5a e 5.5b rispettivamente, i risultati di addestramento within-subject con cross-validazione stratificata a 12 fold sulla finestra temporale $-500 \div 4000\ ms$.

subj	fold 1	2	3	4	5	6	7	8	9	10	11	12	mean	std
1	95.0	87.5	85.0	82.5	92.5	87.5	100	90.0	92.5	90.0	90.0	95.0	90.6	4.58
2	70.2	73.8	72.3	78.8	76.2	73.8	72.5	72.5	66.2	68.8	73.8	67.5	72.2	3.42
3	88.6	88.6	98.9	94.3	93.2	97.7	94.3	89.8	95.2	91.7	98.8	95.2	93.9	3.48
4	97.7	96.6	96.6	96.6	97.7	97.7	97.7	97.7	98.9	96.6	94.3	96.6	97.1	1.09
5	90.8	97.4	98.7	96.1	98.6	95.8	94.4	97.2	95.8	95.8	98.6	95.8	96.3	2.10
6	94.3	94.3	85.2	90.9	93.2	92.0	89.4	91.7	90.5	90.5	94.0	94.0	91.7	2.55
7	89.8	90.9	84.5	88.1	88.1	88.1	89.3	95.2	91.7	88.1	90.5	89.3	89.5	2.47
8	89.7	89.7	92.6	85.3	97.1	94.1	80.9	83.8	85.3	85.3	89.7	87.9	88.5	4.45
9	84.1	89.8	87.5	89.8	94.3	90.9	87.5	87.5	86.9	79.8	86.9	83.3	87.4	3.63
10	84.5	83.1	84.1	84.1	72.8	78.8	76.2	77.5	80.0	81.2	87.5	78.8	80.7	3.99
11	95.5	98.9	93.2	92.0	95.5	95.5	93.2	90.9	91.7	91.7	94.0	90.5	93.5	2.33
12	93.2	92.0	92.0	94.3	92.0	93.2	92.0	96.6	97.6	91.7	91.7	92.9	93.3	1.87
13	76.2	81.2	87.5	83.8	80.0	92.5	82.5	82.5	88.8	85.0	81.2	90.0	84.3	4.46
14	96.6	94.3	94.3	94.3	97.7	97.7	95.5	97.7	97.6	96.4	96.4	96.4	96.3	1.31
												media totale	89.6	2.98

(a) risultati FBCSP+LDA.

subj	fold 1	2	3	4	5	6	7	8	9	10	11	12	mean	std
1	95.0	100	97.5	95.0	97.5	95.0	97.5	95.0	100	95.0	97.5	100	97.1	2.00
2	97.6	98.8	98.8	100	96.2	98.8	98.8	98.8	100	100	98.8	100	98.9	1.06
3	97.7	98.9	98.9	98.9	100	98.9	100	98.9	100	98.8	97.6	100	99.0	0.79
4	100	98.9	100	100	100	94.3	97.7	100	100	100	95.5	98.9	98.8	1.88
5	97.4	100	98.7	100	100	100	98.6	98.6	100	98.6	100	100	99.3	0.86
6	97.7	97.7	97.7	97.7	98.9	100	98.8	97.6	98.8	100	100	98.8	98.7	0.91
7	95.5	100	97.6	96.4	98.8	98.8	97.6	98.8	96.4	95.2	100	95.2	97.5	1.69
8	95.6	97.1	100	98.5	97.1	91.2	100	98.5	98.5	97.1	98.5	100	97.7	2.36
9	97.7	100	98.9	98.9	98.9	96.6	100	100	97.6	100	100	100	99.0	1.13
10	98.8	100	97.6	98.8	95.1	100	97.5	97.5	98.8	96.2	100	98.8	98.2	1.47
11	96.6	94.3	98.9	96.6	98.9	96.6	97.7	100	95.2	98.8	97.6	100	97.6	1.72
12	100	100	100	98.9	97.7	100	98.9	100	100	98.8	98.8	100	99.4	0.74
13	98.8	100	97.5	97.5	95.0	100	96.2	98.8	98.8	100	100	97.5	98.3	1.56
14	92.0	97.7	96.6	96.6	97.7	97.7	96.6	94.3	96.4	96.4	91.7	95.2	95.8	1.99
												media totale	98.2	1.44

(b) risultati Deep ConvNet.

Tabella 5.5: risultati per cross-validazione stratificata a 12 fold.

R5 Within-subject, -1000÷1000 *ms*. Cross-validazione stratificata a 12 fold.

FBCSP+LDA e Deep ConvNet. In tabella 5.6a e 5.6b, i risultati di addestramento within-subject con cross-validazione stratificata a 12 fold sulla finestra temporale $-1000 \div 1000\ ms$.

subj	fold 1	2	3	4	5	6	7	8	9	10	11	12	mean	std
1	67.5	55.0	65.0	62.5	62.5	62.5	62.5	65.0	65.0	52.5	72.5	67.5	63.3	5.14
2	29.8	42.9	31.3	31.2	35.0	23.8	32.5	41.2	35.0	40.0	31.2	27.5	33.5	5.44
3	48.9	54.5	68.2	53.4	52.3	61.4	53.4	47.7	46.4	57.1	59.5	57.1	55.0	5.92
4	47.7	56.8	62.5	55.7	63.6	67.0	63.6	52.3	60.2	51.1	63.6	60.9	58.8	5.74
5	69.7	72.4	56.6	69.7	72.2	69.4	62.5	75.0	65.3	61.1	77.8	66.7	68.2	5.82
6	44.3	42.0	42.0	29.5	45.5	43.2	45.9	42.9	33.3	46.4	42.9	29.8	40.6	5.86
7	45.5	57.5	57.1	54.8	54.8	44.0	54.8	47.6	61.9	54.8	64.3	53.6	54.2	5.81
8	42.6	30.9	48.5	47.1	36.8	45.6	33.8	45.6	38.2	55.9	39.7	40.9	42.1	6.62
9	61.4	58.0	52.3	64.8	55.7	67.0	58.0	65.9	50.0	48.8	54.8	60.7	58.1	5.82
10	39.3	33.3	26.2	31.0	33.3	31.7	20.7	37.0	32.1	30.0	35.0	30.0	31.6	4.65
11	35.2	48.9	42.0	42.0	43.2	52.3	45.5	39.8	53.6	40.5	40.5	47.6	44.3	5.21
12	39.8	30.7	47.7	35.2	47.7	38.6	39.8	35.2	41.7	33.3	40.5	38.1	39.0	4.95
13	61.2	57.5	62.5	48.8	42.5	61.2	57.5	45.0	53.8	50.0	56.2	51.2	54.0	6.27
14	25.0	33.0	37.5	40.9	38.6	36.4	43.2	37.5	35.7	41.7	46.4	40.5	38.0	5.24
												media totale	48.6	5.61

(a) risultati FBCSP+LDA.

subj	fold 1	2	3	4	5	6	7	8	9	10	11	12	mean	std
1	90.0	90.0	92.5	87.5	92.5	87.5	95.0	85.0	97.5	90.0	92.5	97.5	91.5	3.74
2	95.2	90.5	96.4	92.5	93.8	98.8	90.0	85.0	85.0	92.5	91.2	91.2	91.8	3.92
3	95.5	96.6	94.3	100	97.7	97.7	97.7	93.2	98.8	97.6	98.8	97.6	97.1	1.87
4	93.2	95.5	93.2	95.5	94.3	89.8	90.9	84.1	96.6	94.3	88.6	96.6	92.7	3.57
5	86.8	97.4	88.2	92.1	87.5	97.2	93.1	91.7	93.1	95.8	94.4	97.2	92.9	3.64
6	94.3	96.6	89.8	89.8	92.0	95.5	94.1	96.4	95.2	98.8	97.6	95.2	94.6	2.73
7	93.2	92.0	90.5	91.7	97.6	91.7	94.0	86.9	96.4	92.9	84.5	96.4	92.3	3.66
8	76.5	98.5	82.4	82.4	86.8	85.3	95.6	95.6	98.5	97.1	98.5	97.0	91.2	7.61
9	94.3	87.5	86.4	94.3	96.6	93.2	96.6	93.2	89.3	91.7	94.0	89.3	92.2	3.24
10	21.4	48.8	60.7	70.2	25.0	22.0	23.2	60.5	61.7	60.0	22.5	70.0	45.5	19.9
11	94.3	71.6	93.2	78.4	56.8	20.5	73.9	69.3	52.4	65.5	91.7	65.5	69.4	19.7
12	90.9	64.8	89.8	98.9	83.0	96.6	85.2	93.2	92.9	92.9	92.9	94.0	89.6	8.57
13	96.2	95.0	97.5	96.2	93.8	95.0	95.0	95.0	97.5	96.2	98.8	96.2	96.0	1.33
14	94.3	94.3	90.9	93.2	98.9	84.1	88.6	90.9	96.4	92.9	90.5	88.1	91.9	3.80
												media totale	87.8	6.23

(b) risultati Deep ConvNet.

Tabella 5.6: risultati per cross-validazione stratificata a 12 fold.

R6 Within-subject, -1500÷500 *ms*. Cross-validazione stratificata a 12 fold.

FBCSP+LDA e Deep ConvNet. In tabella 5.7a e 5.7b, i risultati di addestramento within-subject con cross-validazione stratificata a 12 fold sulla finestra temporale $-1500 \div 500$ ms.

subj	fold 1	2	3	4	5	6	7	8	9	10	11	12	mean	std
1	15.0	27.5	22.5	20.0	27.5	27.5	32.5	30.0	32.5	32.5	15.0	23.7	25.5	6.07
2	26.2	29.8	18.1	31.2	22.5	23.8	26.2	21.2	25.0	35.0	32.5	28.8	26.7	4.75
3	31.8	34.1	31.8	29.5	30.7	35.2	26.1	26.1	20.2	20.2	31.0	32.1	29.1	4.72
4	28.4	25.0	23.9	30.7	26.1	34.1	31.8	33.0	27.3	28.4	31.8	29.9	29.2	3.09
5	28.9	35.5	27.6	23.7	38.9	16.7	30.6	34.7	34.7	22.2	27.8	22.2	28.6	6.34
6	23.9	27.3	15.9	21.6	27.3	34.1	27.1	26.2	35.7	35.7	26.2	25.0	27.2	5.55
7	19.3	29.9	21.4	20.2	21.4	32.1	20.2	16.7	27.4	34.5	27.4	28.6	24.9	5.49
8	29.4	26.5	29.4	25.0	32.4	33.8	29.4	20.6	27.9	30.9	25.0	25.8	28.0	3.51
9	23.9	28.4	26.1	33.0	23.9	34.1	20.5	35.2	25.0	25.0	23.8	34.5	27.8	4.88
10	23.8	22.6	23.8	31.0	33.3	27.7	35.8	29.6	20.0	23.8	26.2	33.8	27.6	4.84
11	26.1	34.1	29.5	29.5	25.0	29.5	33.0	21.6	38.1	40.5	22.6	23.8	29.5	5.78
12	18.2	23.9	23.9	19.3	25.0	29.5	28.4	22.7	31.0	19.0	23.8	38.1	25.2	5.49
13	31.2	30.0	32.5	31.2	36.2	23.8	32.5	28.8	28.8	26.2	26.2	25.0	29.4	3.48
14	19.3	25.0	22.7	20.5	22.7	25.0	23.9	26.1	22.6	28.6	28.6	32.1	24.8	3.50
												media totale	27.4	4.82

(a) risultati FBCSP+LDA.

subj	fold 1	2	3	4	5	6	7	8	9	10	11	12	mean	std
1	32.5	45.0	17.5	32.5	25.0	30.0	27.5	17.5	37.5	32.5	32.5	21.1	29.3	7.76
2	20.2	25.0	24.1	26.2	22.5	31.2	25.0	27.5	25.0	23.8	28.8	22.5	25.2	2.86
3	26.1	26.1	29.5	33.0	27.3	43.2	33.0	29.5	28.6	22.6	26.2	28.6	29.5	4.99
4	34.1	30.7	26.1	28.4	26.1	27.3	36.4	27.3	28.4	25.0	27.3	24.1	28.4	3.47
5	23.7	25.0	35.5	35.5	34.7	31.9	31.9	26.4	31.9	27.8	18.1	27.8	29.2	5.14
6	27.3	25.0	28.4	31.8	23.9	31.8	38.8	29.8	27.4	31.0	29.8	33.3	29.8	3.83
7	22.7	26.4	29.8	29.8	36.9	31.0	26.2	27.4	23.8	28.6	31.0	25.0	28.2	3.69
8	25.0	27.9	26.5	25.0	19.1	23.5	30.9	33.8	30.9	26.5	32.4	30.3	27.6	4.03
9	34.1	31.8	28.4	34.1	44.3	26.1	52.3	35.2	19.0	23.8	21.4	28.6	31.6	9.06
10	22.6	31.0	25.0	25.0	25.0	24.1	25.9	21.0	25.0	28.8	40.0	35.0	27.4	5.25
11	37.5	35.2	23.9	39.8	23.9	23.9	19.3	25.0	28.6	27.4	31.0	29.8	28.8	5.93
12	26.1	28.4	31.8	28.4	23.9	21.6	33.0	30.7	34.5	23.8	28.6	26.2	28.1	3.79
13	31.2	25.0	31.2	36.2	33.8	45.0	30.0	30.0	33.8	25.0	25.0	31.2	31.5	5.37
14	37.5	34.1	31.8	37.5	37.5	40.9	26.1	19.3	29.8	26.2	35.7	34.5	32.6	5.95
												media totale	29.1	5.08

(b) risultati Deep ConvNet.

Tabella 5.7: risultati per cross-validazione stratificata a 12 fold.

R7 Between-subject, -500÷4000 *ms*. TL con numero di esempi crescente rispetto alla baseline di CL.

FBCSP+LDA e Deep ConvNet. In tabella 5.8, CL e TL con fold da 4, 8, 16, 32, 64 e 128 esempi con finestra temporale $-500 \div 4000\ ms$. La variazione percentuale $\Delta_{\%}$ è calcolata rispetto alla baseline di CL di Deep ConvNet.

	cross-soggetto		transfer learning					
subj	**ML**	**DL**	**4**	**8**	**16**	**32**	**64**	**128**
1	51.5	95.6	94.7	95.0	95.3	96.0	95.6	97.3
2	45.5	67.8	77.8	78.2	79.4	80.6	83.3	87.9
3	49.2	64.7	83.0	84.2	86.2	87.6	89.4	92.2
4	40.9	68.8	89.2	90.2	90.7	91.1	92.1	95.3
5	63.1	85.6	96.1	96.3	96.3	96.5	96.2	97.6
6	45.7	67.4	70.7	72.0	72.6	74.7	77.2	84.1
7	53.7	78.7	80.7	82.1	82.8	83.7	84.6	88.6
8	40.7	45.7	88.7	89.3	90.0	90.4	90.9	94.3
9	46.0	86.1	85.2	85.9	86.9	87.8	89.0	92.9
10	46.7	67.7	73.9	75.0	76.1	77.3	79.6	84.2
11	40.9	53.2	76.5	77.4	77.8	79.5	81.9	86.5
12	46.6	78.8	86.2	87.5	88.6	89.3	91.3	94.1
13	37.4	54.8	71.5	72.9	74.7	76.3	78.7	85.4
14	52.9	85.5	90.2	90.6	91.4	92.4	94.8	97.2
mean	47.2	71.5	83.2	84.0	84.9	85.9	87.5	91.3
std	6.41	13.8	7.95	7.68	7.47	7.01	6.27	4.83
$\Delta_{\%}$	-33.9	0	16.4	17.6	18.8	20.3	22.4	27.7

Tabella 5.8: risultati comparativi fra CL e TL.

Deep ConvNet. In figura 5.3, la curva di apprendimento mediata fra i soggetti per TL all'aumentare degli esempi di training.

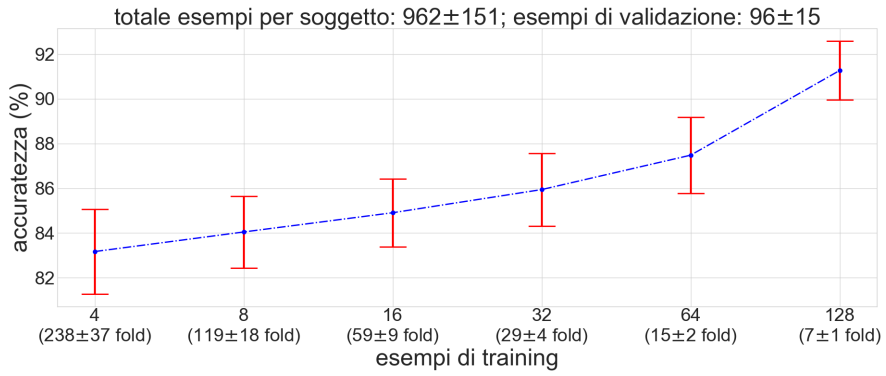


Figura 5.3: curva di apprendimento per transfer learning.

R8 Between-subject, -1000÷1000 *ms*. TL con numero di esempi crescente rispetto alla baseline di CL.

FBCSP+LDA e Deep ConvNet. In tabella 5.9, CL e TL con fold da 4, 8, 16, 32, 64 e 128 esempi con finestra temporale $-1000 \div 1000\ ms$. La variazione percentuale $\Delta_{\%}$ è calcolata rispetto alla baseline di CL di Deep ConvNet.

	cross-soggetto		transfer learning					
subj	**ML**	**DL**	**4**	**8**	**16**	**32**	**64**	**128**
1	34.2	81.0	79.5	80.7	80.9	81.5	82.1	85.5
2	30.6	59.4	65.0	65.7	66.6	68.7	70.9	73.9
3	33.1	60.9	68.9	70.0	71.3	74.3	76.0	80.7
4	26.7	57.5	61.7	63.0	63.8	65.1	66.5	71.3
5	39.3	67.8	72.4	73.1	73.6	74.5	75.6	79.2
6	34.3	60.4	58.6	59.6	60.4	62.8	65.4	71.5
7	36.0	66.3	67.5	68.4	69.0	69.7	70.3	76.3
8	27.0	57.1	74.8	75.5	75.8	76.9	78.4	82.0
9	40.1	60.4	61.2	61.9	62.8	64.8	67.1	73.8
10	25.5	40.4	46.1	46.7	46.8	47.1	48.0	51.5
11	28.8	38.1	51.3	51.8	52.6	54.0	56.1	60.7
12	34.9	63.5	68.6	69.5	70.2	71.2	72.4	74.2
13	33.9	55.6	63.0	64.6	66.0	67.4	71.3	76.5
14	29.7	68.8	74.5	75.1	75.5	76.6	79.4	81.2
mean	32.4	59.8	65.2	66.1	66.8	68.2	70.0	74.2
std	4.37	10.5	8.87	8.92	8.89	8.90	8.86	8.57
$\Delta_{\%}$	-45.8	0	9.07	10.5	11.7	14.0	17.0	24.0

Tabella 5.9: risultati comparativi fra CL e TL.

Deep ConvNet. In figura 5.4, la curva di apprendimento per TL all'aumentare degli esempi di training.

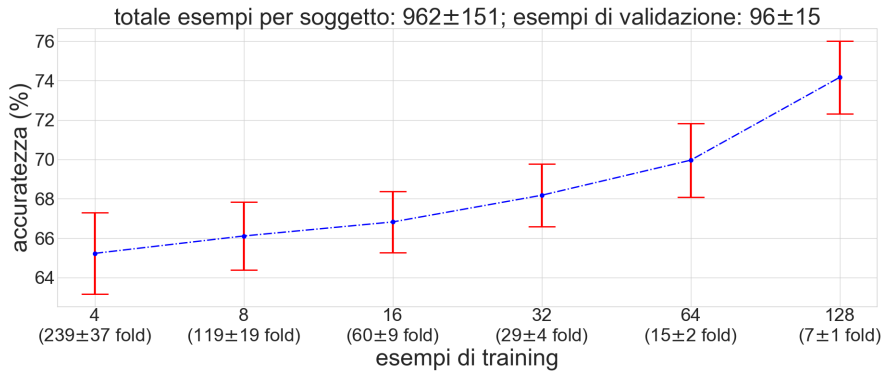
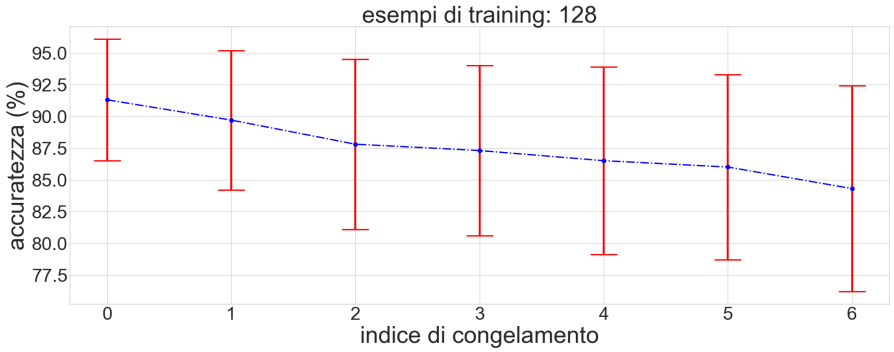


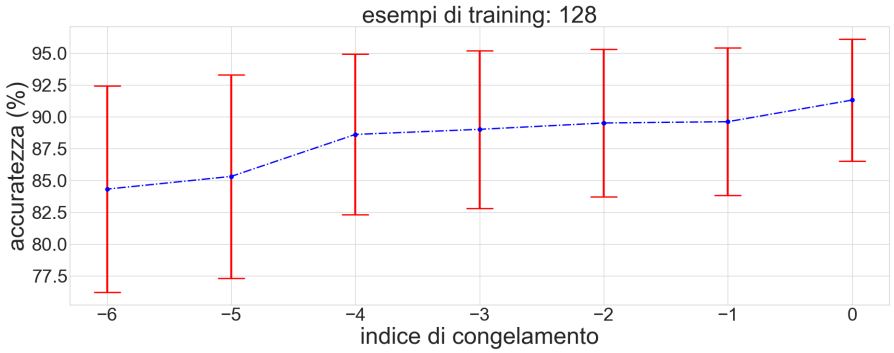
Figura 5.4: curva di apprendimento per transfer learning.

R9 Between-subject, -500÷4000 *ms*. TL con 128 esempi e variazione degli strati congelati.

Deep ConvNet. In figura 5.5 sono raffigurati gli andamenti dell'accuratezza in funzione dell'**indice di congelamento**, un iper-parametro che indica la posizione dell'ultimo strato principale (convoluzionale o denso) che si congela. Se l'indice è positivo, esso indica il numero di strati congelati a partire dallo strato di input della rete (ad es. 2 significa che sono congelati i primi due strati convoluzionali); se è negativo, indica il numero di strati congelati a partire dallo strato di output della rete (ad es. -2 significa che sono congelati lo strato denso e il penultimo convoluzionale).



(a) andamento dell'accuratezza in funzione dell'indice di congelamento. Si parte congelando il primo strato convoluzionale per poi arrivare a congelarli tutti.



(b) andamento dell'accuratezza in funzione dell'indice di congelamento. Si parte scongelando il primo strato convoluzionale per poi arrivare a scongelarli tutti.

Figura 5.5: andamento dell'accuratezza in funzione dell'indice di congelamento.

Deep ConvNet. In tabella 5.10 sono riportati i risultati ottenuti per TL con 128 esempi e finestra temporale di -500÷4000 ms al variare dell'indice di congelamento. La variazione percentuale $\Delta_{\%}$ è calcolata rispetto alla baseline di CL. Quando l'indice di congelamento è pari a ±6 si hanno prestazioni migliori rispetto a CL per il fatto che alcuni parametri della rete, al di fuori degli strati principali, rimangono comunque addestrabili.

		transfer learning con strati congelati											
subj	**CL**	**0**	**1**	**2**	**3**	**4**	**5**	**±6**	**-5**	**-4**	**-3**	**-2**	**-1**
1	95.6	97.3	97.1	96.6	95.9	97.1	95.9	95.3	95.7	96.8	97.0	97.1	97.0
2	67.8	87.9	84.7	83.0	83.0	81.5	80.0	78.1	79.3	83.5	83.5	83.7	84.7
3	64.7	92.2	90.8	90.1	89.2	88.1	86.6	81.4	84.0	88.1	89.3	89.9	90.3
4	68.8	95.3	92.4	91.5	91.4	91.0	90.7	89.9	91.9	94.0	94.1	94.1	94.4
5	85.6	97.6	97.4	97.2	97.1	96.9	97.1	95.9	97.0	97.3	97.6	97.7	97.5
6	67.4	84.1	80.8	77.3	77.4	76.1	74.7	70.9	71.3	77.2	78.5	79.6	79.6
7	78.7	88.6	87.9	85.4	85.1	85.2	83.9	81.7	82.5	85.5	85.8	87.6	88.1
8	45.7	94.3	93.5	92.4	91.5	91.4	91.5	91.0	91.0	93.3	93.9	94.0	93.4
9	86.1	92.9	90.5	89.0	88.0	87.4	86.5	85.7	86.2	89.9	90.5	90.0	90.6
10	67.7	84.2	82.0	79.2	77.6	76.2	76.8	75.2	76.1	81.1	80.9	81.1	81.2
11	53.2	86.5	84.6	83.3	82.0	81.0	80.9	79.3	80.3	83.3	84.0	85.1	84.5
12	78.8	94.1	93.3	91.2	90.9	90.3	89.4	89.0	89.9	92.8	92.6	92.5	93.3
13	54.8	85.4	83.5	77.5	77.7	74.6	75.4	73.1	74.4	82.0	82.1	84.0	83.5
14	85.5	97.2	96.7	95.5	95.2	95.1	94.6	93.6	93.8	95.9	96.0	96.4	96.5
mean	71.5	91.3	89.7	87.8	87.3	86.5	86.0	84.3	85.3	88.6	89.0	89.5	89.6
std	13.8	4.83	5.54	6.65	6.66	7.42	7.34	8.05	7.95	6.34	6.22	5.85	5.84
$\Delta_{\%}$	0	27.7	25.5	22.9	22.2	21.1	20.4	18.0	19.3	24.1	24.5	25.2	25.4

Tabella 5.10: TL su 128 esempi al variare dell'indice di congelamento.

R10 Tempo macchina misurato per ogni test.

FBCSP+LDA e DeepConvNet. In tabella 5.11, la durata dei vari test effettuati. L'ultima riga si riferisce a tutti gli addestramenti con almeno uno strato congelato.

	finestra (ms)	**epoche**	**fold/fold size**	**tempo ML** (h)	**tempo DL** (h)
WS	-500÷4000	1000	2	1.5	3
WS	-500÷4000	1000	4	3	10
WS	-500÷4000	1000	6	4	14
WS	-500÷4000	1000	8	6	26
WS	-500÷4000	1000	10	6.5	28
WS	-500÷4000	1000	12	7	40
WS	-1000÷1000	1000	12	4	21
WS	-1500÷500	1000	12	4	21
CL	-500÷4000	800	10	6	32
CL	-1000÷1000	800	12	3	14
TL	-500÷4000	100	4	-	11
TL	-500÷4000	200	8	-	11
TL	-500÷4000	500	16	-	11
TL	-500÷4000	500	32	-	5.5
TL	-500÷4000	500	64	-	3
TL	-500÷4000	400	128	-	3
TL	-1000÷1000	100	4	-	9
TL	-1000÷1000	100	8	-	4
TL	-1000÷1000	150	16	-	2.5
TL	-1000÷1000	250	32	-	1.5
TL	-1000÷1000	400	64	-	1
TL	-1000÷1000	400	128	-	1
TL	-500÷4000	200	128	-	1

Tabella 5.11: tempi di calcolo a confronto.

R11 Confronto globale fra ML e DL con paired t-test.

FBCSP+LDA e Deep ConvNet. In tabella 5.12, confronto fra i due classificatori su tutte le prove effettuate. Si considera la differenza fra i due algoritmi significativa quando $p < 0.05$.

	finestra (ms)	**FBCSP+LDA**	**Deep ConvNet**	p
WS	$-500 \div 4000$	89.6 ± 2.98	98.2 ± 1.44	5.57e-4
WS	$-1000 \div 1000$	48.6 ± 5.61	87.8 ± 6.23	5.42-8
WS	$-1500 \div 500$	27.4 ± 4.82	29.1 ± 5.08	2.77e-2
CL	$-500 \div 4000$	47.2 ± 6.41	71.5 ± 13.8	1.03e-6
TL			91.3 ± 4.83	9.45e-13
CL	$-1000 \div 1000$	32.4±4.37	59.8±10.5	5.64e-8
TL			74.2±8.57	5.89e-11

Tabella 5.12: confronto globale fra FBCSP+LDA e Deep ConvNet.

Capitolo 6

Analisi dei risultati

R1 Within-subject, -500÷4000 *ms.* Cross-validazione non stratificata a 8 fold.

Come anticipato nel Capitolo 5, si è partiti da una cross-validazione a 8 fold non stratificata per verificare che l'insieme degli iper-parametri scelti – sia per FBCSP+LDA che per Deep ConvNet – garantisse performance comparabili con quanto riportato dall'articolo di riferimento di Schirrmeister et al. [4]. I ricercatori non hanno impiegato la cross-validazione per valutare le performance, bensì hanno utilizzato un partizionamento manuale selezionando gli ultimi gli ultimi 160 trial di ogni soggetto some test set. Un simile partizionamento, in termine di numero di esempi disponibili, lo si ottiene proprio con una cross-validazione a 8 fold.

Si è dunque effettuato il tuning degli iper-parametri, facendo variare dropout rate, learning rate, batch size, numero di epoche, i parametri dei banchi di filtri, nonché le varie modalità di pre-processing. Gli iper-parametri adottati al termine di ogni procedura di tuning sono riportati nel Capitolo 4. In seguito a tale procedura, si sono raggiunti i risultati di tabella 5.1 e di tabella 5.2 per FBCSP+LDA e Deep ConvNet rispettivamente, che sono in linea con quelli riportati nel paper di riferimento.

R2 Within-subject, -500÷4000 *ms.* Cross-validazione stratificata a 8 fold.

Utilizzando lo stesso partizionamento del punto precedente, si è è introdotta la cross-validazione stratificata, i cui risultati sono riportati in tabella 5.3 (ML) e tabella 5.4 (DL).

Per entrambi i classificatori si osserva un incremento delle performance medie, che tuttavia sono significative solo nel caso di Deep ConvNet ($p = 0.0536$ per FBCSP+LDA e $p = 2.20\text{e-}4$ per Deep ConvNet). In particolare, l'accuratezza raggiunta per il metodo di DL aumenta mediamente del 6.4% rispetto a quanto trovato da Schirrmeister et al. (97.6% contro 91.2%).

R3 Within-subject, -500÷4000 *ms*.
Curve di apprendimento.

Per determinare quale fosse il miglior partizionamento per le indagini successive, si è stimata la curva di apprendimento (figura 5.1 per FBCSP+LDA e 5.2 per Deep ConvNet).

Sia k il numero di partizioni. Le differenze fra due punti consecutivi della curva si sono rivelate significative solamente fino a $k = 4$ per FBCSP+LDA ($p = 0.493$ fra $k = 4$ e $k = 6$), mentre si arriva fino a $k = 8$ per Deep ConvNet ($p = 0.0853$ fra $k = 8$ e $k = 10$). Tuttavia, per la rete convoluzionale si torna ad avere differenza significativa fra $k = 8$ e $k = 12$ ($p = 0.0339$), dunque si è preferito utilizzare $k = 12$ nei test successivi.

Un altro risultato interessante è l'andamento stesso della curva di apprendimento: l'FBCSP+LDA raggiunge un asintoto molto prima di Deep ConvNet, che sembra anzi poter aumentare ancora per $k > 12$. Tale andamento è in accordo con quanto detto nel Capitolo 3 a proposito degli algoritmi di DL, i quali beneficiano maggiormente di un dataset più ampio di quanto non faccia l'ML (figura 3.5).

R4 Within-subject, -500÷4000 *ms*.
Cross-validazione stratificata a 12 fold.

I risultati di tabella 5.5a per FBCSP+LDA non differiscono significativamente dai precedenti ($p = 0.708$ fra $k = 8$ e $k = 12$).

Per Deep ConvNet invece, come già prima osservato, vi è differenza significativa. L'accuratezza riportata in tabella 5.5b aumenta ulteriormente rispetto al risultato di Schirrmeister et al., con una divario che si assesta ora al 7% a favore dell'implementazione della rete realizzata in questa tesi (91.2% contro 98.2%).

Questi risultati confermano che un adeguato partizionamento del dataset, sia in termini di stratificazione sia in termini di numerosità di fold utilizzate, è fondamentale per ottenere buone prestazioni di classificazione.

R5 Within-subject, -1000÷1000 *ms*. Cross-validazione stratificata a 12 fold.

Fino ad ora si è considerata la stessa finestra temporale utilizzata nell'articolo di Schirrmeister et al.: fatto coincidere l'istante di zero con l'istante di onset (comparsa della freccia sul monitor), la finestra si estende da 500 ms pre-onset al termine del trial, 4000 ms, con un intervallo di osservazione di 4500 ms dunque.

In tabella 5.6a e 5.6b sono riportati invece i risultati riducendo la finestra di osservazione a 2000 ms e anticipandola in modo da coprire l'intervallo $-1000 \div 1000$ ms. Mentre l'accuratezza di FBCSP+LDA diminuisce drasticamente (da 89.6% a 48.9%), la riduzione di performance è più contenuta con Deep ConvNet, che passa dal 98.2% all'87.8%. Entrambe le differenze sono chiaramente significative (p =1.59e-9 per FBCSP+LDA e p = 1.36e-2 per Deep ConvNet). Questo risultato indica che, contrariamente all'algoritmo di ML, la rete neurale è in grado di operare una classificazione accurata partendo dai segnali estratti in finestre temporali più sbilanciate verso l'intervallo pre-movimento, all'interno del quale viene pianificata l'azione da eseguire.

R6 Within-subject, -1500÷500 *ms*. Cross-validazione stratificata a 12 fold.

Come è evidente da tabella 5.7a e 5.7b, anticipando di ulteriori 500 ms la finestra di osservazione entrambi gli algoritmi non sono più in grado di classificare correttamente il task motorio, avvicinandosi alla soglia di chance (25% per un classificatore casuale a 4 classi). Questo risultato può essere spiegato dalla limitata di porzione residua del segnale acquisito durante il movimento (solo 500 ms).

R7 Between-subject, -500÷4000 *ms*. TL con numero di esempi crescente rispetto alla baseline di CL.

Come è possibile osservare da tabella 5.8, le performance di Deep ConvNet sono significativamente maggiori di quelle di FBCSP+LDA già da un primo confronto con CL (p =1.03e-6). Introducendo poi il TL, tale differenza si fa ancora più marcata.

La curva di apprendimento per TL è tanto più ripida quanto più il training set aumenta, rivelando la natura promettente del TL su dataset EEG multi-soggetto: già con un numero di esempi di training ridotto è possibile ottenere un incremento percentuale considerevole

(+16.4% con 4 esempi di training), sino addirittura a superare – anche se di poco – il risultato di Schirrmeister in WS (91.3% in BS contro 91.2% in WS), utilizzando comunque soltanto 128 esempi.

Visti i risultati ottenuti, si può ipotizzare che, dato l'andamento della curva di addestramento, portando il numero di esempi da 128 ad un numero comparabile con quelli usati nelle partizioni con logica di apprendimento WS, si ottengano risultati migliori persino rispetto a quelli ottenuti con cross-validazione WS.

R8 Between-subject, -1000÷1000 *ms*. TL con numero di esempi crescente rispetto alla baseline di CL.

Considerazioni simili al punto precedente si possono fare anche su questa seconda finestra temporale, anche se con accuratezze minori.

R9 Between-subject, -500÷4000 *ms*.
TL con 128 esempi e variazione degli strati congelati.

Come si vede da tabella 5.10 e dalle figure 5.5a e 5.5b, in nessuno dei casi si sono registrati benefici nel congelare uno o più strati della rete.

Tuttavia, l'andamento della metrica di performance al variare dell'indice di congelamento può suggerirci quale strato sia più importante da addestrare per un soggetto specifico. Si pensi ad esempio a due strati di una rete: se uno è in grado di imparare caratteristiche più soggetto-specifiche dell'altro, congelando il primo noteremo un maggiore decremento della performance rispetto a congelare il secondo. Se invece entrambi gli strati sono congelati, scongelando il primo si otterrà un maggior incremento prestazionale rispetto a scongelare il secondo.

A livello grafico, questo può essere dedotto osservando figura 5.5a. Considerando due punti successivi, tanto maggiore sarà la variazione della metrica tanto più importante sarà lo strato congelato. Lo stesso ragionamento si applica ad una curva con logica di scongelamento, come quella di figura 5.5b.

Detto questo, è ora possibile constatare dalle figure 5.5a e 5.5b che gli strati più soggetto-specifici sono il primo, il secondo e lo strato denso. Gli altri convoluzionali introducono benefici comparabili, essendo la pendenza associata pressoché costante.

R10 Tempo macchina misurato per ogni test.

Addestrare Deep ConvNet richiede in ogni caso un tempo macchina di molto maggiore rispetto a FBCSP+LDA: in media infatti, il tempo computazionale necessario a terminare l'addestramento aumenta del 464%.

R11 Confronto globale fra ML e DL con paired t-test.

Con questa implementazione della rete, si è sempre in grado di ottenere risultati significativamente migliori con pipeline di DL che con pipeline di ML.

Conclusioni

I risultati ottenuti in questo elaborato di tesi dimostrano che, anche con un'architettura deep learning relativamente semplice quale quella del modello Deep ConvNet, è possibile, tramite l'opportuno pre-processing e partizionamento dei dati, superare significativamente il gold standard di FBCSP+LDA per la decodifica di movimenti a partire dai tracciati EEG.

Quanto detto si applica non solo alle strategie di addestramento within-subject, che in letteratura rappresentano la metodologia di training predominante per il segnale EEG, ma anche a quelle between-subject. In questo caso, il miglioramento rispetto alla baseline di machine learning è ancora più rilevante: la rete neurale addestrata tramite transfer learning si è rivelata significativamente più accurata della controparte FBCSP+LDA in entrambe le sue varianti di training between-subject e within-subject. Dunque, a partire dalla conoscenza appresa dalla rete su soggetti precedentemente analizzati, avendo a disposizione anche pochi esempi di un nuovo soggetto è possibile superare il gold standard di FBCSP+LDA addestrato invece con un numero di esempi di gran lunga maggiore.

Un altro elemento di novità rispetto al resto della letteratura scientifica, che mette nuovamente in luce la superiorità della classificazione tramite deep learning rispetto ai metodi più tradizionali di machine learning, è quello ottenuto con finestre temporali differenti del dataset. Fornendo in ingresso ai due algoritmi un segnale che solo in piccola parte contiene l'esecuzione del task motorio (cioè che contiene solo una porzione di ciascun trial), a differenza di FBCSP+LDA la rete Deep ConvNet riesce ad isolare le componenti informative utili e ad ottenere ottime performance di classificazione, addirittura comparabili a quelle di FBCSP+LDA addestrato con il segnale sull'intero trial.

Le applicazioni di un algoritmo in grado di decodificare in maniera accurata il movimento dai tracciati EEG sono molteplici. Per esempio, approcci di deep learning simili a quelli analizzati in questa tesi possono essere impiegati online per il controllo di neuroprotesi o più in generale per la neuroriabilitazione. Inoltre, vi è crescente interesse nell'utilizzo delle reti convoluzionali per lo studio dei circuiti neurali che stanno alla base del movimento: tramite tecniche di visualizzazione delle feature è infatti possibile ricavarne un'interpretazione neurofisiologica. Tali tecniche sono state impiegate, per esempio, nel mappare spazialmente le caratteristiche apprese, così da rivelare la topografia dei contributi dei vari task motori nelle diverse bande di frequenza.

Sviluppi futuri vanno di pari passo con l'avanzamento della ricerca e dell'applicazione di nuove metodologie di addestramento alle reti neurali. Ormai è certo che il futuro della tecnica e della scienza graviterà intorno all'intelligenza artificiale, per ora nella sua particolare declinazione del deep learning, e questo studio dimostra nuovamente come tale campo possa essere esteso anche alle neuroscienze. È auspicabile perciò lo sviluppo di reti sempre più sofisticate e profonde, con l'introduzione di nuovi strati e nuove strategie di training, in grado di migliorare ancora le prestazioni di classificazione.

Bibliografia

[1] Piotr Olejniczak. Neurophysiologic basis of EEG. *Journal of clinical neurophysiology*, 23(3):186–189, 2006.

[2] Federico Carpi and Danilo De Rossi. *Fenomeni Bioelettrici*. Research Center "E. Piaggio" Faculty of Engineering University of Pisa, 2013.

[3] Marc R Nuwer, Giancarlo Comi, Ronald Emerson, Anders Fuglsang-Frederiksen, Jean-Michel Guérit, Hermann Hinrichs, Akio Ikeda, Fransisco Jose C Luccas, and Peter Rappelsburger. IFCN standards for digital recording of clinical EEG. *Clinical Neurophysiology*, 106(3):259–261, 1998.

[4] Robin Tibor Schirrmeister, Jost Tobias Springenberg, Lukas Dominique Josef Fiederer, Martin Glasstetter, Katharina Eggensperger, Michael Tangermann, Frank Hutter, Wolfram Burgard, and Tonio Ball. Deep Learning with Convolutional Neural Networks for brain mapping and decoding of movement-related information from the human EEG. *arXiv preprint arXiv:1703.05051*, 2017.

[5] Alice F Jackson and Donald J Bolger. The neurophysiological bases of EEG and EEG measurement: A review for the rest of us. *Psychophysiology*, 51(11):1061–1071, 2014.

[6] COGNIONICS Inc. *Mobile-128 Hardware Description*. Consultato in data 30/01/2019 all'indirizzo: `https://www.cognionics.net/mobile-128`.

[7] Tonio Ball, Evariste Demandt, Isabella Mutschler, Eva Neitzel, Carsten Mehring, Klaus Vogt, Ad Aertsen, and Andreas Schulze-Bonhage. Movement related activity in the high gamma range of the human EEG. *Neuroimage*, 41(2):302–310, 2008.

[8] Erol Başar, Canan Başar-Eroglu, Sirel Karakaş, and Martin Schürmann. Gamma, alpha, delta, and theta oscillations govern cognitive processes. *International journal of psychophysiology*, 39(2-3):241–248, 2001.

[9] Mahtab Roohi-Azizi, Leila Azimi, Soomaayeh Heysieattalab, and Meysam Aamidfar. Changes of the brain's bioelectrical activity in cognition, consciousness, and some mental disorders. *Medical journal of the Islamic Republic of Iran*, 31:53, 2017.

[10] Lawrence M Ward. Synchronous neural oscillations and cognitive processes. *Trends in cognitive sciences*, 7(12):553–559, 2003.

[11] Gert Pfurtscheller and FH Lopes Da Silva. Event-related EEG/MEG synchronization and desynchronization: basic principles. *Clinical neurophysiology*, 110(11):1842–1857, 1999.

[12] Stuart N Baker. Oscillatory interactions between sensorimotor cortex and the periphery. *Current opinion in neurobiology*, 17(6):649–655, 2007.

[13] Patrick Ofner, Andreas Schwarz, Joana Pereira, and Gernot R Müller-Putz. Upper limb movements can be decoded from the time-domain of low-frequency EEG. *PloS one*, 12(8):e0182578, 2017.

[14] Ranjan Debnath, Virginia C Salo, George A Buzzell, Kathryn H Yoo, and Nathan A Fox. Mu rhythm desynchronization is specific to action execution and observation: Evidence from time-frequency and connectivity analysis. *NeuroImage*, 184:496–507, 2019.

[15] Giacomo Rizzolatti and Maddalena Fabbri-Destro. Mirror neurons: from discovery to autism. *Experimental brain research*, 200(3-4):223–237, 2010.

[16] Henry Gray and S Standring. *Gray's anatomy*. Arcturus Publishing, 2008.

[17] Gert Pfurtscheller. Erd and ers in voluntary movement of different limbs. *Event-related desynchronization: handbook of electroencephalography and clinical neurophysiology*, 1999.

[18] Per E Roland, Bo Larsen, Niels A Lassen, and Eric Skinhoj. Supplementary motor area and other cortical areas in organization of voluntary movements in man. *Journal of neurophysiology*, 43(1):118–136, 1980.

[19] Steven M Rao, JR Binder, PA Bandettini, TA Hammeke, FZ Yetkin, A Jesmanowicz, LM Lisk, GL Morris, WM Mueller, LD Estkowski, et al. Functional magnetic resonance imaging of complex human movements. *Neurology*, 43(11):2311–2311, 1993.

[20] JG Colebatch, MP Deiber, RE Passingham, KJ Friston, and RS Frackowiak. Regional cerebral blood flow during voluntary arm and hand movements in human subjects. *Journal of neurophysiology*, 65(6):1392–1401, 1991.

[21] Norihiro Sadato, Gregory Campbell, Vicente Ibanez, Marie-Pierre Deiber, and Mark Hallett. Complexity affects regional cerebral blood flow change during sequential finger movements. *Journal of Neuroscience*, 16(8):2693–2700, 1996.

[22] Konrad Maurer and Thomas Dierks. *Atlas of brain mapping: topographic mapping of EEG and evoked potentials*. Springer Science & Business Media, 2012.

[23] Per E Roland and Karl Zilles. Functions and structures of the motor cortices in humans. *Current opinion in neurobiology*, 6(6):773–781, 1996.

[24] Hans H Kornhuber and L Deecke. brain potential changes in voluntary movements and passive movements of humans: standby potential and reactive potentials. *Pflüger's Archive for the Entire Physiology of Man and Animals*, 284(1):1–17, 1965.

[25] Hiroshi Shibasaki and Mark Hallett. What is the bereitschaftspotential? *Clinical neurophysiology*, 117(11):2341–2356, 2006.

[26] Gert Pfurtscheller and A Aranibar. Event-related cortical desynchronization detected by power measurements of scalp eeg. *Electroencephalography and clinical neurophysiology*, 42(6):817–826, 1977.

[27] Fabien Lotte, Laurent Bougrain, Andrzej Cichocki, Maureen Clerc, Marco Congedo, Alain Rakotomamonjy, and Florian Yger. A review of classification algorithms for eeg-based brain–computer interfaces: a 10 year update. *Journal of neural engineering*, 15(3):031005, 2018.

[28] Maureen Clerc, Laurent Bougrain, and Fabien Lotte. *Brain-Computer Interfaces 1: Methods and Perspectives*. John Wiley & Sons, 2016.

[29] Gert Pfurtscheller, Gernot R Müller-Putz, Reinhold Scherer, and Christa Neuper. Rehabilitation with brain-computer interface systems. *Computer*, 41(10), 2008.

[30] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[31] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[32] Fei-Fei Li, Justin Johnson, and Serena Yeung. *CS231n: Convolutional Neural Networks for Visual Recognition*. 2017. Dispense del corso di Computer Vision della Standford University. Materiale consultabile online all'indirizzo `http://cs231n.stanford.edu/`.

[33] Pamela McCorduck. *Machines who think: A personal inquiry into the history and prospects of artificial intelligence*. AK Peters/CRC Press, 2009.

[34] Jack Clark. *Why 2015 Was a Breakthrough Year in Artificial Intelligence*. Bloomberg, 2015. Consultato in data 13/02/2019 all'indirizzo `https://www.bloomberg.com/news/articles/2015-12-08/why-2015-was-a-breakthrough-year-in-artificial-intelligence`.

[35] Molly Maskrey and Wallace Wang. Using facial and text recognition. In *Pro iPhone Development with Swift 4*, pages 285–315. Springer, 2018.

[36] Tim Dettmers. *Which GPU(s) to Get for Deep Learning: My Experience and Advice for Using GPUs in Deep Learning*. Consultato in data 13/02/2019 all'indirizzo `http://timdettmers.com/2018/11/05/which-gpu-for-deep-learning/`.

[37] Andrew Burnes. *Port Royal Benchmark Adds NVIDIA DLSS For Improved Image Quality and Performance*. Consultato in data 13/02/2019 all'indirizzo `https://www.nvidia.com/en-us/geforce/news/nvidia-dlss-3dmark-port-royal-benchmark/`.

[38] Ryszard S Michalski, Jaime G Carbonell, and Tom M Mitchell. *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.

[39] Kai Keng Ang, Zheng Yang Chin, Haihong Zhang, and Cuntai Guan. Filter bank common spatial pattern (fbcsp) in brain-computer interface. In *IEEE International Joint Conference on Neural Networks (IJCNN)*, pages 2390–2397, 2008.

[40] Gert Pfurtscheller and A Aranibar. Evaluation of event-related desynchronization (erd) preceding and following voluntary self-paced movement. *Electroencephalography and clinical neurophysiology*, 46(2):138–146, 1979.

[41] Alfons Schnitzler, Stephan Salenius, Riitta Salmelin, Veikko Jousmäki, and Riitta Hari. Involvement of primary motor cortex in motor imagery: a neuromagnetic study. *Neuroimage*, 6(3):201–208, 1997.

[42] Gert Pfurtscheller and Christa Neuper. Motor imagery and direct brain-computer communication. *Proceedings of the IEEE*, 89(7):1123–1134, 2001.

[43] Gufei Sun, Jinglu Hu, and Gengfeng Wu. A novel frequency band selection method for common spatial pattern in motor imagery based brain computer interface. In *The 2010 International Joint Conference on Neural Networks (IJCNN)*, pages 1–6. IEEE, 2010.

[44] Herbert Ramoser, Johannes Muller-Gerking, and Gert Pfurtscheller. Optimal spatial filtering of single trial EEG during imagined hand movement. *IEEE transactions on rehabilitation engineering*, 8(4):441–446, 2000.

[45] Vasilisa Mishuhina and Xudong Jiang. Feature weighting and regularization of common spatial patterns in eeg-based motor imagery bci. *IEEE Signal Processing Letters*, 25(6):783–787, 2018.

[46] Serena Morigi. Dispense del corso di Algebra e Analisi Numerica dell'Università di Bologna, 2018. Sito del corso: `https://www.unibo.it/it/didattica/insegnamenti/insegnamento/2018/434176`.

[47] Mauro Ursino and Elisa Magosso. Dispense del corso di Sistemi Neurali dell'Università di Bologna, 2017. Sito del corso: `https://www.unibo.it/it/didattica/insegnamenti/insegnamento/2017/340953`.

[48] Davide Maltoni. Dispense del corso di Machine Learning dell'Università di Bologna, 2018. Materiale consultabile online all'indirizzo `http://bias.csr.unibo.it/maltoni/ml/#`; sito del corso: `https://www.unibo.it/it/didattica/insegnamenti/insegnamento/2018/412605`.

[49] Zheng Yang Chin, Kai Keng Ang, Chuanchu Wang, Cuntai Guan, and Haihong Zhang. Multi-class filter bank common spatial pattern for four-class motor imagery bci. In *2009 Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 571–574. IEEE, 2009.

[50] Dee Unglaub Silverthorn. *Human physiology: an integrated approach*. Jones & Bartlett Publishers, 2010.

[51] Maria del Mar Quiroga. Sito didattico della faculty of medicine, nursing and health sciences, monash university., 2015. `https://ilearn.med.monash.edu.au/physiology/ActionPotentials/index.html`.

[52] Allan L Hodgkin and Andrew F Huxley. Currents carried by sodium and potassium ions through the membrane of the giant axon of loligo. *The Journal of physiology*, 116(4):449–472, 1952.

[53] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[54] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[55] F Rosenblatt. Principles of neurodynamics. *New York: Spartan*, 1962.

[56] M L Minsky and S A Papert. *Perceptrons*. MIT Press, 1969.

[57] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[58] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[59] John Hastad. Almost optimal lower bounds for small depth circuits. In *Proceedings of the eighteenth annual ACM symposium on Theory of computing*, pages 6–20. Citeseer, 1986.

[60] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[61] Jeff Dean. *Building Intelligent Systems with Large Scale Deep Learning*. 2015. Presentazione del Senior Manager del Google Brain Team; intervento e materiale consultabile all'indirizzo: `https://blog.ycombinator.com/jeff-deans-lecture-for-yc-ai/`; sito del Google Brain Team: `https://ai.google/research/teams/brain`; consultato il 21/02/2019.

[62] Bernard Widrow and Marcian E Hoff. Adaptive switching circuits. Technical report, STANFORD UNIV CA STANFORD ELECTRONICS LABS, 1960.

[63] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[64] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. 1985.

[65] Tony Robinson and Frank Fallside. A recurrent error propagation network speech recognition system. *Computer Speech & Language*, 5(3):259–274, 1991.

[66] Yoshua Bengio, Renato De Mori, Giovanni Flammia, and Ralf Kompe. Global optimization of a neural network-hidden markov model hybrid. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume 2, pages 789–794. IEEE, 1991.

[67] Lawrence K Saul, Tommi Jaakkola, and Michael I Jordan. Mean field theory for sigmoid belief networks. *Journal of artificial intelligence research*, 4:61–76, 1996.

[68] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[69] Herbert Jaeger and Harald Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *science*, 304(5667):78–80, 2004.

[70] Geoffrey E Hinton. Deep belief networks. *Scholarpedia*, 4(5):5947, 2009.

[71] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High performance convolutional neural networks for document processing. In *Tenth International Workshop on Frontiers in Handwriting Recognition*. Suvisoft, 2006.

[72] Ruslan Salakhutdinov and Geoffrey Hinton. Deep boltzmann machines. In *Artificial intelligence and statistics*, pages 448–455, 2009.

[73] Rajat Raina, Anand Madhavan, and Andrew Y Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning*, pages 873–880. ACM, 2009.

[74] Kevin Jarrett, Koray Kavukcuoglu, Yann LeCun, et al. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision (ICCV)*, pages 2146–2153. IEEE, 2009.

[75] Dan Claudiu Cireşan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. *Neural computation*, 22(12):3207–3220, 2010.

[76] Adam Coates and Andrew Y Ng. The importance of encoding versus training with sparse coding and vector quantization. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 921–928, 2011.

[77] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew. Deep learning with cots hpc systems. In *International conference on machine learning*, pages 1337–1345, 2013.

[78] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[79] Emanuela Marcelli. Dispense del corso di Organi Artificiali dell'Università di Bologna, 2017. Sito del corso: `https://www.unibo.it/en/teaching/course-unit-catalogue/course-unit/2017/384315`.

[80] David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

[81] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

[82] David H Hubel and Torsten N Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology*, 195(1):215–243, 1968.

[83] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[84] Pytorch. Sito ufficiale del framework: `https://pytorch.org/`.

[85] NVIDIA. *CUDA Parallel Computing Platform*. NVIDIA Corporation, Santa Clara, California, Stati Uniti. Sito della CUDA Zone: `https://developer.nvidia.com/cuda-zone`; consultato il 25/02/2019.

[86] François Chollet et al. Keras. `https://keras.io`, 2015.

[87] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *CoRR*, abs/1502.01852, 2015.

[88] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[89] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[90] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[91] Adam Page, Colin Shea, and Tinoosh Mohsenin. Wearable seizure detection using convolutional neural networks with transfer learning. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1086–1089. IEEE, 2016.

[92] Felix Darvas, Reinhold Scherer, Jeffrey G Ojemann, RP Rao, Kai J Miller, and Larry B Sorensen. High gamma mapping using eeg. *Neuroimage*, 49(1):930–938, 2010.

[93] Jiří Hammer, Tobias Pistohl, Jörg Fischer, Pavel Kršek, Martin Tomášek, Petr Marusič, Andreas Schulze-Bonhage, Ad Aertsen, and Tonio Ball. Predominance of movement speed over direction in neuronal population signals of motor cortex: intracranial eeg data and a simple explanatory model. *Cerebral Cortex*, 26(6):2863–2881, 2016.

[94] F Quandt, Christoph Reichert, Hermann Hinrichs, Hans-Jochen Heinze, Robert T Knight, and Jochem W Rieger. Single trial discrimination of individual finger movements on one hand: a combined meg and eeg study. *NeuroImage*, 59(4):3316–3324, 2012.

[95] Guido van Rossum. *Python: A dynamic, open source programming language.* Python Software Foundation, 1991.