ALMA MATER STUDIORUM
UNIVERSITA' DI BOLOGNA

SCUOLA DI INGEGNERIA ED ARCHITETTURA

Corso di Laurea Magistrale in
Ingegneria Biomedica

# Decodifica di intenzioni di movimento dalla corteccia parietale posteriore di macaco attraverso il paradigma Deep Learning

Tesi di laurea in
Neurofisiologia Cognitiva

Presentata da:
Luca Talevi

Relatrice:
Prof.ssa Patrizia Fattori

Correlatore:
Dott. Matteo Filippini

Sessione III
Anno Accademico 2017/2018

*All’inizio, tutto sembra falso.*
*C*

# Sommario

# RINGRAZIAMENTI

Ringrazio in primo luogo la Prof.ssa Fattori, la Prof.ssa Breveglieri e il Dott. Filippini che mi hanno dato l'impagabile possibilità di accedere ai loro laboratori ed alle loro sudatissime registrazioni di attività neurale, acquisite tramite macchinari per metà fantascientifici (e costosi, molto costosi) e per metà tenuti insieme grazie alla caparbia volontà dei ricercatori – pensavo che contesti simili esistessero solo nei vecchi film.

In particolare ringrazio Matteo, biotecnologo re-inventatosi neuroingegnere, per avermi guidato e consigliato nei punti più ostici del lavoro mentre si destreggiava tra macachi e macchinari (in più nazioni).

Ovviamente, ringrazio tutti coloro che credono nei miei sforzi, eroi invulnerabili tanto alle mie parolone esaltate quanto ai miei sproloqui più bui. Sopra ogni altra ringrazio la persona che ha pronunciato la citazione: forse non ti farà piacere leggere tutto questo, ma sappi che quella singola frase è la colonna portante che ha permesso a questa tesi di esistere e a me di persistere in tutto ciò che conosci.

Infine ringrazio i miei soci nella startup Vibre, giovani eroi (o stolti arroganti?) che hanno scelto di gettarsi nell'imprenditoria high-tech malgrado il caos del mondo del lavoro e della tecnologia, per aver sopportato i miei ritardi e malumori mattutini in periodo di tesi.

Prometto che dopo essermi laureato imparerò a dormire presto e a sentire le sveglie.

# 1. INTRODUZIONE

## 1.1. Brain Computer Interfaces

Le *Brain Computer Interfaces* ("interfacce cervello-computer", da qui in poi chiamate *BCI*) sono descrivibili come "mezzi di comunicazione diretta tra il cervello ed un dispositivo esterno". Anche se il termine ha trovato i suoi natali intorno al 1970 all'*University of California (UCLA),* la prima BCI in assoluto è nata insieme all'elettroencefalografo (EEG) inventato da Hans Berger nel 1924. Ben distante dagli apparecchi colmi di luci e di incredibili funzioni presentate dall'immaginario fantascientifico, essa consisteva in un semplice sistema di rilevazione delle cosiddette *onde alfa*, una componente delle oscillazioni elettriche associate all'attività cerebrale tanto forte da essere percepibile anche da semplici elettrodi argentati posti sul capo e collegati ad un galvanometro – il prototipo dell'EEG, appunto. Non c'era nessuna forma di controllo: la prima BCI era in effetti un prototipo dei moderni sistemi di Quantified EEG usati nelle cliniche del sonno ed usati per determinare la presenza di pattern associabili a disturbi neurologici.

Dobbiamo aspettare il 1965 per assistere al primo esempio di controllo: il compositore americano Alvier Lucier si esibì in quell'anno in un concerto solista (*"Music for Solo Performer")* dove vari strumenti percussivi erano elettronicamente guidati dall'ampiezza delle sue onde alfa, raccolte da un EEG. Un cambio paradigmatico: non più un metodo per esaminare la salute mentale di un paziente, ma un oggetto capace di donare ad un musicista l'abilità di suonare con il solo pensiero.

Negli anni 70, grazie agli studi dei pionieri Jacques e Laryce Vidal della UCLA [1], viene finalmente coniato il termine *Brain Computer Interface* ed identificato il suo obiettivo primario: il controllo di oggetti attraverso segnali cerebrali (*Figura 1*).

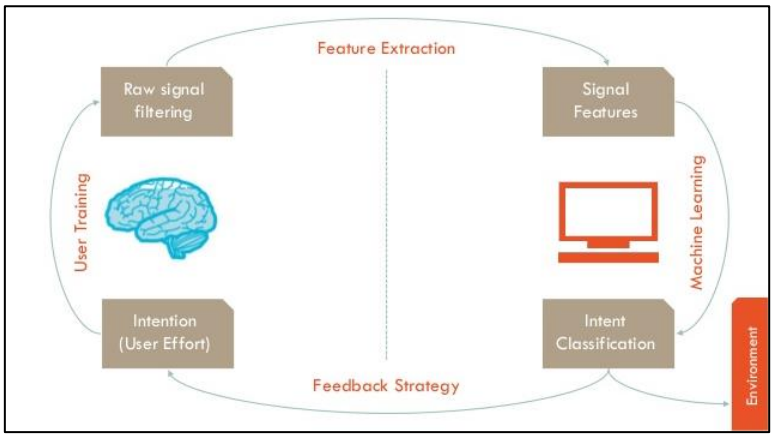


*Figura 1: il tipico schema logico di una BCI. L'intenzione dell'utente produce una variazione nei suoi segnali cerebrali, che viene rilevata da un elaboratore; quest'ultimo trasmette l'intenzione all'ambiente e produce eventualmente un feedback di conferma.*

## 1.2. Il limite dell'EEG

Grazie al crescente corpus di studi neuroscientifici e di elaborazione di segnali la ricerca sulle BCI basate su EEG è oggi più che mai fiorente: per fare qualche esempio, è oggi possibile scrivere con il pensiero [2] comandare robot telepaticamente [3], addestrare intelligenze artificiali con segnali di errore endogeni [4] o persino accelerare la riabilitazione di sopravvissuti ad ictus [5]. L'elettroencefalografo ha tuttavia un forte limite intrinseco, dipendente dal suo essere posto sulla cute.

Il segnale EEG infatti è composto dalla somma di centinaia di migliaia di attivazioni neuronali differenti, filtrate e diffuse attraverso i numerosi strati di tessuto osseo e connettivo che separano gli elettrodi dalla corteccia cerebrale. Ciò che arriva è quindi una versione decisamente distorta e riassuntiva della reale attività: le componenti sopravvissute si identificano nei *ritmi* cerebrali, di cui fanno parte le già citate onde alfa e che si figurano come indicatori dello stato mentale generale (es. sonnolenza, attenzione…) e nei *potenziali evocati*, flessioni del segnale provocate da un improvviso allineamento di massa dei neuroni in una certa zona (prevalentemente motoria, durante un movimento).

Usando l'EEG risulta quindi sostanzialmente impossibile ottenere un fine controllo mentale di oggetti.

Questo è un grave problema per una delle applicazioni apparentemente più immediate di una tecnologia capace di "leggere il pensiero": il controllo di neuroprotesi da parte di disabili.

## 1.3. Registrazioni intracorticali

Negli stessi anni in cui Hans Berger stava sperimentando il suo EEG, neurobiologici ed elettrofisiologi stavano effettuando le prime misurazioni dirette dell'attività neurale in vivo su gatti, topi e scimmie, avviando così negli anni 30 il filone delle registrazioni intracorticali [6]. Le registrazioni erano difficoltose e monocanale, incentrandosi tipicamente su un solo neurone (misurando il singolo potenziale di membrana) o su una zona interstiziale (in questo caso raccogliendo il *Local Field Potential).* Malgrado i preziosi risultati conseguiti, l'informazione contenuta nel sistema nervoso è espressa principalmente nelle connessioni tra i neuroni: le registrazioni monocanale erano quindi lontane dal fornire i mezzi per controllare un qualsivoglia dispositivo volontariamente.

Grazie all'esponenziale miniaturizzazione elettronica è divenuto possibile sperimentare i primi array multielettrodo intorno agli anni 70, portando ad importanti scoperte quali la definizione nel 1986 del *vettore di popolazione* nella corteccia motoria da parte di Georgopoulous [7]. Il primo esempio di *BCI invasiva* si ha nel 1998, quando Kennedy e Bakay riescono a far controllare uno switch on/off computerizzato ad una paziente totalmente paralizzata.

Si arriva infine al momento in cui è risultato chiaro che le BCI invasive hanno potenzialità ben più grandi di quelle basate su EEG: nel 2006 il team di Hochberg presso il Massachusetts General Hospital riesce a fornire ad un paziente tetraplegico il controllo diretto di un dispositivo neuroprotesico [8], dando inizio a quello che oggi è chiamato *progetto BrainGate* (www.braingate.org, nella *Figura 2* un famoso caso applicativo) e aprendo la strada a numerosi progetti similari finanziati da consorzi internazionali, fondi governativi ed ingenti donazioni private.

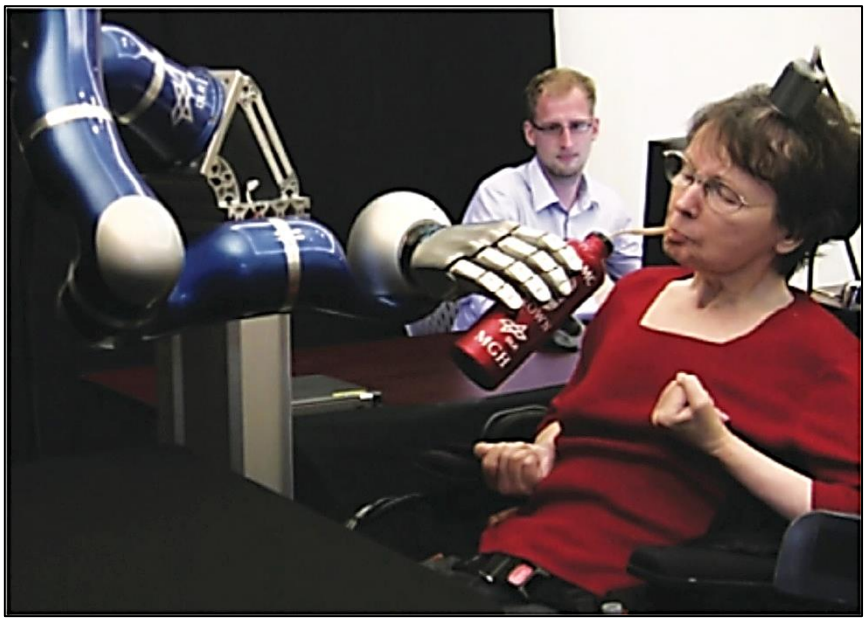


*Figura 2: Una famosa applicazione del progetto BrainGate (2012, Brown University).*

## 1.4. BCI invasive per il controllo neuroprotesico

Come il sistema *Braingate*, la maggior parte delle BCI invasive adibite al controllo neuroprotesico sviluppate finora sfruttano la codifica dei parametri motori rinvenuta da Georgopoulos nella corteccia motoria primaria. Anche se l'*encoding motorio* è ora risaputamente più complesso dell'inizialmente ipotizzato *vettore di popolazione* [9], l'espressione del segnale neurale in M1 risulta comunque fortemente correlato alla direzione del movimento. Questo rende molto semplice la costruzione di complessi sistemi di controllo diretti per neuroprotesi - come *Braingate*, appunto.

L’area M1 è tuttavia unicamente deputata al controllo diretto dei motoneuroni e risulta quindi gerarchicamente all’ultimo posto del controllo cognitivo del movimento. Un completo interfacciamento con una neuroprotesi è possibile solo con l’inserimento di un feedback sensoriale (investigato ad esempio dal team di Gaunt e Schwartz [10]) e con il coinvolgimento di aree cerebrali superiori.

La corteccia parietale posteriore (PPC) fa parte di un sistema che fa da ponte tra le aree sensoriali visive e quelle motorie, risultando quindi gerarchicamente superiore alla M1. I neuroni che ne fanno parte non possono essere strettamente classificati come sensoriali o motori: la loro funzione è di trasformazione sensomotoria e possiedono quindi entrambe le proprietà. E’ possibile intuire l’importanza della PPC nel controllo motorio considerando gli effetti delle sue lesioni: tra le più comuni l’atassia ottica, disturbo visuomotorio che consiste in grossolani errori nel cercare di raggiungere un oggetto, e l’eminegligenza spaziale, in cui il paziente non è conscio della parte di spazio controlesionale.

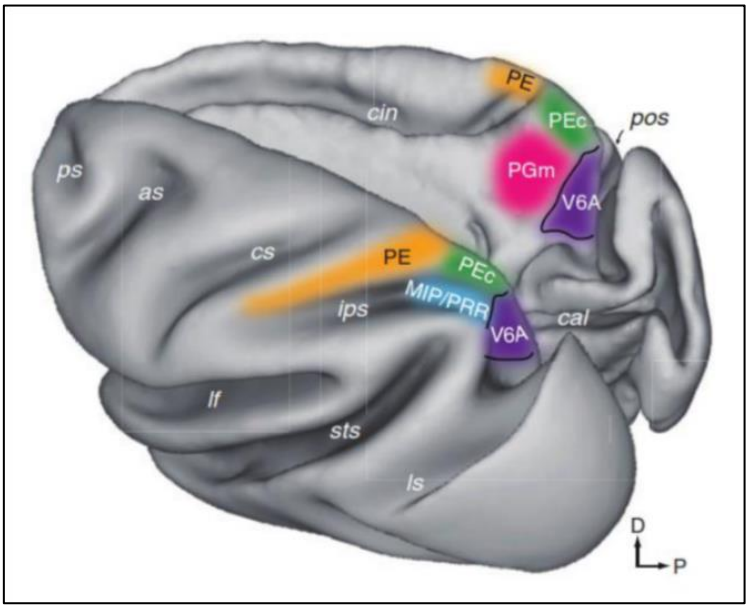


*Figura 3: rendering 3D di cervello di macaco con evidenziate alcune aree componenti la PCC facenti parte del circuito visivo dorsomediale di macaco. (Fattori et al, 2015 )*

L’area V6A fa parte della via visiva dorsale ed è localizzata nel solco parieto occipitale all’interno della PPC (*Figura 3*). Circa il 60% dei neuroni di quest’area rispondono a stimoli visivi, mentre il 30% risulta responsivo a stimoli tattili e propriocettivi prevalentemente associati ad arti superiori e torso [11] [12]. Diversi studi condotti all’interno dell’Ateneo di Bologna dimostrano che esistono in V6A delle cellule visuomotorie attivate preferenzialmente alla vista ed al movimento di *reach* e/o *grasp* verso un oggetto di una specifica forma [13] [14] [15].

Il modo in cui i neuroni di V6A codificano un particolare oggetto cambia a seconda del tipo di presa richiesto per afferrarlo, come visibile dalla *Figura 4* tratta da [15]:

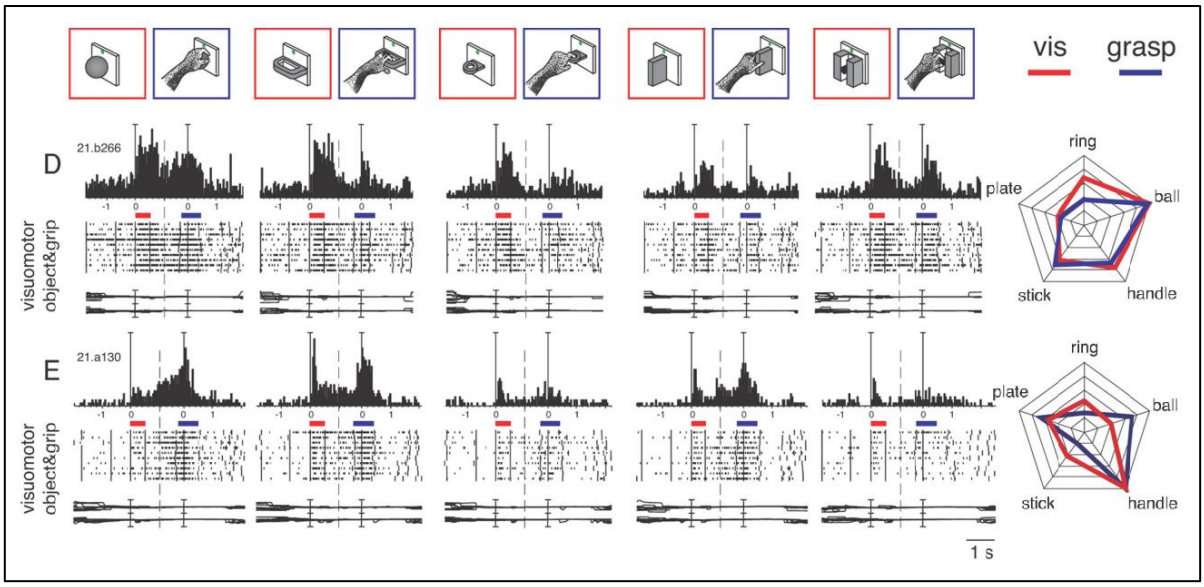


*Figura 4: cambiamento del coding dei neuroni visuomotori in V6A a seconda del tipo di presa. (Fattori et al, 2012)*

Il coding di due neuroni di esempio visuomotori, uno per riga, è rappresentato attraverso istogrammi peristimolo degli spike neuronali. Gli istogrammi rappresentano il numero di attivazioni del neurone in un dato lasso temporale (bin) sommando l'attività su diverse ripetizioni del compito (10 trials in questo caso). Gli istogrammi vengono allineati temporalmente ad eventi di interesse: in questo caso un evento visivo e l'altro motorio. Gli zeri indicano il momento in cui l'oggetto è stato visto (in rosso) e quello in cui è iniziato il movimento per afferrarlo (in blu). Notiamo sin d'ora due dettagli importanti:

1. Lo stesso neurone codifica diversamente le 2 diverse fasi per le 5 diverse prese: l'informazione dapprima puramente visiva viene progressivamente trasformata in piano d'azione, e questo si riflette sull'attività del neurone (confermando le caratteristiche sia visive che motorie del neurone).
2. A differenza dell'attivazione che segue la visione dell'oggetto, quella che corrisponde all'*onset* del movimento si palesa anche prima dell'inizio stesso dell'azione di *reaching+grasp.* Questo è una conseguenza diretta della sua superiorità gerarchica sull'area M1: dovendola modulare attraverso l'integrazione visuomotoria la sua attività inizia necessariamente prima di quella atta al controllo muscolare per il raggiungimento.

A questa si aggiunge la possibilità che tale area codifichi anche informazioni sulla posizione spaziale relativa dell'oggetto obiettivo della presa: si è infatti visto che neuroni dell'area V6A sono sensibili alla posizione del target nello spazio. [16]

Queste considerazioni sono alla base di ciò che rende interessante per lo sviluppo di BCI invasive la decodifica della V6A: comprenderla significherebbe infatti possedere la chiave per lo sviluppo di controllori capaci di preattivare e preimpostare neuroprotesi robotizzate *object-aware*, aprendo ad esempio le possibilità di :

- Sfruttare lo sviluppo tecnologico in ambito biorobotico per costruire neuroprotesi capaci di identificare il miglior movimento a seconda dell'oggetto e della sua posizione ed eseguirlo con precisione e sicurezza;
- Accoppiare il controllo PPC (anticipativo e oggetto-dipendente) con il controllo M1 (contemporaneo e regressivo) per ottenere un dispositivo totalmente controllabile e tuttavia elettronicamente stabilizzato dall'anticipazione.

Il tutto senza bisogno di telecamere esterne per il riconoscimento dell'oggetto. Unendo ai segnali estratti dall'area V6A quelli provenienti dalle aree vicine e dalla cortecce premotoria e motoria supplementare sarebbe certamente possibile ottenere risultati ancor più ragguardevoli.

## 1.5. Lo scopo della tesi

L'area V6A e la sua decodifica sono oggetto di approfondito studio da parte del gruppo della Prof.ssa Fattori, non a caso l'autrice più citata dell'ultimo paragrafo. In particolare, il Dott. Filippini è stato il primo ricercatore a tentare un approccio di decodifica automatizzata dell'attività V6A registrata da due macachi durante compiti di *reach+grasp* [17]. Malgrado la natura pioneristica del lavoro i risultati ottenuti sono stati estremamente interessanti, come visibile in *Figura 5*: l'approccio *naive bayesian* su sliding window di 300 ms con overlap di 10 ms riesce a distinguere con accuratezza prossima alla perfezione i 5 diversi tipi di oggetto/presa in entrambe le condizioni di illuminazione, mantenendo simile potenza discriminatoria per tutto il trial.

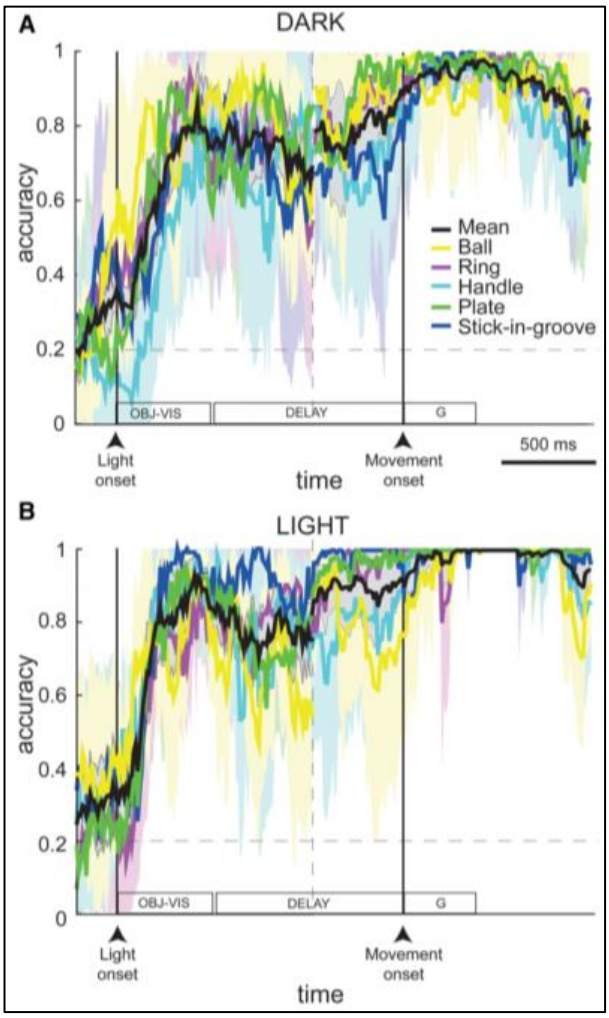


*Figura 5: Risultato dell'analisi sliding window dell'algoritmo sviluppato da Filippini in [17]*

Questo lavoro di tesi, partendo dallo stesso dataset impiegato dal Dott. Filippini, intende proseguire l'esplorazione del decoding dei 5 tipi di presa introducendo due nuovi obiettivi:

- Decodifica *quasi-real-time* della singola fase precedente il *movement-onset*, corrispondente alla nascente intenzione di movimento, distinguendo i 5 oggetti dalla condizione di assenza di intenzione
- Utilizzo di metodi ed algoritmi del paradigma *Deep Learning (DL)* specifici per le serie temporali al fine di sviluppare un *decoder* facilmente addestrabile ed implementabile in un contesto di utilizzo effettivo.

Il raggiungimento di tali obiettivi porterebbe alla costruzione di un modello capace di fornire la componente di rilevamento anticipativo ed *object-aware* necessario ai futuri sviluppi citati precedentemente; la struttura matriciale e combinatoria dei *neural networks* impiegati negli algoritmi DL ne consentirebbe inoltre l'implementazione diretta in circuiti specializzati [18] abbattendo così la latenza del decoding e potenzialmente il suo consumo energetico.

Tale ricerca è stata portata avanti in tre diversi esperimenti:

E1. Il primo esperimento ha l'obiettivo di individuare la migliore architettura neurale nel compito di riconoscimento assenza / presenza di intenzione, senza distinguere i 5 diversi oggetti. La scelta avverrà tra tre diverse alternative tratte dallo stato dell'arte, applicando alcuni basilari principi delle tecniche DL.

E2. Il secondo esperimento verificherà l'efficacia dell'architettura scelta in E1 nel compito di detezione dell'intenzione di movimento per ciascuno dei 5 oggetti. Verranno confrontate due diverse tecniche per il riconoscimento multiclasse.

E3. Il terzo ed ultimo test analizzerà la robustezza dell'algoritmo sia in condizioni single-class che multi-class, verificandone le prestazioni su dati progressivamente meno informativi e simulandone quindi le reazioni al degrado temporale spesso associato ai sistemi di registrazione intracorticali [19].

Lo scopo di questo lavoro è, sommariamente, quello di fornire un solido *benchmark* iniziale per lo sviluppo di algoritmi DL nell'ambito del *decoding real-time* di intenzioni di movimento per il controllo neuroprotesico.

# 2. BACKGROUND

## 2.1. Single Unit Recordings.

Nelle neuroscienze, le *single unit recordings* forniscono un metodo per misurare l'attività elettrofisiologica di neuroni utilizzando un microelettrodo. Quando un neurone genera un potenziale d'azione, il segnale si propaga lungo il neurone come una corrente che fluisce dentro e fuori dalla cellula attraverso regioni di membrana eccitabili nel soma e nell'assone. Il microelettrodo (*Figura 6*) viene inserito nel cervello, dove può registrare il tasso di variazione della tensione rispetto al tempo dovuto all'attività elettrica dei neuroni nell'area. I microelettrodi devono essere conduttori a punta fine e ad alta impedenza; sono principalmente micro-pipette di vetro o elettrodi di metallo di platino o di tungsteno. I microelettrodi possono essere posizionati con cura all'interno (o vicino) della membrana cellulare, consentendo la capacità di registrare intracellularmente o extracellularmente.

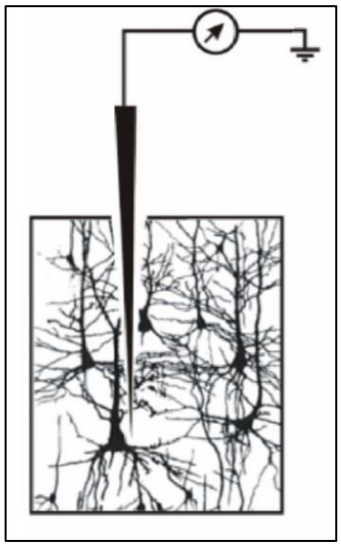

*Figura 6: microelettrodo per SUR.*

Le *single unit recordings* sono ampiamente utilizzate nelle scienze cognitive, dove consentono l'analisi della cognizione e della cartografia corticale. Questa informazione può quindi essere applicata a tecnologie BCI per il controllo cerebrale di dispositivi esterni.

I segnali registrati da un microelettrodo posto extracellularmente come in questo caso non rappresentano in realtà l'attività del solo neurone, bensì una somma di:

- *Local Field Potential*, il campo elettrico locale dovuto ad una sovrapposizione dell'attività di tutti i neuroni circostanti e del suo effetto sull'equilibrio elettrolitico del liquido extracellulare (*Figura 7, a*)
- *Spiking Activity* del/i neurone/i più vicini al microelettrodo (*Figura 7, b*)
- Rumore di fondo

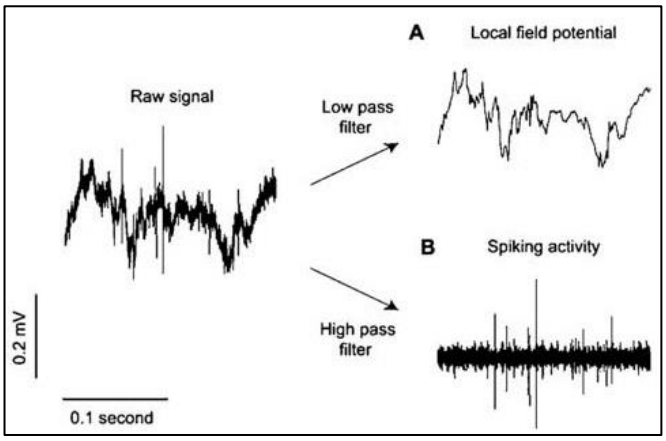


*Figura 7: le due componenti del segnale registrato dal microelettrodo: (A) il Local Field Potential (B) la Spiking Activity dei neuroni limitrofi.*

Come visibile in *Figura 7*, i due segnali sono separabili attraverso un semplice filtraggio: un filtro passa-basso evidenzia la componente LFP, mentre un passa-alto risparmia solo i picchi degli *spikes* neuronali.

Sovente il segnale che si vuole analizzare è espresso nelle attivazioni dei diversi neuroni, come nel contesto in esame; tuttavia differenziare l'attività delle cellule limitrofe all'interno della *Spiking Activity* è più complesso di un semplice filtraggio, poiché l'attività di diversi neuroni può essere descritta da frequenze molto simili.

Il processo di differenziazione, chiamato *Spike Sorting*, è solitamente svolto attraverso:

- l'assegnazione manuale dei picchi ai diversi neuroni. Si ipotizzi infatti che ogni neurone captato nelle vicinanze ha una sua forma caratteristica che ne permette la separazione dagli altri (forma d'onda rossa e blu in *Figura 8*). La classificazione per forma viene eseguita sugli eventi fuori soglia.
- L'utilizzo di tecniche automatiche quali la Principal Component Analysis, il cui operato (sempre basato sull'analisi delle forme d'onda) si può osservare a destra nella *Figura 8. I clusters* individuano i due neuroni e l'attività di fondo non separabile.

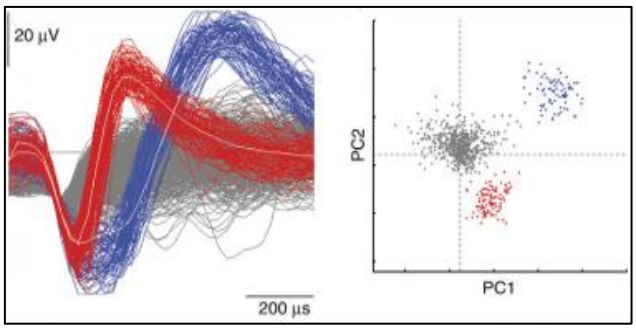


*Figura 8: Spike sorting automatico di neuroni limitrofi attraverso PCA.*

La necessità di effettuare lo *Spike Sorting* prima di proseguire con qualsiasi altra analisi complica non poco l'implementazione di algoritmi *real-time* robusti. Per questo motivo alcuni gruppi di ricerca stanno esplorando soluzioni per estrarre segnali utili direttamente dai segnali grezzi provenienti dagli elettrodi [20] [21].

L'uscita del *sorting* è generalmente un elenco di *timestamps* corrispondenti agli istanti in cui la *Spiking Activity* ha sorpassato la sogliatura definita, ciascuno assegnato ad uno degli N neuroni identificati (*Figura 9*).

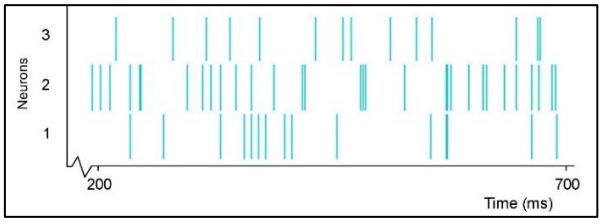


*Figura 9: Segnale in uscita dallo spike sorting.*

## 2.2. Intenzione di movimento

Il movimento non è un evento istantaneo. Il reclutamento dei muscoli richiesti per effettuare una certa azione necessità tempo, così come richiede un lasso temporale l'acquisizione dei segnali sensoriali che precedono la scelta di muoversi in un certo modo o correggere le proprie mosse.

Se la nostra capacità di muoversi dipendesse unicamente da un controllo conscio, diretto e consequenziale come quello fornito dall'area motoria primaria, saremmo sempre in ritardo ed incapaci di correggerci.

La Teoria dei Modelli Interni [22] afferma che ciascun atto motorio è il risultato di una continua, complessa integrazione di meccanismi anticipativi (*feedforward)* e correttivi *(feedback)* come mostrato nella *Figura 10* nel caso di una mano che deve tenere tra le dita una pallina bilanciandone il peso avvertito dal braccio. Tali meccanismi, chiamati appunto Modelli Interni, sono costruiti attraverso l'esperienza dell'individuo e codificati in una zona corticale appropriata per lo schema input-output di quel particolare modello.

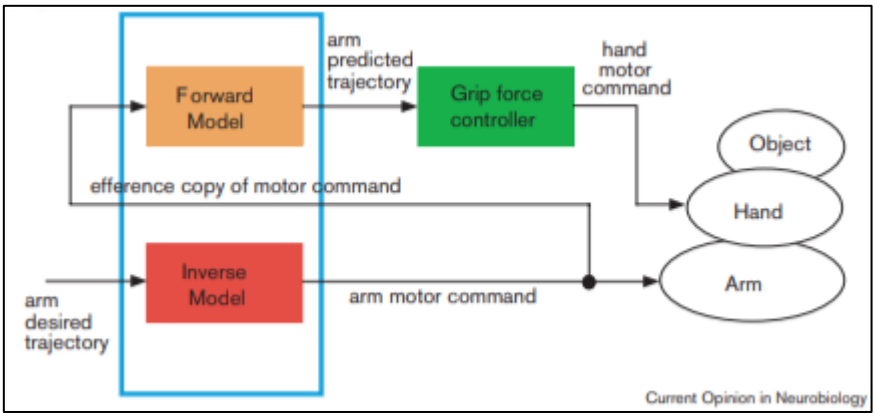


*Figura 10: Schematizzazione dei modelli interni nel caso di una mano con una pallina tra le dita.*

La teoria, oltre ad essere un utile framework per spiegare come il sistema nervoso centrale implementa i propri paradigmi di controllo ed un ottimo ponte con le discipline intorno alla ricerca sull'Intelligenza Artificiale [23], riesce a spiegare alcune capacità di risposta motoria apparentemente inspiegabili.

Un esempio è dato dalla capacità del battitore di baseball di colpire la palla nei pochi millisecondi trascorsi dal lancio: se il controllo motorio fosse unicamente causale, esso arriverebbe in risposta all'elaborazione prima visiva e poi visuomotoria – ma così facendo impiegherebbe ben più di qualche millisecondo! La soluzione è data dai modelli visuomotori interni appresi dal battitore durante gli allenamenti e le precedenti partite: il solo vedere la posizione del lanciatore gli permette di definire quasi incosciamente la traiettoria corretta della mazza da baseball.

Tali modelli visuomotori sono quasi certamente immagazzinati nella corteccia parietale posteriore, a riguardo della quale [24] [25] accompagnano le considerazioni fatte durante l'introduzione sulla presenza del coding neurale prima del *movement-onset*, e nella corteccia premotoria, la cui area 6 definita da Rizzolatti "un vocabolario degli atti motori" in [26].

Studi compiuti su EEG hanno esaminato le caratteristiche della distribuzione di potenziali elettrici emergente nel lasso di tempo precedente il *movement-onset*, chiamata *(Pre-) Movement Related Cortical Potentials* o *Bereitschafts Potential* [27]. Le ricerche in merito hanno evidenziato che tale distribuzione tende a variare la propria localizzazione nel tempo, lasciando presupporre uno scambio informativo tra diverse zone: in un lasso di tempo variabile da 0.5 a 2 secondi *pre-onset* il segnale tocca corteccia parietale posteriore, corteccia premotoria, motoria supplementare (con ordine cangiante a seconda del movimento e del contesto) per finire nella zona della corteccia motoria primaria associata ai muscoli coinvolgenti il movimento.

Nel caso specifico del movimento pianificato (*prassico*) l'attuale consenso è che la distribuzione inizi proprio nella zona parietale. [28]

Si aggiunge alle precedenti evidenze uno studio di stimolazione elettrica su pazienti sottoposti ad *awake brain surgery*, consistente nel confronto delle sensazioni riportate in seguito ad uno stimolo su aree parietali posteriori rispetto ad uno sulla corteccia premotoria. Quando la stimolazione giungeva parietalmente, i pazienti riportavano la forte convinzione di aver svolto un movimento anche in assenza di segnale elettromiografico; uno stimolo premotorio provocava invece un movimento reale ma riportato come totalmente involontario. [29]

Risulta da quanto detto sinora che decodificare il codice neurale espresso nella PPC (nel caso specifico, nell'area V6A) porterebbe all'accesso dei modelli interni associati al movimento conscio specifico per un dato tipo di presa, permettendo effettivamente la rilevazione dell'intenzione di eseguirla.

## 2.3. Deep Learning per Reti Neurali – Storia e metodi

Da un punto di vista prettamente algoritmico, il paradigma Deep Learning descrive tutti gli algoritmi di *Machine Learning* (Apprendimento Automatico) che [30]:

1. Usano una cascata di strati di unità di processing non lineari per l'estrazione e la trasformazione di *features*. Ogni strato utilizza come input l'output di quello precedente.
2. Apprende in maniera *supervisionata* (con dati il cui significato è noto) o *non supervisionata* (l'algoritmo deve apprendere da solo come separare i dati)
3. Apprendono molteplici livelli di rappresentazione che corrispondono a diversi livelli di astrazione; i livelli formano una gerarchia di concetti.

In questo lavoro si useranno le *Deep Neural Networks*, un sottotipo del più ampio insieme delle Reti Neurali, a loro volta contenute nel settore del *Machine Learning*, infine parte del campo *dell'Artificial Intelligence.*

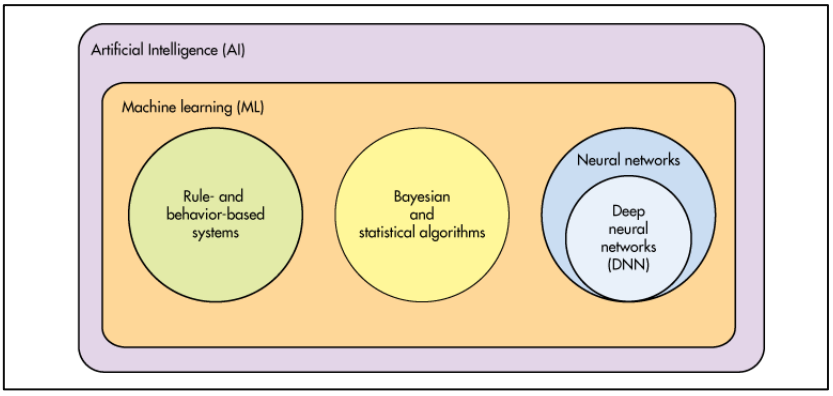


*Figura 11: La posizione del Deep Learning (Deep Neural Networks) all'interno del panorama ML/AI.*

Le reti neurali sono, sommariamente, algoritmi costruiti "a strati", dove ogni strato è composto da un numero variabile di unità simil-neuroni capaci di miscelare gli ingressi dello strato precedente e fornire un'uscita dipendente dall'esperienza dell'unità stessa. Il paradigma *Deep Learning* ha esteso funzionalità ed architettura delle reti moltiplicando esponenzialmente numero e complessità strutturale degli strati, evoluzione resa possibile da un'incrementale potenza di calcolo e quantità di memoria a costi in continuo calo – le *Deep Neural Networks* sono quindi reti neurali molto complesse e, appunto, "profonde".

Per giustificare l'importanza rivestita dalle *Deep Neural Networks* nello sviluppo tecnologico degli ultimi anni e comprendere come i metodi elaborati per costruirle, gestirle, addestrarle possano renderle adatte al contesto del presente lavoro è necessario ripercorrere la storia delle Reti Neurali sin dai loro albori.

### 2.3.1. Breve storia delle reti neurali

E' opinione comune che la storia delle Reti Neurali sia iniziata con il lavoro seminale di Walter Pitts e Warren McCulloch intitolato *"A Logical Calculus of Ideas Immanent in Nervous Activity"*, pubblicato nel 1943 [31]. Tale articolo introduce l'idea di un modello di rete di neuroni senza associarla a sistemi per addestrarlo, quanto piuttosto come un ottimo candidato per comprendere matematicamente i processi sottostanti il pensiero umano.

Il *McCulloch-Pitts Neuron* è stato il primo modello di neurone in assoluto (quello assai più conosciuto di Hodgkin-Huxley è stato proposto solo nel 1952 [32]) ed è conseguentemente molto semplice:

- In ingresso da altri neuroni può riceve solo 0 (nessuna attività), +1 (attività eccitatoria) o -1 (attività inibitoria)
- Se la somma degli input supera un certo valore T di *bias* (detto anche *activation threshold*, soglia di attivazione) allora l'uscita è 1 – altrimenti 0.

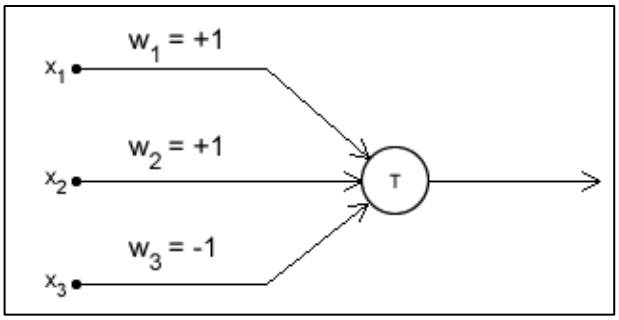

*Figura 12: Il modello McCulloch-Pitts.*

Concatenando più neuroni di questo tipo è possibile ottenere un discreto numero di combinazioni logiche (AND, OR, XOR, …); in effetti, il fulcro del lavoro era l'idea che la mente fosse costituita da un insieme di precise catene logiche del tutto simili a quelle alla base dei calcolatori.

Credo sia importante considerare il contesto storico: le idee di Alan Turing (decrittatore di *Enigma* durante la seconda guerra mondiale e pioniere della moderna *Artificial Intelligence* con [33] e molti altri), Claude Shannon (principale fautore della *Information Theory*, base teorica di tutti i nostri sistemi di comunicazione attuali [34]), John von Neumann (ideatore dell'architettura computazionale condivisa da quasi tutti i moderni computer) e di decine di altri innovatori che nel corso del 1900 hanno costruito le fondamenta scientifiche dell'attuale Era dell'Informazione stavano per trovare il loro vaso di pandora nella costruzione del primo transistor ai Bell Labs, datata 23 dicembre 1947.

In un momento in cui il progresso tecnologico era guidato dall'implementazione di porte logiche in serie e parallelo su circuiti, l'idea di poter impiegare idee similari per studiare il pensiero risultò sensata e certamente pragmatica. Il neurone di McCulloch e Pitts andò quindi a inquadrarsi in un paradigma nuovo, quello della *Cibernetica*,

fondato da Norbert Wiener nel suo *"Cybernetics, or control and communication in the animal and the machine"*[35].

E' in questo contesto che, nel 1957, lo psicologo Frank Rosenblatt presentò al Cornell Aeronautical Laboratory un progetto intitolato *"The Perceptron: A Perceiving and Recognizing Automaton"* [36] [37]. Il centro del progetto era una rete di neuroni di McCulloch-Pitts frapposta tra un "input sensoriale" ed un "output cognitivo", resa capace di apprendere attraverso un modello matematico del condizionamento classico, chiamata *Percettrone* (nella *Figura 13* un percettrone con un solo output).

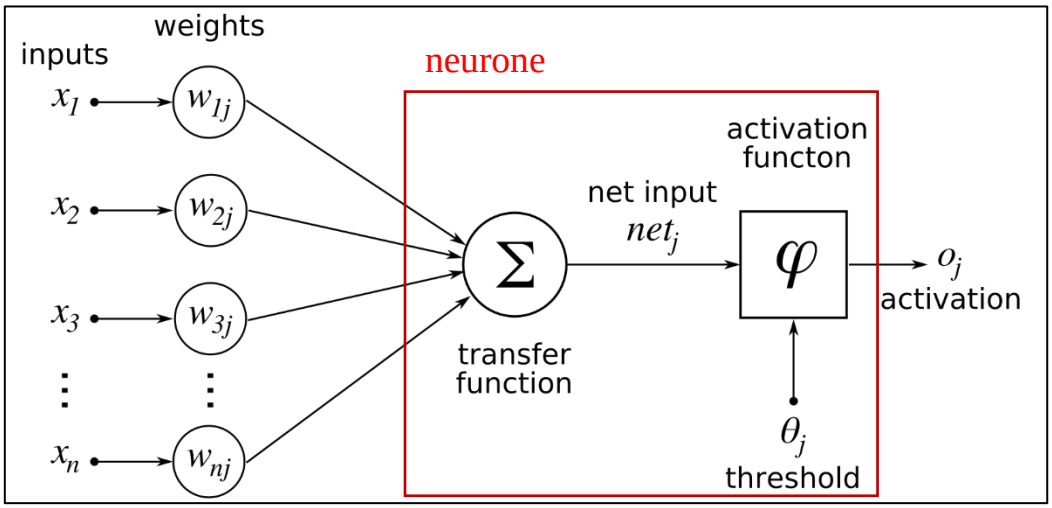


*Figura 13: Un percettrone di Rosenblatt con output singolo. Le "j" indicano che l'output può essere molteplice.*

Nella figura si può vedere come, sostituendo la funzione di attivazione con un gradino centrato sulla soglia, il tutto torni a corrispondere al modello MC-P, con tuttavia l'importante differenza che gli input non sono ristretti a -1, 0 e 1 ma anzi associati ad un input sensoriale di tipo continuo (Rosenblatt nel suo paper usa come esempio un input retinico), peraltro pesato dai *pesi sinaptici* $w_{ij}$. L'innovazione importante è tuttavia la sua capacità di apprendere da un *training set* (un insieme di dati di addestramento), implementata attraverso l'algoritmo di seguito:

**Algoritmo di Convergenza del Percettrone**

1. Inizializzazione dell'algoritmo settando a zero o a valori random i pesi sinaptici e la soglia del neurone di output.

2. Per ogni campione nel *training set* $X_k = [x1_k, x2_k, x3_k, \ldots]$ con uscita prevista $d_k$, detta *label* (ad esempio, $x_1 = [foto\ di\ cane] \rightarrow d_1 = \text{"cane"}$) :

   a. Calcola l'output attuale:

$$u = \sum_{i=1} w_i\, x_i - \theta \quad , \quad o = \phi(u)$$

b. Aggiorna i pesi:

$$w_i = w_i - \mathrm{r}\frac{d\phi}{du}x_i(o-d)\ \forall\ 0 \le i \le n$$

3. Ripeti 2 finché non si minimizza (nel migliore dei casi, si annulla) la quantità:

$$SE\ = \frac{1}{2}(d^2 - o^2)$$

Dove SE è lo *squared error* (errore o scarto quadratico) per ogni campione del *training set*. L'obiettivo è quindi minimizzare lo *scarto quadratico medio* (*mean squared error, MSE)* sull'intero dataset.

In questo modo il percettrone viene *condizionato* a riconoscere i label corretti.

Per percettroni multi-output basta sostituire u con $u_j$, o con $o_j$, $w_i$ con $w_{ij}$ e d con $d_j$: per ogni output j-esimo si fanno le stesse considerazioni fatte prima, notando però che lo scarto quadratico ha una differenza vettoriale.

Nel passo 1, la funzione $\phi$ nel percettrone originale è un gradino, e ha quindi solo 0 (u negativo, quindi attivazione inferiore alla soglia) o 1 (u positivo, soglia superata). Poiché la funzione gradino ha derivata discontinua, l'aggiornamento dei pesi viene eseguito in questo caso seguendo la regola:

$$w_i = w_i - r(o-d)\ \forall\ 0 \le i \le n$$

Nel passo 2b, il termine $(d-o)$ si azzera quando risultato atteso ed uscita coincidono, indicando che tutti i pesi e le soglie hanno raggiunto il valore corretto e non devono più cambiare; quando invece non è azzerato, il suo valore viene moltiplicato per la *learning rate* r che agisce modulando la velocità dell'apprendimento.

Rosenblatt sviluppò il Percettrone presso la CAL sia come software su un calcolatore IBM che sotto forma di macchinario fisico, con tanto di pesi e soglie implementate attraverso potenziometri regolati automaticamente da un sistema meccanico. Merito anche delle forti parole usate da Rosenblatt per descriverlo, il percettrone riscosse enorme successo sia nella società (Il *New York Times* lo descrisse come "l'embrione di un computer elettronico che […] potrà camminare, parlare, […] essere conscio della propria esistenza") sia nell'ambiente cibernetico, nel quale produsse la prima ondata di ricerche sui network neurali.

Tuttavia, nel 1969, Marvin Minsky e Seymour Papert dimostrarono senza ombra di dubbio che il percettrone era in grado di apprendere unicamente pattern *linearmente*

*separabili* (*Figura 14*) e avrebbe inevitabilmente fallito in tutti gli altri casi, persino nell'apprendimento di una funzione logica basilare come lo XOR.

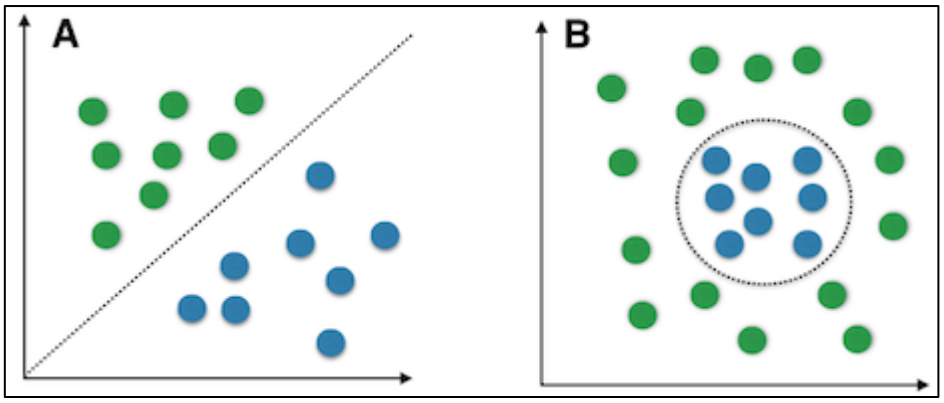


*Figura 14: A sinistra un problema linearmente separabile, a destra un problema non-LS.*

La loro ricerca e quella di altri scettici portarono a due conclusioni:

- Che fosse possibile per il percettrone apprendere lo XOR e qualche altra distribuzione non linearmente separabile attraverso funzioni di attivazione ad-hoc alternative al gradino, inficiando tuttavia l'abilità di apprendimento totalmente automatico e comunque lasciando inapprendibili molte altre distribuzioni;
- Che un *percettrone multistrato* potesse apprendere qualsiasi funzione, XOR compreso (vedi 2.3.3) .

La seconda conclusione fu all'epoca percepita come una lapide sulle ricerche intorno alle reti neurali, poiché gli algoritmi capaci di aggiornare i pesi sinaptici attraverso diversi strati erano estremamente complessi, lenti ed inefficienti.

Queste scoperte portarono ad un generale abbandono delle *neural networks* per quasi vent'anni, finché nel 1986 Rumelhart, Hinton e Williams pubblicarono uno studio in cui presentarono una revisione dell'algoritmo di *back-propagation* [38], metodo di addestramento inizialmente sviluppato nei primi anni '60 da Kelley [39], capace di risolvere il problema dell'apprendimento multistrato. Questa singola pubblicazione è alla base di tutte le architetture neurali esistenti al giorno d'oggi.

### 2.3.2. L'algoritmo di back-propagation

La procedura matematica descritta come "propagazione all'indietro" consiste nel replicare per ogni strato di neuroni quello che succedeva nell'unico *layer* del percettrone: propagare il segnale di successo / di errore dato dal termine $(d - o)$,

corrispondente alla differenza tra *label vero del dato* ed *uscita predetta dal modello*, al fine di cambiare i pesi ed ottenere una migliore performance.

Tutto ciò avviene matematicamente nel contesto del cosiddetto *gradient descent (GD)*. Consideriamo lo scarto quadratico medio (MSE), il cui annullamento è l'obiettivo dell'addestramento:

$$\text{MSE} = \frac{1}{M}\sum_{j=1}^{M}\frac{1}{2}\left(d_j^2 - o_j^2\right)$$

Il suo valore varia a seconda dei pesi e delle soglie (detti in generale *parametri*) della nostra rete, quindi può essere visto come *funzione* di pesi e soglie; il termine generale di funzioni come MSE è "funzioni costo", *Loss Functions*, in quanto determinano la "perdita" rispetto ad una performance ideale. Immaginiamo di avere solo due parametri e visualizziamo la funzione $\text{MSE}(w_1, w_2)$ al loro variare (*Figura 15)*.

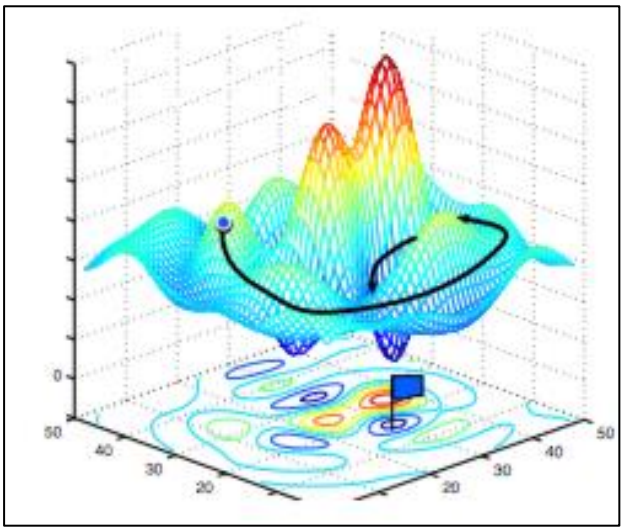


*Figura 15: La superficie di MSE in funzione di w1e w2*

Scegliando a caso w_1 e w_2 ci ritroviamo a partire dal cerchio blu. Dobbiamo arrivare alla bandiera, ma non sappiamo dov'è – l'unica cosa che sappiamo è che in basso, quindi bisogna *scendere.* L'inclinazione di una funzione è definita dal suo *gradiente*, ossia dal vettore ottenuto con la derivazione della funzione lungo tutte le direzioni (=parametri): per ottenere la soluzione del problema dobbiamo quindi cambiare i parametri in modo da *scendere lungo il gradiente*. Quindi, *Gradient descent.* La forma generale della GD per un parametro w è:

$$w \leftarrow w - r\frac{dLoss}{dw}$$

L'algoritmo di convergenza del percettrone è in realtà un'implementazione particolare della procedura GD. La soluzione in questo caso è semplice perché i parametri sono tutti in un unico strato ed influenzano direttamente l'output finale.

In una rete multistrato, al contrario, i pesi e le soglie degli strati precedenti forniscono un contributo ben più difficile da calcolare. L'algoritmo di *back propagation* è stato il primo a fornire un metodo veloce e semplice per farlo.

Consideriamo un neurone j-esimo appartenente allo strato l-esimo. Se, attraverso una variazione di parametri, la sua somma pesata $u_j^l$ viene cambiata di $\Delta u_j^l$, il suo output cambia:

$$o_j^l = \phi\left(u_j^l + \Delta u_j^l\right)$$

E cambia di conseguenza il valore di *Loss* di un termine

$$\Delta \text{Loss} = \frac{dLoss}{du_j^l} \Delta u_j^l$$

Se supponiamo di essere vicini a minimizzare Loss e quindi in una delle valli di minimo, allora $\Delta$Loss non può essere grande: questo perché la derivata nei pressi di un punto critico (quale è un minimo) non può essere alta. La derivata $\frac{dLoss}{du_j^l}$ è quindi una sorta di indice di errore*:* se è alta, siamo ancora distanti dall'obiettivo.

Definiamo quindi l'*errore del neurone j-esimo nello strato l-esimo:*

$$\delta_j^l = \frac{dLoss}{du_j^l}$$

L'algoritmo *BP* agisce calcolando tale termine per ciascun neurone a partire dall'ultimo strato e risalendo fino al primo, per poi derivarne i nuovi valori dei pesi sfruttando le uguaglianze:

$$\frac{dLoss}{dw_{jk}^l} = \delta^l{}_j o_j^{l-1}$$

$$\frac{dLoss}{d\theta_j^l} = \delta^l{}_j$$

Considerando l'ultimo strato nel caso semplificato di un solo neurone di uscita (*Figura 16)* e con *Loss=Squared Error (SE)* otteniamo:

$$\nabla_{o^L} = \frac{dSE}{do^L} = -(d - o^L)$$

$$\delta^L = \frac{dSE}{du^L} = \frac{dSE}{do^L}\frac{d\phi(u^L)}{du^L} = \frac{d\phi}{du^L}(o^L - d)$$

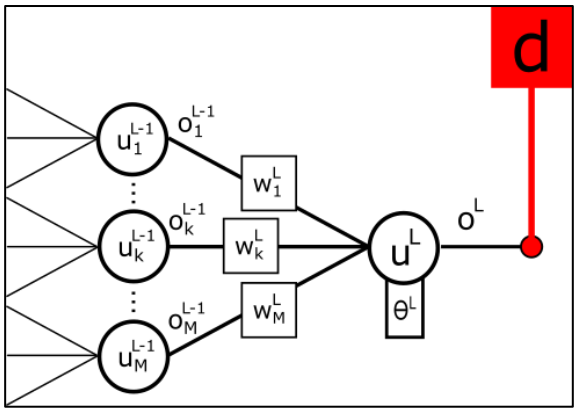


*Figura 16: Esempio di rete con ultimo strato contenente un solo neurone.*

E i pesi e la soglia associati a questo singolo neurone vengono aggiornati con:

$$w_k^L \leftarrow w_k^L - \mathrm{r}(\delta^L o_k^{L-1})$$

$$\theta^L \leftarrow \theta^L - r\,\delta^L$$

Considerando il percettrone e sostituendo opportunamente i termini si riottiene la formula di convergenza già scritta precedentemente, palesando come anche allora si fosse utilizzata una forma di *gradient descent* non dichiarata e dimostrando la validità del procedimento attuale.

La *backpropagation* dell'errore viene poi eseguita con la seguente formula, dove i grassetti indicano i vettori:

$$\boldsymbol{\delta}^l = ((\boldsymbol{w}^{l+1})^T \boldsymbol{\delta}^{l+1}) \boldsymbol{\Phi}'(\boldsymbol{u}^l)$$

Se lo strato l-esimo e quello l+1-esimo possiedono un solo neurone:

$$\delta^l = w^{l+1}\delta^{l+1}\frac{\mathrm{d}\phi}{du^l}$$

Possiamo qui vedere l'essenza della formula: l'errore l-esimo dipende da quello *l+1-esimo* per mezzo di una costante moltiplicativa composta dal peso sinaptico *l+1-esimo* e dalla pendenza della funzione di attivazione del neurone a cui è associato. Questo significa che l'errore *l-esimo* è alto quando:

1. L'errore *l+1-esimo* è alto
2. Il peso sinaptico che lega il neurone dello strato l-esimo a quello dello strato *l+1-esimo* è alto (=connessione "importante")
3. Il neurone dello strato l-esimo è lontano dalla saturazione (le funzioni di attivazione normalmente seguono un andamento sigmoidale o similare per evitare comportamenti "esplosivi").

Di seguito le equazioni di *backpropagation* nel caso generale con costo *Loss*:

1. **[BP1]** Errore all'ultimo strato:

$$\boldsymbol{\delta}^{\mathrm{L}} = \boldsymbol{\Phi}'(\boldsymbol{u}^{L})\, \boldsymbol{\nabla}_{\mathbf{o}^{\mathrm{L}}}(\mathrm{Loss})$$

2. **[BP2]** Propagazione dell'errore allo strato l-esimo:

$$\boldsymbol{\delta}^{l} = ((\boldsymbol{w}^{l+1})^{T}\boldsymbol{\delta}^{l+1})\boldsymbol{\Phi}'(\boldsymbol{u}^{l})$$

3. **[BP3]** Relazione tra Loss e soglia del neurone j-esimo nello strato l-esimo:

$$\frac{dLoss}{d\theta_j^l} = \delta_j^l$$

4. **[BP4]** Relazione tra Loss e peso sinaptico tra neurone j-esimo dello strato l-esimo e neurone k-esimo dello strato l-1-esimo:

$$\frac{dLoss}{dw_{jk}^l} = \delta_j^l o_j^{l-1}$$

Dove $\boldsymbol{\Phi}'(\boldsymbol{u}^{L})$ è una matrice diagonale contenente $\frac{d\phi}{du_j^L}$ e $\boldsymbol{\nabla}_{\mathbf{o}^{\mathrm{L}}}(\mathrm{Loss})$ è il gradiente della funzione costo rispetto agli output $\boldsymbol{o}^{L}_{j} = \phi\left(\boldsymbol{u}_j^L\right)$.

I parametri vengono infine aggiornati seguendo il *gradient descent*:

$$\boldsymbol{w}^{L} \leftarrow \boldsymbol{w}^{L} - \mathrm{r}\,(\,\boldsymbol{\delta}^{L}\,(\boldsymbol{o}^{L-1})^{T}\,)$$

$$\boldsymbol{\theta}^{L} \leftarrow \boldsymbol{\theta}^{L} - r\,\boldsymbol{\delta}^{L}$$

Tutto ciò va ripetuto numerose volte per ogni campione nel *dataset*.

E' importante notare che i parametri subiscono un aggiornamento maggiore, a parità di *learning rate r*, quando l'errore associato al loro neurone è alto. Questo ha senso, considerando che un parametro "giusto" non dovrebbe più subire cambiamenti.

A differenza delle reti neurali, l'algoritmo di *back-propagation* non ha alcun corrispettivo biologico confermato. Il motivo principale per cui il mondo della ricerca è abbastanza sicuro della non-biologicità del meccanismo BP è per la sua relativa lentezza ed inefficienza: un network addestrato attraverso BP+GD deve osservare un campione molte volte per imparare a distinguerlo, mentre l'apprendimento naturale avviene spesso attraverso *one-shot*: quante volte abbiamo osservato un cane prima di capirne le caratteristiche generali? [40] [41]

Malgrado questa limitazione, e diverse altre di natura matematica risolte o compensate nel tempo (alcune saranno trattate brevemente nel capitolo di strumenti e metodi) l'algoritmo BP è alla base del funzionamento di tutti i moderni *framework* di sviluppo di reti neurali utilizzati sia nella ricerca che nel mondo aziendale.

In conseguenza a quanto detto la sua entrata in scena è per molti il marcatore d'inizio delle moderne *Deep Neural Networks.*

### 2.3.3. Deep Neural Networks (DNN)

Una rete neurale è generalmente considerata *Deep* quando il numero di strati è maggiore di 2. Questo perché Cybenko e Hornik hanno dimostrato intorno agli anni '90 che un percettrone con uno strato nascosto oltre a quello di output, quindi 2 strati, è la più "corta" rete (*shallow network*) capace di approssimare qualsiasi funzione dato un numero sufficiente di neuroni e risolvere, tra l'altro, il problema XOR citato in 2.3.1 [42] [43].

Ben prima della pubblicazione dell'algoritmo BP e delle dimostrazioni suddette i ricercatori connessionisti avevano, intorno agli studi di Minsky e Papert del 1969 già citati in 2.3.1, già intuito le potenzialità di una rete multistrato.

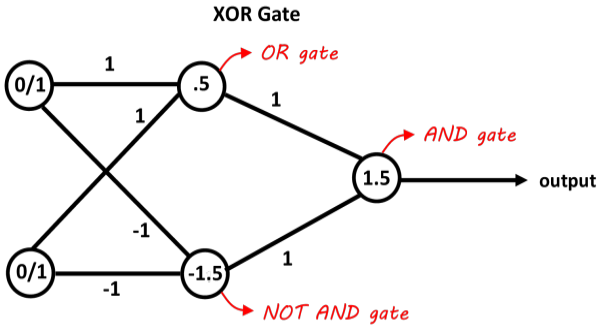


*Figura 17: Percettrone multistrato per la risoluzione dello XOR.*

Lo XOR è infatti risolvibile con un percettrone a due strati (*Figura 17*), una rete tanto semplice da poter essere tracciata manualmente; eppure, gli algoritmi capaci di ottimizzarla non ottennero risultati e/o prestazioni soddisfacenti per molti anni.

Il primo algoritmo capace di ottimizzare efficientemente algoritmi *Deep Learning* (non solo reti neurali) fu quello sviluppato dal matematico russo Alexey G. Ivaknhnenko nel 1968 con il nome di *Group Method of Data Handling (GMDH)* [44]. Questo metodo, tutt'ora impiegato in certe particolari applicazioni, è in grado di selezionare i migliori parametri di ogni strato trattandolo come fosse una funzione polinomiale progressivamente più complessa.

Le reti neurali così addestrate erano chiamate "*polynomial neural networks"*, ancora distanti dalle moderne architetture neurali.

Le prime reti multistrato rassomigliabili alle moderne implementazioni per il riconoscimento di immagini sono quelle sviluppate da K. Fukushima intorno al 1980: il suo modello *Neocognitron* fu costruito imitando alcune caratteristiche del sistema visivo dei vertebrati e, oltre ad essere una delle prime reti capaci di apprendere in modo *unsupervised* attraverso la sola ripetizione degli stimoli, fu anche la prima rete a contenere *"strati convolutivi"* – estremamente importanti per le moderne DNN.

Il *Neocognitron* replica nella sua struttura la specifica abilità del nostro sistema visivo di aggregare caratteristiche progressivamente più complesse nel tragitto tra la corteccia V1 (primo punto di elaborazione cognitiva della vista) fino ad una delle aree di riconoscimento specializzate della corteccia temporale ("via del *what*"). All'epoca il tragitto dell'elaborazione visiva non era così conosciuto, tuttavia la scoperta della mappa retinotopica e della struttura colonnare di V1 ad opera di Hubel e Wiesel nel '59 [45] bastò a fondarne la logica:

1. Ogni colonna neuronale in V1 codifica un preciso *edge lineare* in uno specifico punto della retina
2. Colonne vicine codificano per punti retinici vicini (mappa retinotopica)
3. Ogni colonna invia ad aree di elaborazione successive una singola *feature* corrispondente alla presenza/assenza di un bordo in un determinato punto della retina - riassumendo quindi l'input di un insieme di cellule gangliari retiniche (il *campo recettivo* della colonna) in un "singolo valore".

Esperimenti successivi evidenziarono un comportamento simile nelle cortecce gerarchicamente superiori: l'area V2 ad esempio individua "bordi angolari" a partire da un insieme di output *edge-detector* di V1, allargando il *campo recettivo* complessivo ed aumentando la *complessità* della *feature* individuata.

Il Neocognitron implementa queste scoperte attraverso l'uso di strati di "cellule semplici" e "cellule complesse".

Le cellule semplici identificano una specifica figura all'interno del proprio campo recettivo ed inoltrano tale output a cellule complesse, ciascuna delle quali riassume le attivazioni semplici in ingresso ed indica la presenza di quella figura in un'area estesa / nell'interità dell'immagine.

Ogni "stadio" [cellule semplici, cellula complessa] agisce basandosi su uno specifico *filtro spaziale* (o *kernel*) che rappresenta il prototipo ricercato: ciascuna cellula semplice genera un output corrispondente ad un *indice di somiglianza* tra il kernel ed il proprio campo recettivo attraverso una procedura matematica detta *convoluzione*

*spaziale* (*Figura 18, a sinistra*), dopodiché la cellula complessa estrae una probabilità di presenza attraverso il cosiddetto *max-pooling* (*Figura 18, a destra*).

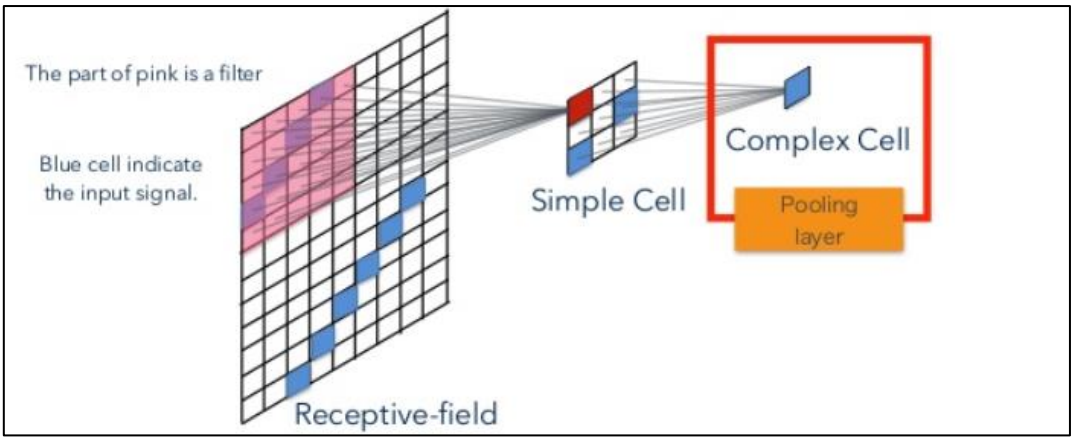


*Figura 18: Convoluzione e pooling nel Neocognitron.*

La convoluzione spaziale per ciascuna cellula semplice agisce seguendo la formula:

$$\text{s\_cell}[m,n] = \sum_x \sum_y Im[x,y]\, filter[m-x, n-y]$$

$$\forall\ \text{x, y} \in (\text{campo ricettivo s\_cell[m, n]}) \quad , \quad \forall\ \text{m, n} \in (\text{gruppo s\_cell})$$

Mentre il pooling della cellula complessa associata segue:

$$\text{c\_cell} = max_{m,n}(\text{s\_cell}[m,n])$$

Ognuno dei numerosi stadi del Neocognitron implementa queste formule per ottenere una codifica gerarchica delle *features* spaziali, anticipando gli algoritmi odierni.

Non ebbe tuttavia molto successo, poiché il suo metodo di apprendimento non-supervisionato – per quanto interessante – gli impediva di imparare task specifici con tempi e performance ragionevoli per il mondo aziendale.

In contemporanea a Fukushima, anche il gruppo di Yann LeCun stava esplorando le possibilità offerte da reti stratificate con stadi di convoluzione-pooling. Nel 1989 LeCun pubblicò il fondamentale *"Backpropagation Applied to Handwritten Zip Code Recognition"*, nel quale addestrò un *Convolutional Neural Network* (*CNN*) a riconoscere le cifre dei codici postali statunitensi in modo supervisionato con l'algoritmo di *Back Propagation.*

Le performance ottenute e l'idea di poter addestrare in maniera specifica un algoritmo dandogli in pasto grandi quantità di dati destarono l'interesse del mercato per le CNN, che risultò nel primo boom di finanziamenti per la ricerca nel campo delle reti neurali dopo i trent'anni di "inverno" che seguirono il fallimento del percettrone.

Nella *Figura 19* è raffigurato un esempio di architettura CNN. Gli strati convolutivi fungono da *estrattori di features*, mentre gli ultimi tre non sono altro che un percettrone multistrato (con qualche modifica, soprattutto alle funzioni di attivazione – lo si vedrà nei metodi) con la funzione di ricombinatore e classificatore di tali *features*.

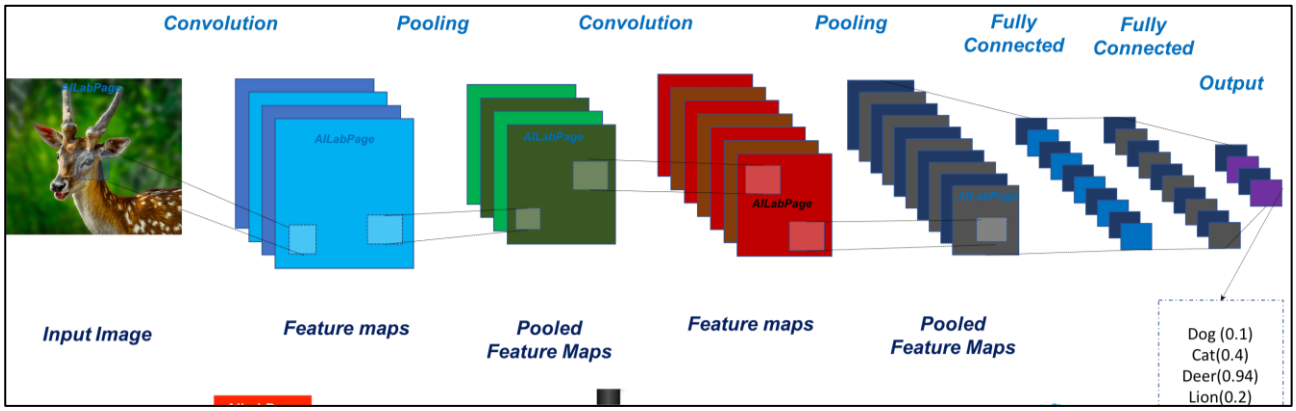


*Figura 19: Un esempio di Convolutional Neural Network.*

La capacità delle CNN di estrarre pattern visuali comuni le rende estremamente efficienti nei compiti di *Computer Vision*: in effetti la ricerca immagini di Google, il sistema di *tagging* delle foto degli amici su Facebook, e molti altri sistemi commerciali che impiegano contenuti visivi sfruttano CNN nel loro funzionamento.

Tra gli anni ’80 e ‘90 si sviluppò grandemente anche un altro tipo di rete neurale: le *Recurrent Neural Networks*, nate dagli studi di Hopfield del 1982 su metodi di memorizzazione a base NN [46] e consolidate nella comunità NN dall’architettura Elman-Jordan per l’approssimazione di dinamiche temporali [47].

Le reti RNN non possiedono solo connessioni tra uno strato e quello successivo, generalmente chiamate *feed-forward*, ma anche connessioni rientranti nello stesso strato, quindi *recurrent*, ed influenti sulla *memoria interna* dei suoi neuroni. Si può immaginare di “svolgere nel tempo” uno *strato ricorrente* (*recurrent layer*) ed analizzarlo come una rete neurale multistrato a sé stante, dove ciascuno “strato interno” modella uno *stato temporale* del sistema dinamico (*Figura 20*)

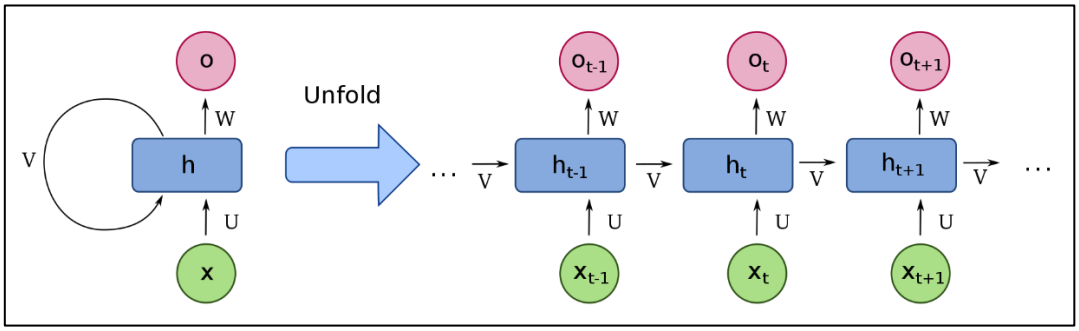


*Figura 20: Uno strato ricorrente "svolto" nel tempo.*

Evidentemente questo rende esponenzialmente profonde e complesse le reti neurali contenenti strati *recurrent*. L'eccessiva profondità portò presto a grossi problemi nell'implementazione pratica dell'algoritmo di *back propagation*: il gradiente del costo impiegato per la propagazione tende infatti a divergere, svanendo o esplodendo, via via che risale gli strati della DNN. Il *vanishing/exploding gradient* bloccò per qualche tempo ulteriori sviluppi nel campo delle RNN.

Jurgen Schmidhuber fu il primo a costruire una *Deep Recurrent Neural Network* di più di 1000 strati nel 1993 1000, risolvendo parzialmente il problema del *vanishing gradient* "pre-addestrando" la rete sopra il segnale da classificare x[t] in modo *unsupervised* per farle incorporare la struttura del segnale, facilitando un *tuning* successivo dato dal BP supervisionato [48].

Sempre Schmidhuber pubblicò nel 1997 insieme a Hochreiter [49] una nuova architettura neurale, la cella *Long Short Term Memory (LSTM, Figura 21),* capace di risolvere il problema del *vanishing gradient* attraverso un sistema di *gating* delle informazioni apprese. Solo dati particolarmente "sorprendenti" e quindi informativi possono entrare nelle celle LSTM, salvando quindi l'algoritmo BP dall'aggiornamento di numerosi "strati interni" e contemporaneamente rendendo la memoria dei "neuroni LSTM" più efficiente.

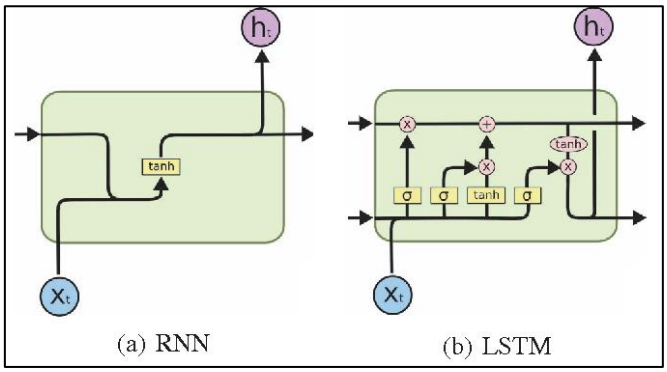


*Figura 21: La differenza tra RNN ed LSTM.*
*I numerosi gate della cella LSTM rende la sua memorizzazione più economica e generalizzante.*

Le reti costruite con LSTM sono perfette per compiti che coinvolgono serie temporali e sistemi dinamici, e sono effettivamente alla base di tutte le tecnologie di riconoscimento vocale e *Natural Language Processing* (ad esempio, Google Translate) presenti sul mercato.

La forma di apprendimento *BP + Gradient Descent* mostrata nel capitolo 2.3.2 viene definitivamente adottata dalla comunità DNN nel 1998 in seguito alla pubblicazione *"Gradient-Based Learning Applied to Document Recognition"* di LeCun [50]; l'unica differenza dell'implementazione reale è che costo ed aggiornamenti dei pesi vengono calcolati sopra la performance media ottenuta su sottoinsiemi (*batches)* dei campioni di addestramento (*Stochastic Gradient Descent, SGD)*.

Con le reti *CNN*, le celle *LSTM* e la coppia *BP+SGD* in campo le DNN entrano così nella forma con cui sono impiegate attualmente. Il primo decennio degli anni 2000 ha visto numerosi avanzamenti sotterranei della tecnologia, soprattutto in termini di potenza di calcolo hardware, profondità delle reti e metodologie per sveltirne l'addestramento e migliorarne le performance predittive.

Nel 2009 Fei-Fei Li, professoressa a capo del *Artificial Intelligence Lab – Stanford University* diede inizio al progetto *ImageNet*, il più grande database al mondo di immagini annotate, cioè dotate di *labels* descrittivi. Citandola:

> *"Our vision was that Big Data would change the way machine learning works. Data drives learning."*

Il motivo di tale affermazione è che le DNN, a differenza di altri algoritmi diffusi precedentemente, non fanno alcuna assunzione a priori sulla struttura dei dati se non "architetturalmente", dipendendo dal tipo e dalla disposizione degli strati – perciò, dato un numero sufficiente di *layers* e *units* in essi, e soprattutto con un quantitativo adeguato di dati, le DNN sono potenzialmente capaci di raggiungere la perfezione in qualsiasi task (*Figura 22)*

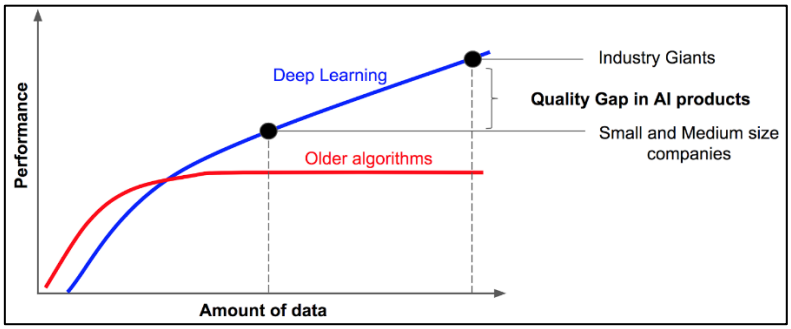


*Figura 22: La differenza fondamentale dell'approccio Deep Learning (qui sottintendendo le DNN) rispetto ad altri metodi ML.*

Grazie alle milioni di immagini facilmente accessibili numerosissimi ricercatori di tutto il mondo hanno quindi potuto attingere ai *Big Data*, normalmente disponibili alle sole grandi aziende, per addestrare i loro algoritmi e sviluppare nuove tecniche. Tra questi, Alex Krizhevsky si distinse tra il 2011 e il 2012 in numerose competizioni internazionali di machine learning con la sua *AlexNet*, una rete costruita sopra un modello proposto anni prima da LeCun e che divenne poi la base di diverse architetture usate anche da Google [51]. Da allora i grandi giganti della tecnologia hanno continuato ad utilizzare il paradigma Deep Learning e ad investire in esso grandi somme di denaro, costruendo inoltre numerosi *Frameworks* di sviluppo *Open-Source* per continuare la linea democraticizzante promossa da ImageNet. Tra questi, il framework Keras, sviluppato da François Chollet in collaborazione con Google, utilizzato in questa tesi per implementare le reti neurali utilizzate [52].

# 3. MATERIALI E METODI

Come già affermato nel capitolo introduttivo, il lavoro è suddiviso in tre diversi esperimenti che mirano a verificare l'applicabilità delle Deep Neural Networks e della metodologia Deep Learning in generale alla decodifica dell'intenzione motoria. I suddetti esperimenti, descritti ed analizzati nel capitolo 4, posseggono nel loro svolgimento degli elementi comuni: tali componenti saranno descritti di seguito con il fine di snellire la successiva analisi sperimentale.

## 3.1. Ambiente di sviluppo

Il progetto è stato sviluppato con il linguaggio Python (python.org) impiegando l'interprete e i moduli forniti nella distribuzione Anaconda 2018.12 – Python 3.7 (anaconda.com).

Tutto il codice è stato scritto nell'ambiente di programmazione interattivo Jupyter Lab 0.35.4 (jupyter.org) e salvato dopo ogni modifica in un *repository Git* privato sulla piattaforma Bitbucket (bitbucket.org).

Le librerie open-source impiegate per la manipolazione numerica del dataset sono NumPy 1.15.4 (numpy.org) e Pandas 0.23.4 (pandas.pydata.org); quelle utilizzate per la visualizzazione di segnali e risultati sono matplotlib 3.0.2 (matplotlib.org) e seaborn 0.9.0 (seaborn.pydata.org); infine, le *Deep Neural Networks* sono state sviluppate con la libreria Keras 2.2.4 (keras.io) con *backend* TensorFlow 1.12.0 (tensorflow.org) e le loro performance calcolate e validate con la libreria scikit-learn 0.20.1 (scikit-learn.org).

Il computer usato per addestrare e testare le reti possiede le seguenti caratteristiche:

- SO: Microsoft Windows 10 Home build 17763
- Processore: Intel Core i7-7700HQ
- Scheda Grafica: NVIDIA GeForce GTX 1060 Max-Q Design
- RAM: 16 GB DDR4

Keras e Tensorflow sono interfacciati alla GPU attraverso le librerie di primitive NVIDIA CuDNN 7.1.4 (https://developer.nvidia.com/cudnn), le quali consentono l'utilizzo della GTX 1060 nei calcoli associati all'addestramento delle DNN parallelizzando tutte le operazioni matriciali sopra i 1280 processori grafici della scheda.

## 3.2. Analisi del dataset

Il dataset impiegato consiste in un insieme di timestamps di spiking neuronali *spike-sorted* manualmente durante la registrazione da *Single Unit Recordings* dell'area V6A di due esemplari maschi di *macaca fascicularis* durante compiti di *reach to grasp* con 5 diversi oggetti.

Descrivo di seguito l'esperimento che ha originato il dataset, prendendola da [53].

Gli oggetti, posti su un pannello automatico, sono stati presentati uno alla volta in maniera random, nascondendo gli altri alla vista del macaco. L'animale poteva, con il braccio controlaterale al sito di registrazione, avviare e terminare il *trial* corrente premendo un bottone posto vicino al torso e fuori dallo spazio visivo (*home button*) e compiere il movimento di *reach to grasp* sull'oggetto proposto.

Il task iniziava al buio con la pressione dell'*home button* per 1s, intervallo definito *epoch FREE* in quanto l'animale era libero di muovere gli occhi a volontà. Successivamente, un LED verde di fissazione si accendeva per 0.5-1s; l'interruzione della fissazione rilevata dal sistema di controllo oculare o il rilascio del bottone portava al riavvio del trial. L'oggetto veniva quindi illuminato per 0.5s durante l'*epoch OBJ-VIS;* a questo punto l'illuminazione cessava nella variante sperimentale *DARK* o persisteva nella variante *LIGHT* per consentire una successiva separazione degli effetti puramente visivi da quelli visuomotori e motori. Dopo 0.5-1.5s di attesa, definita *epoch DELAY*, si accendeva un LED rosso che indicava all'animale di lasciare il bottone ed effettuare il movimento di prensione (*epoch REACH-TO-GRASP, RTG)*. Tale presa doveva essere mantenuta fino allo spegnimento del LED rosso, segnale di ritorno sull'home button. L'intero esperimento è descritto nello schema della *Figura 23* sottostante.

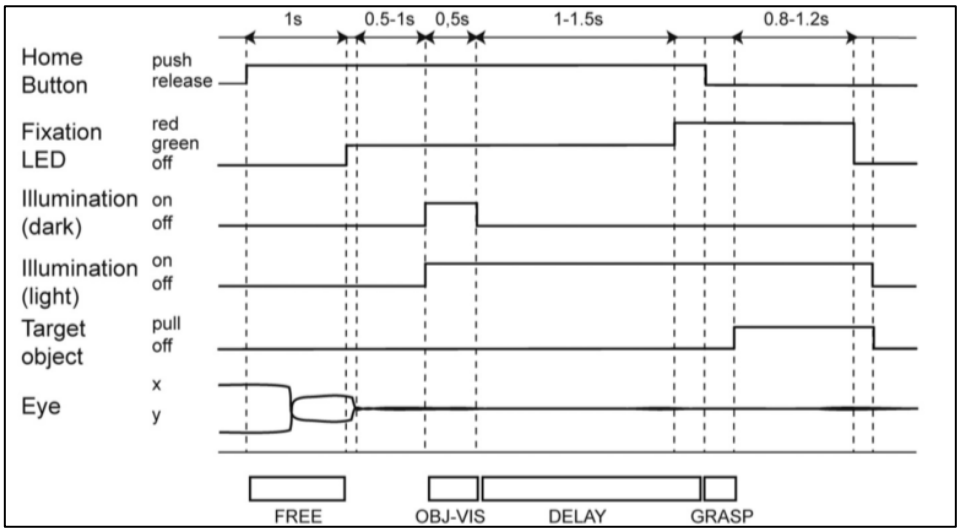


*Figura 23: Schema dell'esperimento dal quale proviene dataset impiegato nel lavoro.*

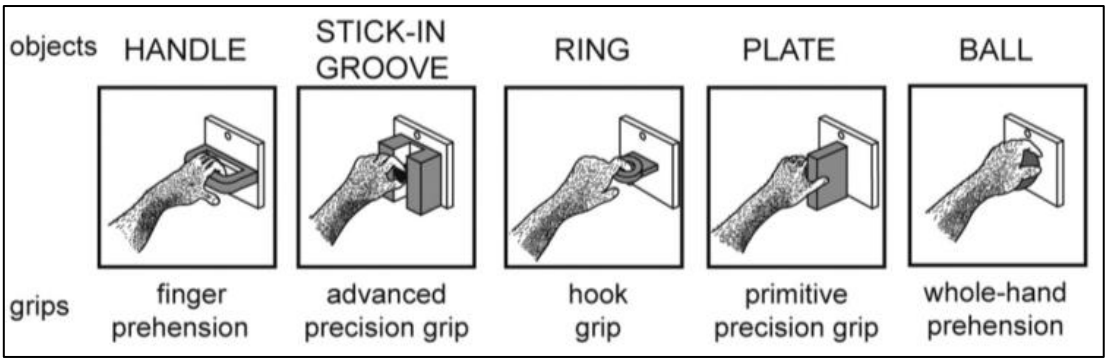


*Figura 24: Oggetti impiegati nell'esperimento e prese associate.*

I 5 oggetti proposti, visibili in *Figura 24*, sono stati scelti in modo da indurre 5 prese diverse.

- PALLA, prensione con l'intera mano
- MANIGLIA, prensione con tutte le dita eccetto il pollice
- ANELLO, prensione a uncino con indice
- PIASTRA, prensione semplice usando il pollice contrapposto alle falangi distali delle altre dita
- CILINDRO NELLA FESSURA, prensione avanzata con pollice e indice.

Ogni oggetto è stato riproposto per 10 volte in maniera casuale, per un totale di 50 *trials*. Questo è stato ripetuto numerose volte, ogni volta per ciascun neurone trovato durante l'esplorazione dell'area oggetto di studio.

Nell'ipotesi che i modelli visuomotori siano già stati imparati dalle scimmie e che quindi sia possibile trascurare il cambiamento di codifica nel tempo, è possibile aggregare i neuroni risultanti dallo *spike-sorting* come se fossero stati acquisiti contemporaneamente nei *trials*. In questo modo ai 50 *trials* della scimmia 1 sono associabili 47 serie di *spike-timestamps* ciascuna relativa ad un neurone, mentre alla scimmia 2 sono associati soli 32 neuroni.

Per gli scopi di questo lavoro è stata usata la sola variante DARK dell'esperimento per entrambe le scimmie.

Come visibile dagli schemi, la durata dei trial è randomicamente variabile da un minimo teorico di 3.8s ad un massimo teorico di 5.2s. Per uniformarne la lunghezza sono stati estratti per ciascuna scimmia due dataset separati, uno allineato sopra *KEY-DOWN* [0:+2500] (dalla pressione *home button* all'inizio di *DELAY*) e l'altro sopra *KEY-UP* [-1000:+1000] (intorno al rilascio di *hb* fino al completamento del task). Nell'ipotesi che il centro di *DELAY* avesse poca rilevanza per il lavoro i due dataset sono stati uniti in un unico insieme con *trials* di estensione fissata a 4.5 secondi.

## 3.3. Pipeline di preprocessing

Il dataset è quindi composto di due macroblocchi (i macachi) contenenti 5 blocchi (uno per oggetto) di 10 trials contenenti 32 oppure 47 liste di timestamps, ciascuna di lunghezza variabile a seconda del numero di eventi rilevati nei 4.5 secondi considerati. Questo formato presenta all'ingresso dell'algoritmo un insieme di dati sparso e senza una struttura temporale definita, rendendo impossibile qualsiasi analisi. E' stato quindi necessario passare il dataset attraverso una *pipeline* di pre-elaborazione al fine di renderlo intellegibile per le reti neurali testate.

La pipeline è divisa in due diverse componenti: una anteriore al processo di validazione, con il compito di formattare i dati in maniera intellegibile all'algoritmo, ed una ripetuta ad ogni ciclo validativo, deputata all'estrazione di campioni casuali dai dati precedentemente formattati al fine di assicurare una valutazione di performance realistica.

### 3.3.1. Pre-validation preprocessing

#### BINNING

Per ogni scimmia, per ogni oggetto/classe, per ogni trial le liste sono state sottoposte ad un processo di *binning* con *bins* di 10 ms, ossia è stata generata una matrice $\boldsymbol{A}$ di 47 o 32 colonne e $int\left(4.5s * \left(\frac{1000ms}{10ms}\right)\right) = int(4.5 * 100) = 450$ righe con ogni elemento $A_{ij}$ contenente il numero di *spikes* con *timestamp* compresa tra $10ms * i$ e $10ms * (i + 1)$ nel canale (=neurone) j-esimo. Questa trasformazione definisce in uscita una *time-series* multi-canale discreta $\left(Fs = \frac{1}{10ms} = 100Hz\right)$ a valori discreti (Ogni *spike* neurale corrisponde ad 1, quindi ogni bin contiene un numero intero $\geq 0$).

In uscita da questo stadio di *preprocessing* vi è quindi, per ogni scimmia, una matrice di dimensioni [5, 10, 450, 47/32].

#### LABELING

Date le considerazioni fatte nei capitoli 1. e 2. sulle tempistiche dello stato di intenzione di movimento si è deciso di associare i 500 ms precedenti il *KEY-UP* all'evento intentivo. Per utilizzare tale decisione nell'addestramento dell'algoritmo è necessario costruire un *label temporale* che segnali la differenza tra quei 500 ms ed il restante tempo del trial. Per ogni scimmia, per ogni classe (=oggetto), per ogni trial è stata

quindi costruita una sequenza lunga tanto il numero di *bins* che identificasse la presenza (1) o l'assenza (0) dell'intenzione di RTG per un preciso oggetto durante quei *bins*. Poiché ogni *bin* comprende 10 ms, il label 1 giace sui 50 *bins* prima del *KEY-UP*, mentre il label 0 occupa il resto (*Figura 25*).

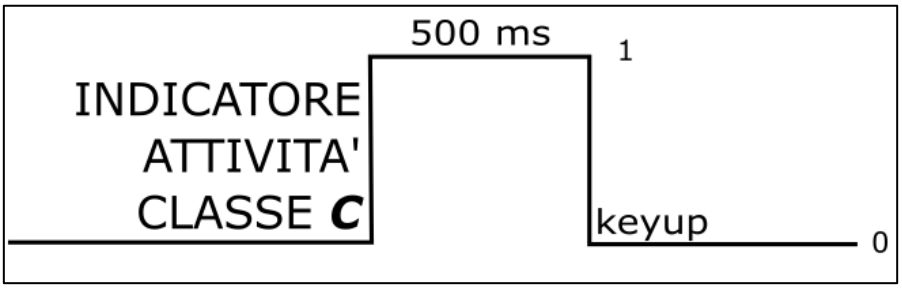


*Figura 25: Un* label temporale *che segnala la presenza/assenza dell'intenzione di movimento per la classe/oggetto C.*

Il processo di labeling genera quindi, per ogni scimmia, una matrice di dimensioni [5, 10, 450, 5], dove i canali dedicati ai labels temporali sono 5 in tutti i trial in quanto è necessario segnalare la presenza o l'assenza di tutti gli oggetti in tutti i segnali registrati.

### 3.3.2. Mid-validation preprocessing

#### SAMPLING

Per ottenere una corretta validazione delle performance vedremo in 3.4. che è necessario dividere il dataset in tre insiemi separati: *training*, *validation*, *test*. L'algoritmo sarà addestrato sopra il *training set*, il suo grado di apprendimento durante l'addestramento sarà verificato sul *validation set* e, alla fine dell'apprendimento, le performance effettive del modello saranno calcolate sopra il *test set*. Il primo importante passo di questo procedimento è la separazione di (training, validation)=*TRAINVAL* e *TEST*, che deve avvenire in modo da:

1. Mantenere la struttura dei trial, in modo da non mischiare frammenti temporali di trial diversi;
2. Separare lo stesso numero di trial per ogni classe;
3. Essere totalmente casuale.

Il componente di sampling della *pipeline* provvede a soddisfare queste tre richieste implementando le funzioni di scelta casuale della libreria NumPy. Dato un numero K di trial da assegnare all'insieme *TRAINVAL*, l'uscita di questo componente è consiste nelle matrici:

- data_trainval : [5, K, 450, 47/32] labels_trainval: [5, K, 450, 5]
- data_test: [5, 10-K, 450, 47/32] labels_test: [5, 10-K, 450, 5]

### CONCATENAZIONE

L'algoritmo accetta in *input* solo un segnale nella forma [id_campione, [bins, canali]] ed un label nella forma [id_campione, [classi]]. Le matrici attuali non sono quindi un ingresso accettabile. Le dimensioni "classe" e "trial" delle matrici *data* e *labels* di TRAINVAL e TEST sono quindi concatenate in modo da ottenere in uscita le dimensioni:

- data_trainval : [5*K, 450, 47/32]  labels_trainval: [5*K, 450, 5]
- data_test: [5*(10-K), 450, 47/32]  labels_test: [5*(10-K), 450, 5]

Il primo asse, concatenazione degli assi "classe" e "trials", prende il nome di "acquisizione".

### SHUFFLE DELL'ASSE DI ACQUISIZIONE

Per evitare qualsiasi possibile effetto di interdipendenza tra sequenze di trials dello stesso tipo, evento che indurrebbe ad una performance stimata più elevata del normale, l'asse "acquisizione" creato al passo precedente viene mescolato randomicamente ad ogni validazione.

### SLIDING WINDOWING

In un ambiente di utilizzo reale l'algoritmo avrebbe, in qualsiasi momento della sua esecuzione, accesso solo all'ultimo campione per ogni canale insieme eventualmente ad un *buffer* contenente quelli acquisiti negli attimi precedenti. Questo componente della *pipeline* ha il compito di impostare l'addestramento in modo da ottenere un modello capace di affrontare tale scenario.

Nelle ipotesi di:

- Acquisizione multi-array (i neuroni vengono considerati come acquisiti contemporaneamente)
- Informazione irrilevante nelle fasi *FREE* e post-*RTG*
- Buffer di 50 *bins* precedenti con scorrimento di 1 *bin.*
- Processing e classificazione richiedono, considerando un'implementazione ottimizzata tipica di uno scenario di utilizzo *quasi-real-time* effettivo, un tempo inferiore a 10 ms (il tempo di produzione minimo di un *bin* e quindi l'inizio di un nuovo processo di classificazione a partire dal buffer)

Questo componente effettua, per ogni scimmia:

1. Una concatenazione di tutte le acquisizioni

2. Un processo di *sliding-window* sopra l'intero segnale concatenato, costruendo "campioni" composti di 50 bins e 47/32 neuroni a distanza di 1 bin uno dall'altro e mantenendo l'associazione bin-label temporale fissandola sull'ultimo bin (*Figura 26*) – questo perché l'algoritmo venga addestrato a classificare l'attuale condizione temporale a partire dal buffer di *bins* precedenti.

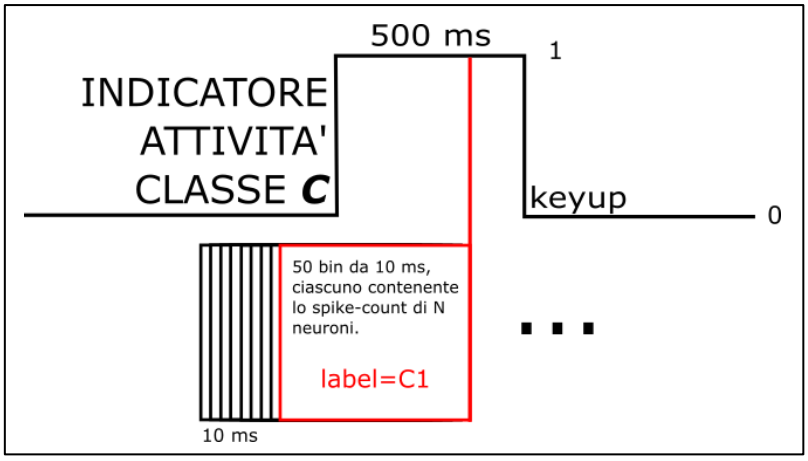


*Figura 26: Immagine riassuntiva dell'uscita post-sliding-windowing.*
*Il "label=C1" indica l'assegnamento al buffer del label di "presenza attività della classe C".*

Con questo processo viene generato il primo *buffer*, la prima finestra solo a partire dal 50-esimo *bin* delle acquisizioni concatenate.

Le dimensioni delle matrici in uscita da questo componente sono:

- data_trainval : [(5*K*450-50), 50, 47/32] labels_trainval: [(5*K*450-50), 5]
- data_test: [(5*(10-K)*450-50), 50, 47/32] labels_test: [(5*(10-K)*450-50), 5]

### PADDING A 50 CANALI.

Per evitare possibili differenze di calcolo causate dalle differenze nel numero di neuroni nei dataset delle due scimmie sono state aggiunti canali con *bins* totalmente azzerati fino a portare entrambi i dataset a 50 canali. Il canale azzerato simula un "neurone perso", una cellula di cui non si rileva alcuna attività di *spiking*. Questo concetto verrà impiegato nell'esperimento 3 per l'analisi *neuron loss*.

Le matrici in uscita da questo stadio finale hanno dimensioni:

- *N_trainval = (5*K*450-50)*
- *N_test = (5*(10-K)*450-50)*
- data_trainval : [*N_trainval*, 50, 50] labels_trainval: [*N_trainval*, 5]
- data_test: [*N_test*, 50, 50] labels_test: [*N_test*, 5]

## 3.4. Metriche impiegate

### 3.4.1. Cross-Entropy Loss (o Log Loss)

L'errore quadratico medio (*MSE*) non è l'unica funzione *Loss* esistente, né la migliore: in effetti non è quasi mai utilizzata nei problemi di classificazione, perché tendente a "non sbilanciarsi", ad essere "media" appunto – mentre un classificatore produce idealmente risultati estremi, 0 oppure 1 per ciascuna classe. Generalmente la MSE viene quindi impiegata per problemi di regressione, dove l'uscita del modello va ad approssimare un'uscita continua; per i problemi di classificazione come quello di questo lavoro viene invece sovente impiegata la *Cross-Entropy* (*CE*) *Loss*, detta anche *Log Loss*. Il motivo di ciò è del suo comportamento estremizzante, mostrato in *Figura 27*: tanto più una classificazione è distante dal valore atteso quanto più la funzione *Loss* cresce esponenzialmente.

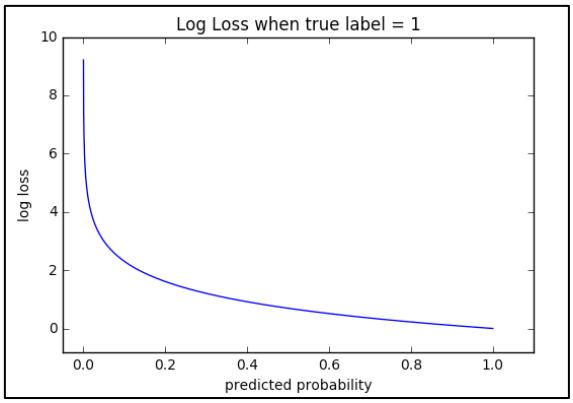


*Figura 27: Grafico della* CE *Loss al variare della probabilità predetta rispetto al* true label.

Si nota subito che la *CE* lavora in termini prettamente probabilistici, come sottolineano dal nome dell'asse delle ascisse. Questo dipende dal fatto che l'Entropia è una misura definita nell'ambito della già citata *Information Theory* di Shannon con lo scopo di quantificare l'informazione associata ad una certa sorgente di dati $X$ in termini della *probabilità* che tale sorgente emetta una data "parola" $x$.

$$\text{Entropia} = \mathrm{H}(X) = -\sum p(X) \log p\,(X)$$

La cross-entropia è una misura di entropia "cross", quindi "tra", due distribuzioni binarie ("parole"=0, 1): la distribuzione predetta $P$ e quella vera $T$. La CE misura quindi la somiglianza tra P e T quantificando l'informazione di T (*true distribution)* contenuta in P (*artificial estimated distribution)*.

P si identifica con la distribuzione in uscita dal modello, di fatto interpretando le uscite con range (0, 1) come "probabilità di appartenenza alla classe C".

Si mostra nella prossima equazione il caso di distribuzioni a singolo valore (problema single-class), date le due label $p_x$ e $t_x$ associate al singolo campione classificato $x$.

$$CE_X = -\big(t_x \log(p_x) + (1 - t_x) \log(1 - p_x)\big)$$

Quando $t_x$ e $p_x$ differiscono (0 contro 1) $CE_x$ assume valore infinito, mentre diventa 0 quando corrispondono: quest'ultimo caso identifica una corrispondenza delle due distribuzioni, almeno rispetto al singolo campione. La cross-entropia totale delle distribuzioni $T$ e $P$ viene ovviamente stimata con una media su tutti i campioni. In letteratura la CE per distribuzioni binarie single-class viene spesso definita *Binary Cross-Entropy (BCE)*

Nel caso multi-class, la CE viene chiamata *Categorical Cross-Entropy (CCE)* e così definita:

$$\mathrm{CCE_x} = -\sum_{c=1}^{N_c lassi} \mathbf{1}_{x,c} \log\big(p_{x,c}\big)$$

Dove $\mathbf{1}_{x,c}$ è una "funzione indicatrice binaria" che vale 1 solo quando $c$ è la classe di appartenenza del campione $x$ e 0 altrove e $t_{x,c}$ e $p_{x,c}$ sono rispettivamente il *label* reale e la probabilità predetta di appartenenza alla classe C associati al singolo campione $x$.

E' bene notare che, per un dato $x$, le $p_{x,c}$ del modello diverse da quella corrispondente al *label* reale sono ignorate dalla *CCE*: questa la rende una funzione valida solo per situazioni in cui ogni campione appartiene ad una sola classe. Per situazioni *Multi-label* in cui x può essere associato a più "classi", quindi a più *labels*, è necessario impiegare altre funzioni di *Loss*.

### 3.4.2. F-Score (o F1 measure)

La natura del compito che si vuole svolgere, cioè distinguere una singola sezione di 500 ms (label 1) da tutto il resto della registrazione (label 0) rende il *dataset* estremamente sbilanciato: la proporzione 1/(1+0) è <15% nel caso *single-class* di riconoscimento di una generica attività e <3% nel caso *multi-class* di un movimento rispetto a tutti gli altri. Il fortissimo sbilanciamento rende totalmente inutile l'utilizzo della metrica di accuratezza, che nel secondo caso restituirebbe una performance media del 97% nel caso di un classificatore con output costantemente nullo.

La soluzione più impiegata in questo caso è l'utilizzo delle metriche denominate *precision* e *recall***.**

$$Precision = \frac{tp}{tp+fp} \quad , \quad Recall = \frac{tp}{tp+fn}$$

Dove le abbreviazioni significano:

- tp: *true positive*, il label vero e quello classificato sono entrambi =1;
- fp: *false positive*, il classificatore ha erroneamente classificato =1 un label=0;
- fn: *false negative*, identifica l'errore opposto.

La *precision* rappresenta la capacità del modello di separare correttamente i casi positivi da quelli negativi, mentre il *recall* la sua bravura nel richiamare tutti i casi positivi. Date 10 biglie di cui 3 rosse e 7 bianche, un modello incaricato di estrarre quelle rosse ottiene 1. P=1 se non confonde neanche una biglia 2. R=1 se identifica 3 o più biglie rosse.

Esiste una metrica costruita in modo da considerare *precision* e *recall* insieme, consistente nella loro media armonica, chiamata *F1-score* o *F-measure***.**

$$\mathrm{F} = 2\frac{Precision * Recall}{Precision + Recall}$$

Nel problema di prima, un modello con F=1 è capace di selezionare correttamente le 3 biglie rosse da tutte le altre, senza confondersi.

La metrica F è indipendente dallo sbilanciamento del dataset e a differenza di altre metriche come la Area Under Curve (AUC, descritta successivamente) non richiede iterazioni sopra la scelta di una soglia (fissata normalmente a metà dell'escursione del classificatore, 0.5 in questo caso), quindi rappresenta la performance "naturale" dell'algoritmo, senza ulteriori manipolazioni.

Il caso multiclasse si affronta calcolando la F per ogni singola classe (presenza di quella specifica attività contro parimenti l'inattività e la presenza di altre) e mediando tutte le F risultanti. La metrica così ottenuta, denominata *Average F-score*, descrive l'abilità del modello di identificare totalmente e correttamente le attività.

### 3.4.3. Receiver Operating Characteristic ed Area Under Curve

Sebbene la presenza / assenza di attività sia discretamente separata in termini di 0 ed 1, l'output del modello non è di per sé discreto: come detto precedentemente, esso è descrivibile come "probabilità di appartenenza alla classe C" con range continuo (0, 1). Per ottenere un risultato binario è quindi necessario sogliare l'uscita con un valore $thr$, sotto al quale essa equivale a 0 e sopra ad 1. La scelta più naturale è utilizzare una $thr = 0.5$, dividendo a metà l'escursione totale dell'output e separando massimamente

i due estremi; tale soglia è impiegata in generale per la determinazione di metriche quali l'accuratezza, la *precision*, il *recall*, … ma parte dall'assunzione che le distribuzioni degli output associabili al label 0 (=*negatives*) o al label 1 (=*positives)* abbiano uguale estensione nel range di uscita. Questo è idealmente vero, ma realisticamente una soglia diversa da 0.5 può portare a risultati migliori. Questo crea due problemi:

1. come identificare la migliore soglia?
2. come misurare la performance di un algoritmo senza dipendere da una soglia?

La risposta ad entrambe è identificabile nella curva *Receiver Operating Characteristic* (ROC) che grafica i punti $(x, y) = (FalsePositiveRate, TruePositiveRate)$ in funzione della soglia (*Figura 28*).

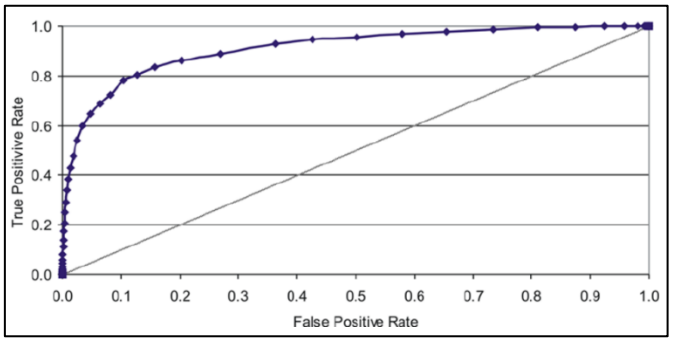


*Figura 28: esempio di curva ROC. Ogni punto corrisponde ad un diverso valore di soglia applicato a tutte le uscite del modello, generalmente con thr progressivamente più vicina ad 1 con il tendere al punto (0,0).*

Il valore $FPR = \frac{fp}{fp+tn}$ è descritto anche come *1-specificity*, mentre $TPR = \frac{tp}{tp+fn}$ è definito *sensitivity*: questo perché spostandosi verso la sinistra del grafico e quindi impiegando una $thr$ crescente il modello sta diventando sempre più specifico (produce meno falsi positivi) e meno sensibile (produce meno positivi in generale), viceversa dirigendosi a destra. La bisettrice indica la cosiddetta "curva ROC random", cioè la curva che produrrebbe un algoritmo che per tutte le soglie fornisce uscite totalmente a caso (TPR=FPR=0.5).

La curva ROC permette di valutare due cose:

1. La soglia migliore per determinare un certo grado di sensibilità / specificità dell'algoritmo;
2. La bontà dell'algoritmo a prescindere dalla soglia, a seconda della sua distanza dalla bisettrice.

Fornendo così le due risposte cercate. Il secondo punto in particolare è riassumibile in un singolo numero, considerando che l'area sotto la bisettrice ha valore 0.5 e che quindi un algoritmo ben costruito dovrebbe averne una maggiore: l'*Area Under Curve (AUC),*

che matematicamente descrive “la probabilità cumulativa di assegnare label positivo ad un campione positivo”.

Purtroppo, la natura stessa di ROC ed AUC ne limitano l’applicabilità al solo caso *single-class.* Questo perché la distribuzione del caso negativo (0) differisce a seconda della classe che si sta considerando: la classe C considera nel proprio caso negativo anche i casi positivi delle altre classi!

Questa metrica verrà quindi impiegata solo nell’Esperimento 1 per evidenziare ulteriormente le performance ottenute dagli algoritmi.

## 3.5. Metodo di addestramento

### 3.5.1. Algoritmo

Per l’addestramento delle reti neurali è stata impiegata una versione fortemente modificata della già citata *Stochastic Gradient Descent (sgd)*, chiamata *Adam (Adaptive moment estimation)* [54]. Si ricorda che “stocastico” sottintende il fatto che il valore della funzione costo (*loss function*) e gli aggiornamenti dei parametri dipendenti da questa sono calcolati non sopra ciascun campione, ma mediando sopra un sottoinsieme di campioni (chiamato *batch*) estratto iterativamente dal *dataset*. L’effetto del considerare la media è di compensare la rumorosità dei singoli campioni.

Prima di scrivere le equazioni del metodo Adam, di seguito si rappresenta la versione stocastica del metodo *gradient descent* classico per un *batch* di M campioni:

$$SGD\colon \boldsymbol{w}^{l} \leftarrow \boldsymbol{w}^{l} - \frac{\mathrm{r}}{M}\sum_{x} \boldsymbol{Grad}^{x,l}$$

Dove nel caso si stesse impiegando l’algoritmo di *back propagation,*

$$\boldsymbol{Grad}^{x,l} = \boldsymbol{\delta}^{x,l}(\boldsymbol{o}^{x,l-1})^{T}.$$

Come detto in precedenza, il metodo GD non fa altro che seguire le discese della superficie della *loss function*; non c’è alcun controllo matematico sul modo in cui scende, quindi la sua velocità e solamente funzione del gradiente della funzione costo. Quando la rumorosità dell’informazione contenuta nei campioni è tale da rendere anche il gradiente medio sul *batch* poco costante, il metodo SGD può produrre degli “scatti” nella traiettoria parametrica tanto incoerenti da inficiare il processo di apprendimento. Per questo motivo sono nati dei metodi “adattivi” che tentano:

1. Di modulare l'accelerazione e la velocità dello stato parametrico nella sua traiettoria unendo al gradiente attuale quelli precedenti;
2. Di adattare i parametri di apprendimento (es. *learning rate*) nel corso del tempo in modo da coadiuvare quanto detto al punto 1.

Tra questi, il più diffuso è appunto il metodo *Adam*. Di seguito si scrivono le sue equazioni costituenti per un singolo parametro w:

$$m^x{}_t = \beta_1 m^x_{t-1} + (1-\beta_1)\frac{1}{M}\sum_x \frac{dLoss}{dw}^x{}_t$$

$$v^x{}_t = \beta_2 v^x{}_{t-1} + (1-\beta_2)\frac{1}{M}\sum_x \left(\frac{dLoss}{dw}^x{}_t\right)^2$$

Queste rappresentano stime iterative (nel tempo, indicato dal contatore intero t) rispettivamente della media e della varianza del gradiente della funzione costo rispetto ad un preciso campione x di M totali del *batch*.
$\beta_1$ e $\beta_2$ sono parametri di apprendimento (detti *weight decay*) con valore tra 0 ed 1 esclusi: infatti con 0 le quantità dipenderebbero solo dalle stime presenti, viceversa con 1 solo dalle stime passate – in questo caso bloccando addirittura l'apprendimento.
La stima iterativa parte con entrambe le quantità poste a 0; con valori delle due beta prossimi ad 1 l'iterazione tenderebbe a rimanere nei pressi del valore iniziale nullo, inficiando l'apprendimento. Per evitarlo si "correggono" tali quantità nel modo seguente:

$$\widehat{m_t^x} = \frac{m_t}{1-\beta_1^t}\ ,\ \widehat{v_t^x} = \frac{v^x{}_t}{1-\beta_2^t}$$

Bisogna notare l'elevazione a potenza t delle beta a denominatore. All'inizio (con t=1) un valore alto delle beta fa tendere a zero il denominatore, andando a compensare il valore nullo iniziale di media e varianza. Valori bassi delle beta non producono nessun comportamento particolare.
L'update di un generico parametro *w* avviene come segue:

$$\mathrm{w}_{t+1} \leftarrow \mathrm{w}_t - \frac{\mathrm{r}}{\sqrt{\widehat{v^x}_t} + \epsilon}\widehat{m^x}_t$$

La deviazione standard (radice della varianza) a denominatore ha lo scopo di compensare la rumorosità del gradiente medio stimato: quando questo è particolarmente rumoroso la sua varianza stimata risulta altresì alta, quindi il valore a denominatore diviene alto ed aumenta il suo potere compensativo, limitando l'influenza di questo gradiente rumoroso. Il termine $\epsilon$ ha funzione di impedire errori numerici quando la varianza è bassa.

Quanto scritto sinora giustifica l'impiego del metodo Adam, in effetti il più usato nella letteratura recente grazie alla velocità ed alla robustezza del suo metodo di apprendimento.

### 3.5.2. Parametri di addestramento

I parametri $r$, $\beta_1$, $\beta_2$, $\epsilon$ dell'algoritmo Adam sono settati ai valori di default consigliati dal paper.

Tutti gli addestramenti sono svolti scorrendo sopra il training set per un numero di ripetizioni fissato a 50, scelto manualmente osservando i grafici del valore della funzione *Loss* su alcune prove delle architetture (*Figura 29*) e settandolo al raggiungimento dell'asintoto minimo della curva del *training set.* Il valore fissato consente una maggior ripetibilità della validazione rispetto a metodi automatici di interruzione dell'addestramento (*early stopping*), in quanto qualsiasi forma di automazione introduce rumore sperimentale sulle misurazioni.

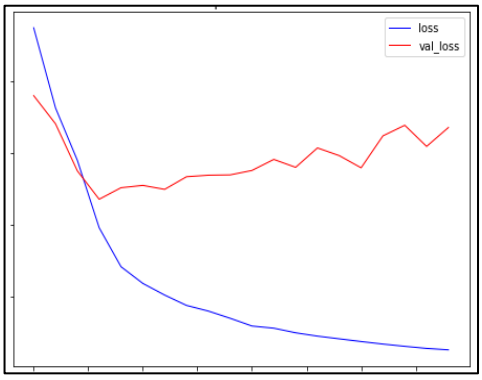


*Figura 29: Una coppia di curve di loss risultanti dall'addestramento di una rete. Notare l'incremento di val_loss dovuto all'*overfitting.

Il *batch size* è fissato a 512 campioni per ottenere aggiornamenti dei parametri poco rumorosi e per consentire un corretto funzionamento del metodo di *batch normalization* impiegato in alcune delle architetture testate (si rimanda all'esperimento 1 per una spiegazione del metodo)

## 3.6. Metodologia di validazione dei risultati

Si riprende e si amplia qui quanto accennato nel paragrafo 3.3.2.*Sampling.*

Per poter misurare correttamente le performance di un algoritmo addestrato su un dato *training set* è necessario testarlo su dati che non ha mai visto. Questo vale tanto per decidere quando l' addestramento è completato (*validation*) quanto per il testing vero e proprio: per evitare di falsarne l'esito queste due operazioni devono tuttavia essere svolte su insiemi diversi, cioè i già citati *validation set* e *test set.*

A complicare ulteriormente le cose, è possibile che le performance ottenute differiscano a seconda dei sottoinsiemi del dataset presi in considerazione, perciò è necessario ripetere la valutazione per diverse scelte di *training, validation e test set* e collezionare tutti i risultati. Si fa notare che *training* e *validation* vengono estratti

insieme (*trainval set*) dal dataset principale, simulando la condizione realistica in cui si è costruito un insieme di dati di "calibrazione" per un nuovo soggetto, e poi separati successivamente.

Infine, ci si aspetta che le performance migliorino funzionalmente alle dimensioni del *trainval set*, aumentando il numero potenziale di informazioni sul compito da svolgere che l'algoritmo può ottenere; tuttavia, in uno scenario di implementazione effettiva del modello, ottenere una grande quantità di dati per costruire il "*calibration set*" con cui adattarlo all'utilizzatore della BCI potrebbe essere molto oneroso. E' quindi di interesse capire qual è la dipendenza dell'algoritmo dalla dimensione del *trainval set.*

Per soddisfare quanto detto finora, ogni esperimento viene validato per ogni scimmia nel modo seguente:

1. Viene assegnato un numero k di trial scelti randomicamente per ciascuna classe all'insieme TRAINVAL ed i restanti 10-k al TEST. Questo punto è portato a termine dal componente di *Sampling* della *mid-validation pipeline.*
2. Si entra nel ciclo detto di *cross-validation*. TRAINVAL è diviso in NS *splits* di campioni random con il metodo *StratifiedKFold* della libreria *Scikit-Learn* (divisioni scelte in modo da mantenere il bilancio tra i campioni associati alle diverse classi), NS-1 usati per il *training* ed uno per la *validation*. Per ogni passaggio degli NS definiti, il modello da valutare su *TEST* viene scelto tra tutti quelli addestrati in base alle performance sullo *split* di validazione. OUTPUT: Le metriche ottenute da questo test, per un totale di 5 insiemi di metriche.
3. Si ritorna al punto 1, ricampionando casualmente un numero *k* di trial da ciascuna classe per P volte. Per ogni k si ottengono quindi NS*P insiemi di metriche.

In tutti gli esperimenti NS=5, k viene scelto da K=[1, 3, 5, 7, 9] e P=10 (tranne in E3, dove P=3 per la lunghezza dei calcoli richiesti). La ripetizione del processo di estrazione di k trials da ciascuna classe per P volte vuole simulare P diversi possibili *calibration set*.

Per ogni k, ciascuno degli NS*P insiemi di metriche estratte in questo modo vuole rappresentare la performance di un modello costruito su *calibration set* differente; la distribuzione delle metriche per ogni k darà quindi informazioni sulla generale affidabilità dell'algoritmo proposto.

# 4. ESPERIMENTI

## 4.1. Esperimento 1 – Single-class decoder

Il primo esperimento ha i seguenti obiettivi:

1. Valutare l'applicabilità delle *Deep Neural Networks* nel decoding dell'intenzione di movimento, nei termini di presenza (label 1) o assenza (label 0) di una intenzione di RTG generica (single-class);
2. Confrontare tre reti rappresentanti delle più diffuse architetture di DNN impiegate nell'ambito delle *time series:*
    a. Una composta di soli strati neuronali standard (*dense*)
    b. Una rete convolutiva strutturata per l'analisi temporale *(cnn)*
    c. Una con strati ricorrenti implementati con il modello *Gated Recurrent Unit (gru)*.

Per tentare di dare a tutte le architetture la stessa possibilità di eccellere, si sono stabiliti alcuni criteri comuni per la loro costruzione, elencati di seguito.

### 4.1.1. Criteri architetturali comuni

#### Criterio 1: numero di strati

Tutte e tre le reti posseggono quattro strati:

1. i primi due cambiano a seconda dell'architettura (*dense, cnn, gru*) e fungono idealmente da *feature extractors.*
2. il terzo strato è sempre composto da 100 neuroni "dense", con la funzione di aggregatore e classificatore.
3. il quarto comprende il solo neurone di uscita.

#### Criterio 2: Rectified Linear Units

Tutti i neuroni degli strati non ricorrenti sono *Rectified Linear Units (ReLU,* visibile a destra nella *Figura 30),* ossia utilizzano la funzione di attivazione impiegata da Krizhevsky nella sua AlexNet [51] poiché in generale ritenuta capace di velocizzare l'apprendimento e migliorare le prestazioni nei più disparati settori di applicazione [55] [56].

La ReLU è stata progettata per compensare il problema del *vanishing gradient*: a differenza della normale attivazione sigmoidale, la cui derivata diminuisce con

l'avvicinarsi ai valori estremi (la già citata saturazione), la funzione rettificata (*Figura 30)* possiede sempre la stessa derivata per ingressi (somme pesate) positivi; grazie all'annullamento degli ingressi negativi inoltre tende a spingere il network verso la condizione di *sparsity*, cioè la codifica di informazioni in pochi neuroni per volta: poiché una somma pesata positiva è, in linea di principio, tanto probabile quanto una negativa, è lecito supporre che circa metà delle unità ReLU tenderanno ad essere sempre spente con diversa distribuzione per ciascun campione.

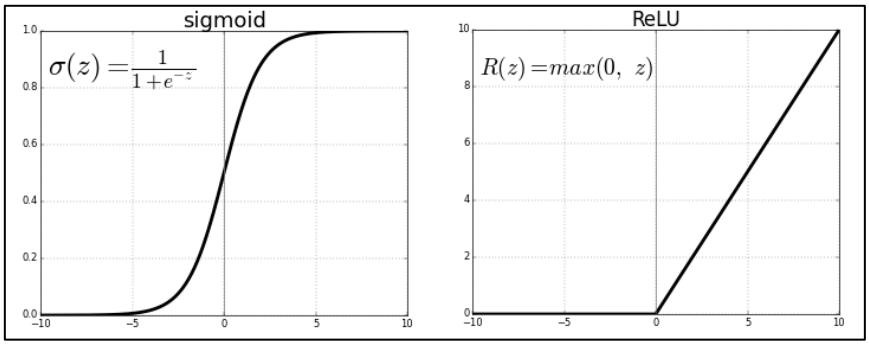


*Figura 30: a sinistra la funzione di attivazione sigmoidale dei neuroni "standard"; a destra la funzione delle unità ReLU.*

Questo è vantaggioso sia per la quantità di informazione immagazzinabile sia per la robustezza dell'immagazzinamento stesso.

### Criterio 3: Dropout

A tutti gli strati è applicato il metodo *Dropout* (*Figura 31*), proposto da Srivastava insieme a Krizhevsky ed Hinton come un rimedio alla tendenza delle reti neurali di "imparare a memoria" le rumorosità del *training-validation set* e sfruttarle per raggiungere performance falsate e non rappresentative di quelle reali. Il metodo consiste nell'escludere totalmente una percentuale random di neuroni dai calcoli sia nel *forward-pass* (ossia nel calcolo dell'uscita) sia nella *back-propagation*, annullandone le uscite e non aggiornandone i pesi.

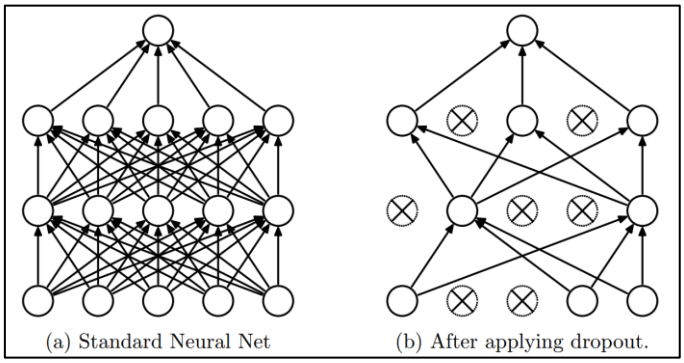


*Figura 31: Raffigurazione del meccanismo di dropout.*

Questo forza la rete ad usare meno neuroni sempre diversi per apprendere i pattern, impedendo la formazione di sottoreti superspecializzate sui dati di training e favorendo invece sparsità e ridondanza di codifica.

A seconda della percentuale di *dropout* questo meccanismo può essere più o meno potente: il valore giusto non è banale da ottenere, poiché un rateo troppo basso potrebbe non avere alcun effetto evidente (estremo: 0, nessun drop) ed uno troppo alto potrebbe non lasciare abbastanza neuroni "vivi" da apprendere il pattern evidenziato (estremo: 1, drop totale).

### Criterio 4: Ricerca iperparametrica automatica con Hyperas

Tutti gli iperparametri delle architetture il cui *tuning* è stato ritenuto sensato (ad esempio, il numero di neuroni nei primi due strati – verrà meglio specificato in seguito) sono stati decisi automaticamente attraverso l'utilizzo della libreria python Hyperas [57] che implementa in Keras il metodo di ricerca iperparametrica Hyperopt [58]. La ricerca iperparametrica viene fatta sopra una singola estrazione k=1 casuale (10% dataset) dai dati di una delle due scimmie, nell'ipotesi che la struttura basilare dei dati neurali, la "grammatica" dei segnali sia in qualche modo soggetto-indipendente. L'estrazione è così piccola per evitare *data-leakage*, ossia per impedire una raffinazione tale degli iperparametri da *"overfittarli"* sul dataset.
La procedura di decisione è implementata attraverso i seguenti passaggi:

1. Vengono definiti degli spazi di ricerca per gli iperparametri, che possono essere discreti, continui o booleani. Ad esempio: units = {{choice([10, 20, 30])}}.
2. Viene definita una metrica da minimizzare attraverso gli iperparametri. In questo caso la CE Loss sull'insieme di validazione.
3. Hyperas addestra e valuta per 50 volte l'architettura oggetto di studio con diverse combinazioni di iperparametri.
4. Infine Hyperas ritorna l'insieme stimato di iperparametri capace di minimizzare la funzione loss sul sottoinsieme di validazione dell'estrazione random.

Questa procedura vuole assicurare che tutti gli iperparametri architetturali degni di nota siano scelti motivatamente ed in modo da massimizzare le performance delle reti.

### Criterio 5: funzione di uscita

L'uscita della rete è espressa attraverso una funzione sigmoidale. Questo perché si vuole imparare a riconoscere una "probabilità di presenza di intenzione di movimento" e la sigmoide (*Figura 30, a sinistra)* ha caratteristiche tali da tendere, con sufficienti dati, a rappresentare proprio una probabilità di classificazione.

## 4.1.2. Architetture

### Architettura dense

Tutti gli strati di questa architettura sono semplici strati "densi", cioè di neuroni connessi in modo standard – niente connessioni ricorrenti, niente filtri convolutivi, solo uno strato simil-percettrone. L'assenza di particolarità strutturali la rende estremamente duttile, che non è necessariamente un pregio: il fatto stesso di essere capace di assorbire informazioni e pattern molto facilmente la rende suscettibile all'*over-fitting* su rumori caratteristici del training set. Per compensare questa suscettibilità, a tutti gli strati è applicato un *dropout* e tutte le unità sono ReLU.

L'ingresso, costituito da una matrice di 50 bins per 50 canali, deve diventare monodimensionale per essere processato dalla rete: è quindi appiattito (*flattened*) riga per riga. E' importante notare che gli strati dense non posseggono alcuna capacità intrinseca di gestire strutture temporali, quindi non importa l'ordine e la modalità con cui avviene tale appiattimento – basta che sia lo stesso per tutti i campioni.

Di seguito sono riportati gli spazi iperparametrici sottoposti alla ricerca di Hyperas e i corrispondenti risultati (→) dell'ottimizzazione:

- Unità del primo strato: [10, 25, 50, 100] → 100
- Unità del secondo strato: [10, 25, 50, 100] → 50
- Percentuale di droprate di tutti gli strati: *range*(0,1) → 0.74

Ottenendo l'architettura rappresentata nella *Figura 32*.

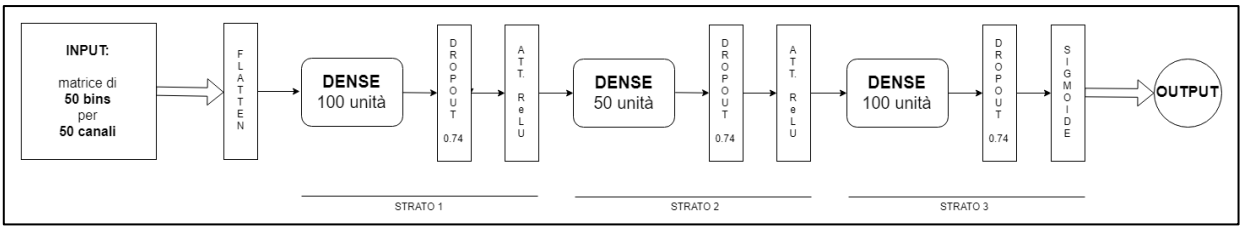


*Figura 32: Architettura* dense

Per un totale di 260351 parametri addestrabili:

- 250100 nel primo strato (2500 valori in input, 100 neuroni con bias)
- 5050 nel secondo strato (100 valori in input, 50 neuroni con bias)
- 5100 nel terzo strato (50 valori in input, 100 neuroni con bias)
- 101 in output (100 valori in input, 1 neurone con bias)

I primi due strati di questa architettura consistono in strati convolutivi monodimensionali. A differenza degli strati convolutivi standard, che lavorano con filtri (*kernel*) bidimensionali su immagini, uno strato convolutivo 1D è costruito per lavorare sopra serie temporali ed impiega quindi un numero stabilito di filtri con la sola dimensione della lunghezza sopra ciascun "canale" in input (*Figura 33*). Questo significa che l'input 50x50 non è analizzato "senza struttura" come nell'architettura dense, bensì i *bins* associati a ciascun neurone vengono effettivamente filtrati con dei *kernel* addestrati a discriminare *patterns* nel tempo.

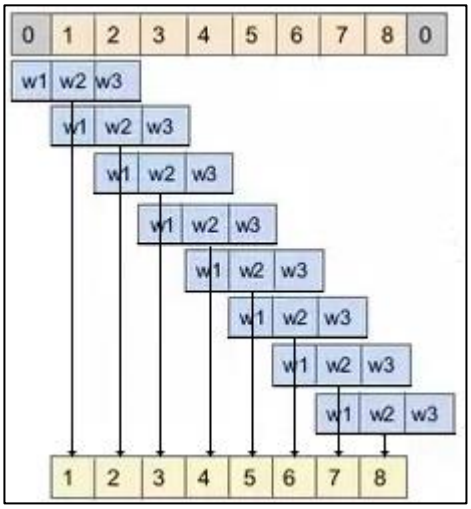


*Figura 33: applicazione di un kernel di uno strato convolutivo 1D sopra un canale dell'input.*

L'utilizzo di uno strato del genere porta due vantaggi:

1. Viene conservata la struttura temporale dell'input, permettendo quindi all'algoritmo di apprendere *features* più utili in maniera più robusta.
2. Nell'ipotesi di un numero limitato di *patterns* temporali discriminatori, il numero di *kernels* convolutivi 1D necessario ad identificarli è potenzialmente piuttosto basso – abbassando così drasticamente il rapporto $\frac{\#parametri}{prestazioni}$ e quindi la complessità computazionale di addestramento e classificazione.

Un terzo vantaggio, non sfruttato in questo lavoro, è l'estrema interpretabilità delle reti convolutive. Consideriamo i seguenti fatti:

- in ogni *kernel* di ciascuno strato, durante l'addestramento, tende a formarsi un *template* di un *pattern* particolarmente discriminatorio e/o frequente nel canale;
- ciascuna singola uscita di uno strato risulta un "indice di rilevamento" del *pattern* rilevato dal *kernel* che l'ha generata;
- tracciando l'influenza sul rilevamento di ciascun valore nel network è possibile capirne l'importanza;

ciò significa che è possibile analizzare i filtri dei vari strati di una CNN (1D e non) per capire quali pattern o combinazioni di pattern nel segnale d'ingresso (in questo caso, binning di spikes neuronali) sono maggiormente discriminatori per l'uscita considerata (in questo caso, l'intenzione di movimento). Esistono numerosi metodi per condurre questo tipo di analisi sulle reti convolutive bidimensionali [59], alcuni dei quali potrebbero essere adattati anche per il caso monodimensionale – soprattutto la cosiddetta *relevance map*, che evidenzia i punti dell'input in cui un particolare filtro ha prodotto uscite di alto valore.

Un difetto delle reti convolutive 1D multistrato consiste nel fatto che i filtri non tengono intrinsecamente conto della temporalità del segnale in ingresso: un pattern viene cercato indistintamente ovunque lungo il segnale, senza consequenzialità, come se il segnale temporale fosse un'immagine spalmata su una linea e non un'uscita di un sistema dinamico con una causalità intrinseca – per questo motivo lo stato dell'arte sull'analisi temporale impiega reti ricorrenti.

Ad ogni strato convolutivo, funzionalmente alle decisioni dell'ottimizzazione Hyperas, vengono associate le ulteriori operazioni di:

- *max-pooling*: raggruppa un numero fissato di attivazioni in uscita dagli strati convolutivi e le riassume in un singolo valore, diminuendo il numero di parametri in ingresso allo strato successivo ed aumentandone il "campo recettivo temporale" (come già accennato in 2.3.3.).
- *batch normalization (BN)*: normalizza l'uscita i-esima dello strato sottraendo la *batch mean* e dividendo per la *batch standard deviation*, cioè media ed std delle uscite i-esime calcolate per ciascun campione del *batch* attualmente in analisi. Questo incrementa la stabilità della rete neurale, ne aiuta le capacità di apprendimento decrementando attivazioni esagerate ed esaltando quelle deboli ed aiuta a compensare l'*overfitting* (attraverso l'introduzione di "rumore" dovuto alla sottrazione ed alla divisione, operazioni che fanno apparire lo stesso campione come "diverso" a seconda del *batch* con cui è presentato nel caso di rimescolamenti). [60]

L'ultimo strato denso funge da classificatore, combinando le *features* estratte dai *layers* convolutivi e deducendone l'output.

Di seguito sono riportati gli spazi iperparametrici sottoposti alla ricerca di Hyperas e i corrispondenti risultati (→) dell'ottimizzazione:

- Numero di filtri nel primo strato: [5, 10, 20, 50] → 50
- Numero di filtri nel secondo strato: [5, 10, 20, 50] → 50
- Lunghezza dei filtri nel primo strato: [5, 10, 20, 25] → 10
- Lunghezza dei filtri nel secondo strato: [5, 10, 20, 25] → 10
- Presenza e lunghezza del pooling: [0, 2, 5] → 2
- Presenza di BN dopo gli strati convolutivi: [0, 1] → 1 (presente)
- *Drop-rate* negli strati convolutivi: *range*(0, 1) → 0.8
- *Drop-rate* nello strato denso di classificazione: *range*(0, 1) → 0.3

Ottenendo l'architettura rappresentata nella *Figura 34.*

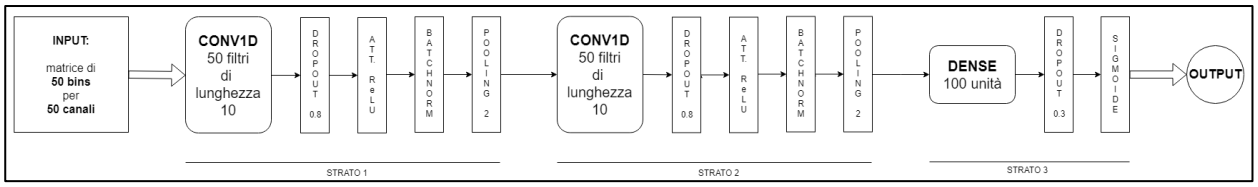


*Figura 34: Architettura* cnn.

Per un totale di 110701 parametri addestrabili:

- 25250 nel primo strato convolutivo (200 dovuti ai parametri *BN*)
- 25250 nel secondo strato convolutivo (200 dovuti ai parametri *BN*)
- 60100 nello strato denso ( *floor(50/2/2)*=12 x 50 in input a causa del pooling)
- 101 in output

## Architettura gru

L'architettura Gated Recurrent Unit è una particolare implementazione delle unità Long Short Term Memory che possiede un numero inferiore di *gates* interni (*Figura 35)*, abbassando quindi il carico computazionale del suo addestramento [61]. Le sue performance sono generalmente comparabili a quelle delle reti LSTM, anche se sembrano ottenere risultati migliori su dataset di piccole dimensioni [62].

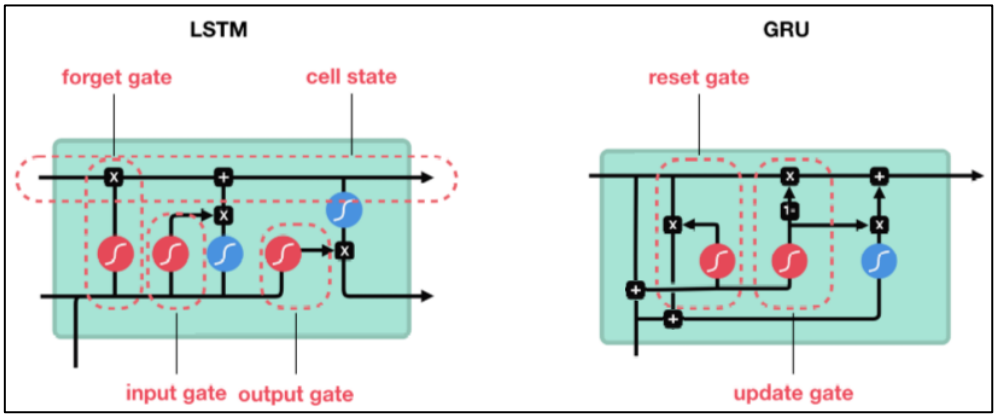


*Figura 35: Differenze tra LSTM e GRU. In rosso, sigmoidi; in blu, funzioni tanh.*
*L'ingresso sotto l'unità LSTM/GRU proviene dal layer precedente, mentre quello a sinistra dallo* stato interno precedente*; la cella LSTM differenzia lo stato in "stato della cella" (in alto) e "stato nascosto" (in basso).*

Considero la cella LSTM della *Figura 35* per analizzare i *gates:*

- *Forget gate:* la funzione sigmoidale (in rosso) prende in ingresso *l'hidden state* precedente concatenato con l'input attuale ed emette in output un valore compreso tra 0 *(forget)* ed 1 *(memorize)*, che poi moltiplica il *cell state*.
- *Input gate:* la sigmoide mappa i valori della la concatenazione [input, hidden state] tra 0 (non importante) e 1 (importante); la tanh mappa l'ingresso tra -1 ed 1, di modo che nessuna quantità arrivi ad "esplodere" e prevalere sulle altre. Il loro prodotto fornisce un "ricordo" con informazioni pesate per valore.

- *Cell state:* se l'uscita del *Forget gate* è prossima a 0, lo stato della cella viene azzerato; dopodiché viene sommato punto-punto con l'uscita dell'*Input gate*, aggiornandolo così con le informazioni in ingresso ritenute di valore dalla cella.
- *Output gate:* decide il prossimo *hidden state* della cella LSTM (utilizzato per modellare la dinamica dell'ingresso e per fornire le predizioni) passando la concatenazione [input, *hidden state (precedente)*] attraverso una sigmoide e moltiplicandola per il *cell state* passato attraverso una tanh. L'*hidden state* è quindi frutto dello stato della cella modulato dall'ingresso attuale e dall'*hs* precedente.

Ciascuna delle funzioni nominate sinora ha ovviamente un numero di pesi e *bias* proporzionale alle dimensioni dell'input in ingresso, determinando celle con numerosissimi parametri associati. La cella GRU adotta il seguente schema semplificato:

- *Update gate:* aggrega *Input gate* e *Forget gate* in un unico passaggio, confrontando direttamente *hidden state* ed input corrente ed aggiornando l'*hs* unicamente se l'input è "abbastanza rilevante".
- *Reset gate:* se l'input non è per nulla relazionato all'*hs* (e quindi è decisamente informativo) l'uscita della sigmoide risulta nulla; il reset gate aggiunge a questa uscita nulla l'input non modificato, in modo da "resettare" l'*hidden state* della cella.

Il numero di *gate*, di funzioni associate e quindi di parametri è così decisamente ridotto.

Queste celle sono costruite appositamente per modellare strutture temporali, quindi sono lo *standard de facto* per la classificazione e la predizione di serie temporali come quella oggetto del presente studio. Ci si aspetta quindi la miglior performance possibile da questa architettura – a scapito di un tempo di addestramento superiore dovuto alla *back-propagation through time (BPTT)* e ad una maggior necessità di dati.

Nel caso specifico, l'input 50 *bins* x 50 neuroni è presentato agli strati ricorsivi in termini di "50 stati successivi di 50 neuroni", quindi la rete apprende tramite *BPTT* ben 50 "*hidden layers"* per ciascuno strato ricorrente! In assenza della memoria interna (=*gates*) di GRU, Il *vanishing gradient* impedirebbe qualsiasi tentativo di addestramento.

Come per l'architettura cnn viene fornita all'ottimizzatore Hyperas la possibilità di assegnare ai primi due strati un'operazione di *batch normalization* per aiutare la convergenza. Il *pooling* non ha invece alcun significato per gli strati ricorrenti, quindi non viene proposto.

L'ultimo strato denso funge nuovamente da classificatore, combinando le *features* temporali dedotte dagli strati ricorrenti (o, per meglio dire, gli stati interni stimati) per definire un output.

Di seguito sono riportati gli spazi iperparametrici sottoposti alla ricerca di Hyperas e i corrispondenti risultati (→) dell'ottimizzazione:

- Numero di unità nel primo strato GRU: [5, 10, 20, 50, 100] → 20
- Numero di unità nel secondo strato GRU: [5, 10, 20, 50, 100] → 50
- Dropout negli strati ricorrenti: *range*(0, 1) → 0.58
- Dropout nello strato denso: *range*(0, 1) → 0.87
- Presenza di *BN* dopo gli strati GRU: [0, 1] → 1 (presente)

Ottenendo l'architettura rappresentata nella *Figura 36:*

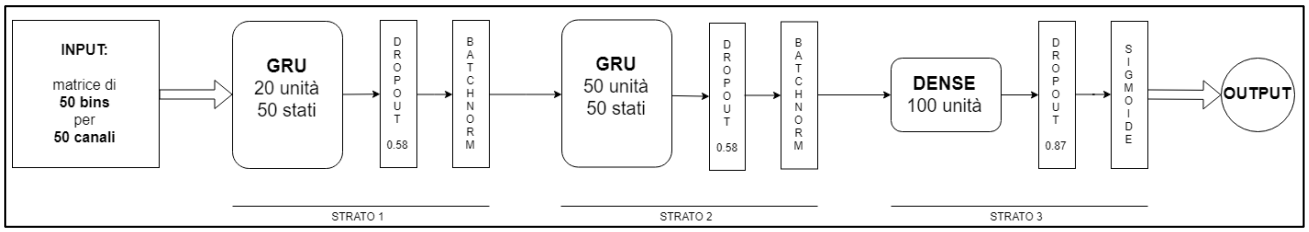


*Figura 36: Architettura* gru.

Per un totale di 265601 parametri addestrabili:

- 4400 nel primo strato GRU (80 per i parametri *BN*)
- 11000 nel secondo strato GRU (200 per i parametri *BN*)
- 250100 nello strato denso
- 101 in output

La rete è stata implementata con la tecnologia CuDNN di NVIDIA, capace di rendere gestibili su GPU i calcoli delle GRUnits, normalmente svolti su CPU a causa di una ridotta parallelizzabilità multi-core rispetto alle altre architetture.

### 4.1.3. Risultati dell'esperimento

Recapitolando, l'esperimento 1 coinvolge:

- 2 macachi, dai quali sono stati registrati rispettivamente 47 e 32 neuroni;
- 50 trials Reach-to-Grasp, 10 per oggetto;
- Una sola classe, intenzione di movimento presente (1) o assente (0);
- L'intenzione è ricercata nei 500 ms precedenti il key-up;

- 3 architetture DNN (*dense*, *cnn*, *gru*) aventi in ingresso una finestra traslante di forma [50 neuroni (*padded*), 50 *bins*] ed in uscita un valore tra 0 ed 1;
- P=10 estrazioni di k trials randomici per ciascun oggetto, con k preso da K=[1, 3, 5, 7, 9], presi come *training-validation set* per le architetture, seguentemente testate sugli altri 10-k trials rimasti per ciascun oggetto;
- Ciascun test è svolto come *cross-validation* con Nsplits=5 splits;
- *Precision*, *Recall*, *F-Score*, *Area Under Curve* come metriche di uscita.

L'esperimento ha in output, per ciascuna scimmia, per ciascun *sampling* k in K, per ciascuna architettura, P*Nsplits = 50 misurazioni delle performance architetturali – rispettando la procedura di validazione evidenziata nel capitolo di materiali e metodi.

L'intero processo è stato completato in (circa) 5 ore per *dense*, 5 ore per *cnn*, 6 ore per *gru*.

Nelle prossime pagine sono riportati i risultati sotto forma di *violin plots*, dove ciascun "violino" è associato ad una metrica di una distribuzione di 50 campioni ed assume la forma della *probability density function* (*pdf*) stimata di tale distribuzione.

Si fa notare che la forma dei *violins* è un indice della robustezza del modello rispetto alla metrica: più uno di essi è "schiacciato" più l'architettura che rappresenta è capace di dimostrare la stessa performance su diversi insiemi di training, validation e test.

Precision

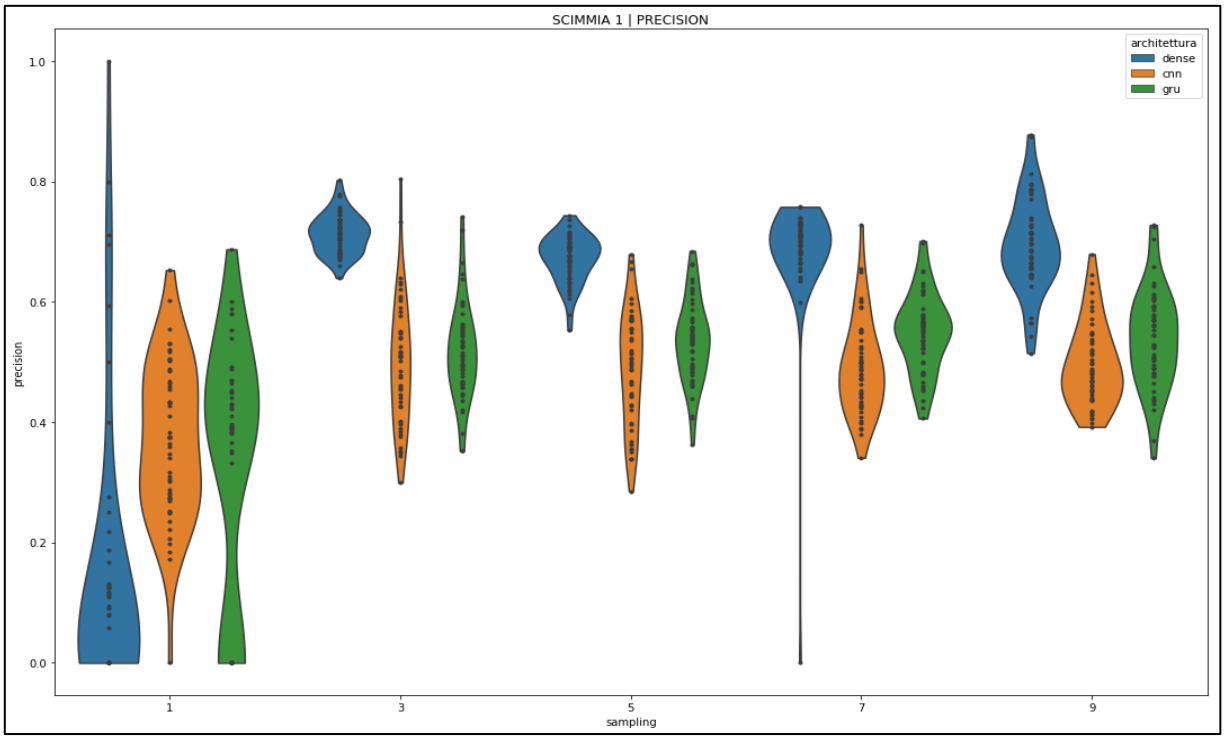


*Figura 37: PRECISIONs ottenute dalle architetture per ciascuna scelta di sampling k (scimmia 1)*

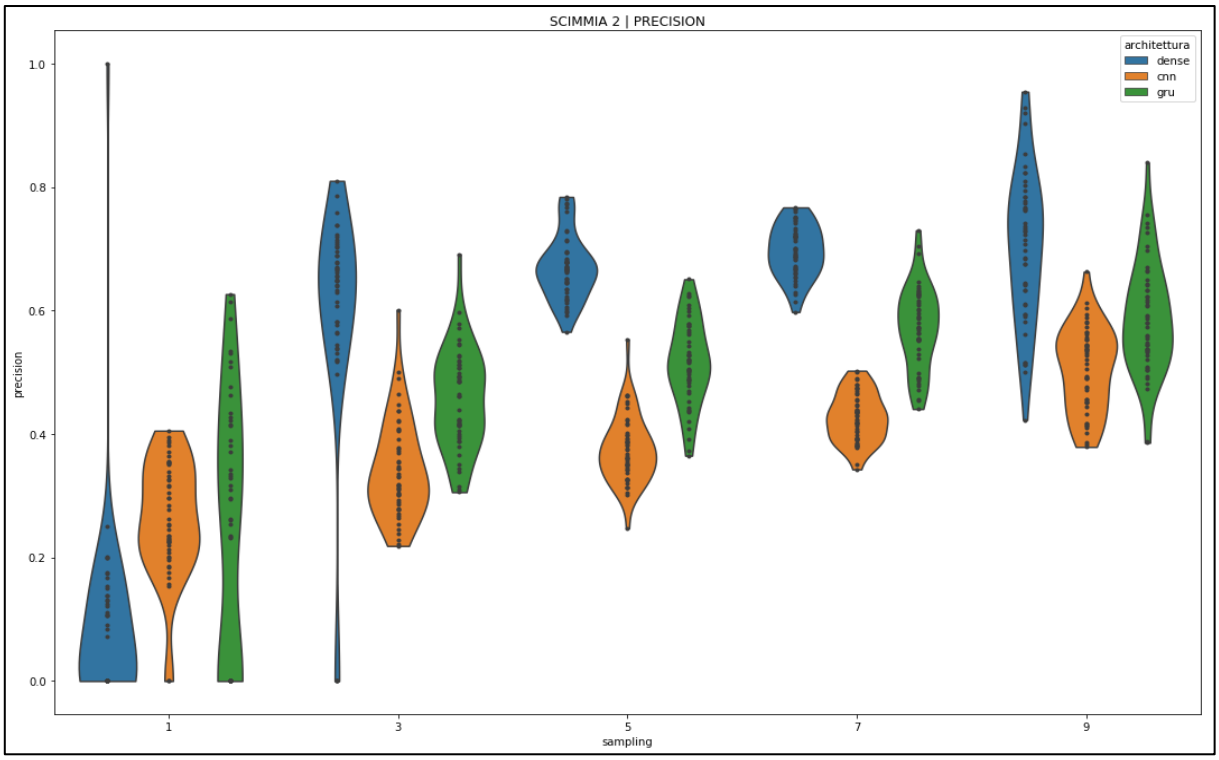


*Figura 38: PRECISIONs ottenute dalle architetture per ciascuna scelta di sampling k (scimmia 2)*

Sorprendentemente, l'architettura più precisa (capace di minimizzare i falsi positivi sul totale dei positivi rilevati) sembra essere la *dense*, su entrambe le scimmie e per tutti i sampling; sorprendente in particolare la sua performance con sampling del 30%.

Non è detto che sia un bene: nel specializzarsi a riconoscere unicamente *true positives* la rete potrebbe aver imparato ad attivarsi pochissime volte, rendendosi inutile in un contesto applicativo.

La peggiore architettura in termini di precisione è la *cnn*, che tuttavia dimostra una certa compattezza nella distribuzione dei risultati.

Considerando il caso del minimo insieme di *training-validation* (sampling di un solo trial per oggetto, quindi 10% del dataset) la *dense* risulta invece l'architettura meno stabile, probabilmente a causa del grande numero di parametri non guidati nell'addestramento da nessun tipo di struttura temporale; *cnn* e *gru* hanno prestazioni simili per la scimmia 1, mentre *cnn* risulta decisamente più robusta (anche se in media meno performante) per la scimmia 2.

E' importante ricordare che *cnn* possiede meno della metà dei parametri delle altre due architetture: in termini assoluti, questo significa che ha meno *capacità potenziale* di apprendere strutture. Nell'ipotesi della presenza di patterns più o meno fissi e distinguibili, la sua struttura convolutiva dovrebbe aiutarla – dai prossimi grafici si cercherà di capire quanto questa ipotesi sia sostenibile.

## Recall

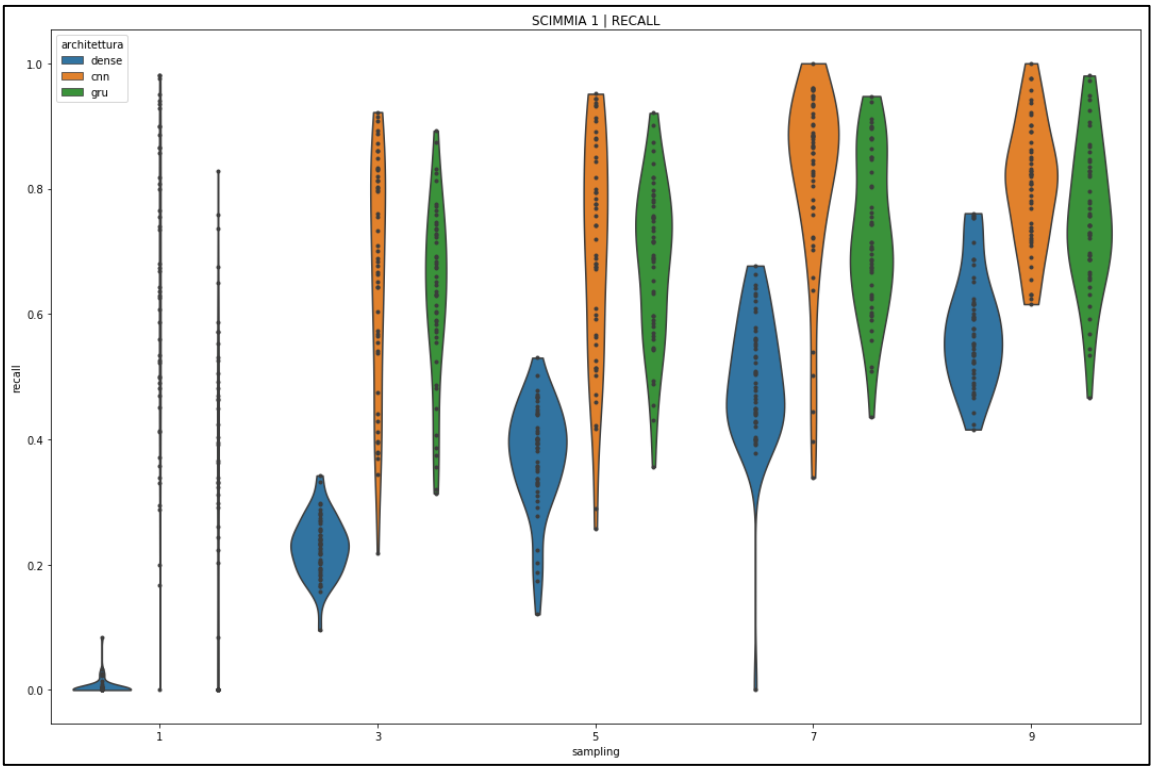


*Figura 39: RECALLs ottenuti dalle architetture per ciascuna scelta di sampling k (scimmia 1)*

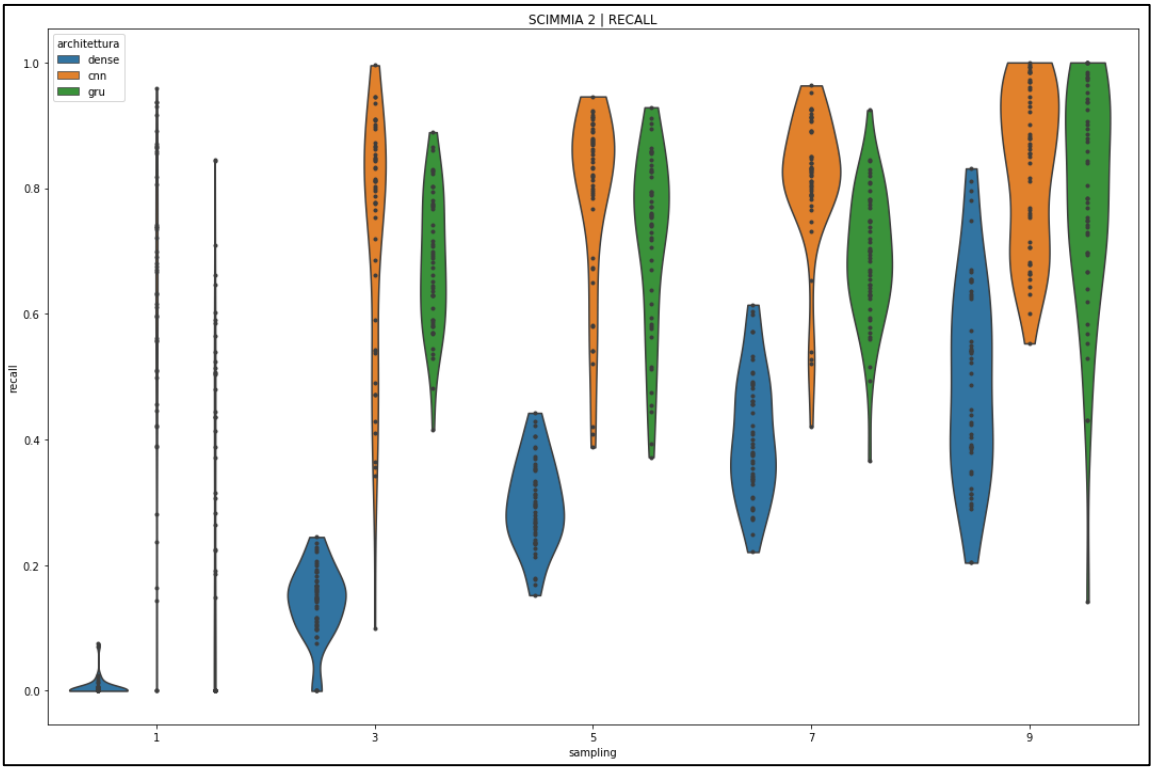


*Figura 40: RECALLs ottenuti dalle architetture per ciascuna scelta di sampling k (scimmia 2)*

L’architettura *cnn* si dimostra la migliore in *recall* (quantità di veri positivi riconosciuti sul totale dei presenti nei *test set*) sia in termini di performance che di robustezza: ciò significa che, malgrado l’inferiore numero di parametri, i modelli *cnn* sono riusciti ad imparare il maggior numero di informazioni (nel suo caso, quindi, di *patterns* temporali) identificative per l’intenzione di movimento.

Le reti *dense* si sono invece dimostrate le meno capaci nel riconoscere tutti i casi positivi, mostrando che la notevole precisione rilevata precedentemente è stata ottenuta grazie ad un’alta “soglia di accettazione” o, nel caso peggiore, ad un mancato apprendimento di tutte le informazioni associate all’intenzione di movimento.

L’architettura *gru* ottiene risultati similari alla *cnn*, con tuttavia meno robustezza complessiva.

Il caso con k=1 mostra che le architetture non sono assolutamente in grado di ottenere risultati coerenti con così pochi dati nel compito di detezione dell’intenzione, fissando così a 3 trials per oggetto la quantità minima rilevata per una performance accettabile.

## F-Score

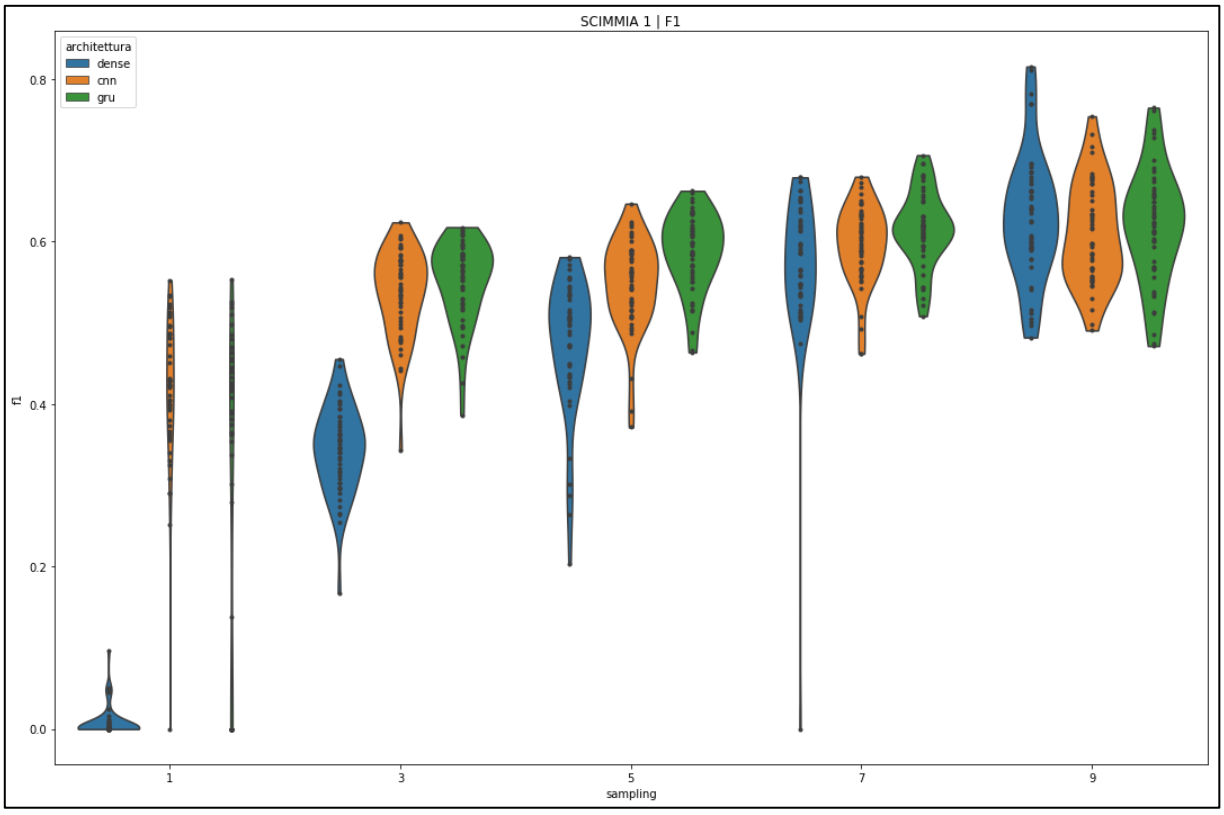


*Figura 41: F-SCOREs (F1) ottenui dalle architetture per ciascuna scelta di sampling k (scimmia 1)*

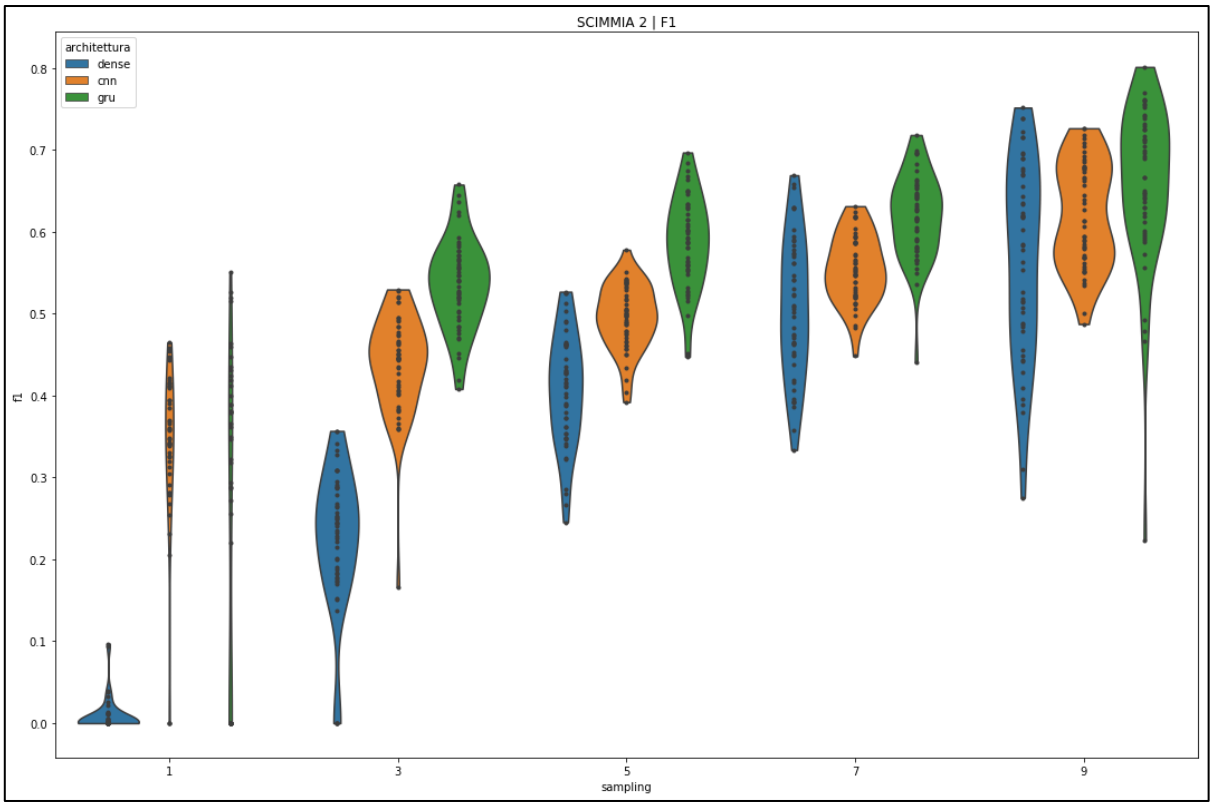


*Figura 42: F-SCOREs (F1) ottenuti dalle architetture per ciascuna scelta di sampling k (scimmia 2)*

La metrica *F-Score* (o *F1 measure*) riassume *precision* e *recall* con la loro media armonica. Da questo sunto il modello *gru* risulta la migliore sia in termini di performance che di robustezza (compattezza dei *violins*), dimostrando la già prevista superiorità delle reti ricorrenti nel compito di classificazione di serie temporali. Particolarmente rilevante la differenza di performance con k=3, che dimostra la viabilità di impiegare 3 sole sessioni di calibrazione per soggetto per ottenere un F-score intorno allo 0.5 – ciò significa, interpretando blandamente, che il modello riesce distinguere e classificare correttamente circa metà del totale dei positivi in un dataset del quale rappresentano solo il 15%.

A seguire, l'architettura *cnn* dimostra performance inferiori alla rete *gru* ma robustezza superiore. Riesce a raggiungere un risultato soddisfacente soltanto con 7 trials per oggetto, ma considerando i vantaggi del numero di parametri dimezzato rispetto alle altre proposte e la potenziale interpretabilità del modello quanto ha ottenuto è rilevante.

L'architettura *dense*, a causa dei bassissimi *recall* ottenuti dai suoi modelli, dimostra infine una performance ed una robustezza decisamente inadeguate al compito.

## Area Under Curve (AUC)

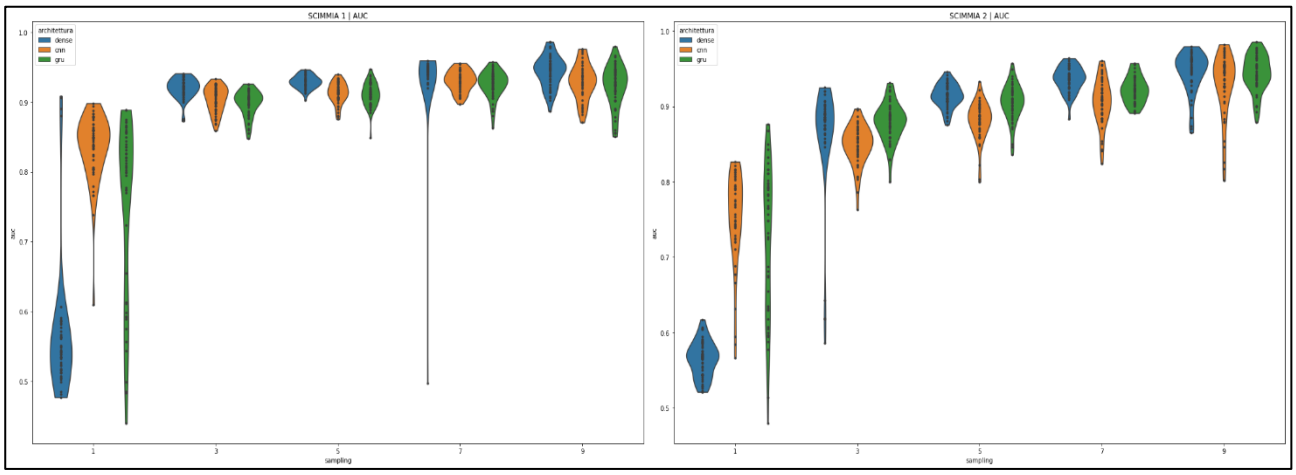


*Figura 43: AUCs ottenute dalle architetture per ciascuna scelta di sampling k.*

Si ricorda che *precision*, *recall* ed *F-Score* sono calcolate con una soglia sulle uscite di 0.5, ma non necessariamente questa è la migliore possibile: come spiegato definendo la metrica AUC, variando tale soglia è possibile calibrare il modello facendolo tendere verso una maggior *sensibilità* (*recall*) piuttosto che *specificità (precision*). Questo è mostrato graficamente dalla curva ROC, che tenderà a sottendere un'area (AUC) maggiore a seconda del generale potere discriminatorio del modello.

Il *violin-plot* delle AUC dei modelli vuole dimostrare che, potenzialmente, tutte le architetture possiedono un alto potere discriminatorio, per di più similare tra tutte le proposte – ma è importante notare che le soglie per il calcolo sono scelte scorrendo sopra i risultati ottenuti con il *test set*, quindi le AUC mostrate non sono indicative di una "performance massima ottenibile in addestramento" in quanto questa sarebbe espressa scegliendo le soglie sul sottoinsieme di validazione.

In questi termini, l'architettura che mostra generalmente più potenziale su tutti i sampling sembra essere la *cnn*, capace di distribuzioni di AUC sufficientemente compatte e di alto valore sin dai sampling più bassi.

Quest'ultima considerazione, insieme ai robusti risultati ottenuti in termini di F-score, il basso numero di parametri e la potenziale interpretabilità, motiva la scelta dell'architettura *cnn* per i prossimi esperimenti.

## 4.2. Esperimento 2 – Multi-class Decoder

Il secondo esperimento ha il duplice scopo di:

1. Verificare la fattibilità di un classificatore DNN capace di distinguere intenzione di RTG per ciascuno dei 5 oggetti malgrado l'estremo sbilanciamento del dataset;
2. Confrontare un approccio di multiclassificazione "diretto" ed uno "composito":
    a. Il modello diretto prevede una sola rete addestrata in modo equivalente a quanto fatto nell'esperimento 1, ma con 5 uscite (ciascuna predicente la presenza di un'intenzione specifica - 1: palla, 2: maniglia, 3: anello, 4: piastra, 5: cilindro nella fessura)
    b. Il modello composito è invece costituito da una DNN "precisa-monoclasse" di rilevamento di una generica intenzione (=E1) ed una DNN "estesa-multiclasse" capace di distinguere i segnali neurali legati ai diversi oggetti ma con inferiore precisione temporale.

A causa della complessità del compito è improbabile che DNN poco profonde come quelle proposte, costruite con in mente l'esplorazione del problema piuttosto che una sua soluzione ottimale, siano in effetti in grado di ottenere risultati soddisfacenti; tuttavia, nel caso le reti mostrassero un qualsiasi *trend* di apprendimento sufficientemente robusto all'aumentare del numero di trials impiegati per addestramento e validazione allora il risultato sarebbe da considerarsi positivo: questo infatti legherebbe le performance alla quantità di dati forniti e/o all'architettura della rete in ingresso piuttosto che all'approccio errato, motivando quindi studi successivi.

A causa dei risultati conseguiti nell'esperimento 1 verrà impiegata l'architettura *cnn*.

### 4.2.1. Modelli impiegati

#### Modello diretto

Il modello multiclassificatore "diretto" consiste nella stessa architettura *cnn* impiegata nell'esperimento 1 con l'uscita modificata in modo da consentire 5 diverse uscite, una per oggetto. L'ultimo strato consiste quindi in 5 neuroni con funzione di uscita *softmax*, una generalizzazione della precedentemente impiegata funzione sigmoidale:

$$\mathrm{o}(\boldsymbol{u})_j = \frac{e^{u_j}}{\sum_{n=1}^{5} e^{u_n}} \quad \text{per} \quad \mathrm{j} = 1\,,\ \ldots\,,\ 5$$

La funzione è costruita in modo che la somma delle uscite sia sempre uguale ad 1, costringendole ad assumere la forma di una "probabilità di appartenenza" ad una classe

con l'ipotesi di classi disgiunte – sensata in questo caso, perché gli oggetti erano diversi uno dall'altro e afferrati in trials separati.

Altro accorgimento adottato è stato utilizzare la funzione di loss *categorical crossentropy* al posto della *binary* impiegata nell'esperimento 1, in modo da tener conto correttamente degli errori commessi rispetto a ciascuna delle classi.

### Modello composito

Questo modello nasce dal fatto che lo sbilanciamento del dataset rischia di impedire un corretto apprendimento delle classi, ciascuna rappresentata per solo il 3% dei dati e con una "classe 0" che invece ne occupa l'85%; la soluzione proposta cerca di scindere il problema di classificazione in due sottoproblemi rappresentati da percentuali più alte:

- [*mono*]: Classificazione nei 500ms pre-keyup di intenzione di movimento generica, come nell'esperimento 1, la cui classe 1 è data dalla somma delle 5 classi-oggetto e rappresenta quindi il 15% del dataset;
- [*extended*]: Classificazione del singolo oggetto focalizzato dall'animale nell'intervallo di tempo [keyup-1.5, keyup+0.5], comprendente anche le fasi di osservazione e di movimento effettivo, ottenendo così una rappresentazione del 12% per ciascuna classe al prezzo di una inferiore definizione temporale.

Moltiplicando gli output dei due classificatori si ottiene la relazione:

$$\mathrm{out}_{mono} * out_{extended} = P_{intenzione} * P_{focus-oggetto} = P_{intenzione-oggetto}$$

Che dovrebbe approssimare quella ottenuta con la classificazione diretta. I pro di questo approccio sono che il modello dispone di molti più dati sulle classi non-0 per apprenderle e che sono impiegati due algoritmi differenti per ottenere un solo risultato, metodologia chiamata *ensemble learning* e notoriamente capace di ottenere performance migliori di un approccio standard [63]. I contro assicurati e potenziali sono tuttavia molteplici:

1. Il carico computazionale è raddoppiato;
2. L'output di un qualsiasi classificatore è modellabile come $\mathrm{out} = \mathrm{out}_0 + \epsilon$, dove la epsilon rappresenta un errore/rumore di classificazione: moltiplicare l'output di due algoritmi rischia di incrementare la varianza dell'output complessivo.
3. E' probabile che il numero di *patterns* temporali contenuti nel segmento di 2s intorno al key-up impiegato dal classificatore *extended* per ciascun oggetto sia di molto superiore a quello memorizzabile dall'architettura *cnn*, i cui iperparametri sono stati specializzati su un compito temporalmente più ristretto e generico.

### 4.2.2. Risultati dell’esperimento

Si mostrano nelle prossime pagine le distribuzioni della media sulle 5 classi delle metriche di *precision*, *recall* ed *F-Score*. Le distribuzioni di *F-Score* per ciascuna classe sono riportate in appendice.

#### Precision (average)

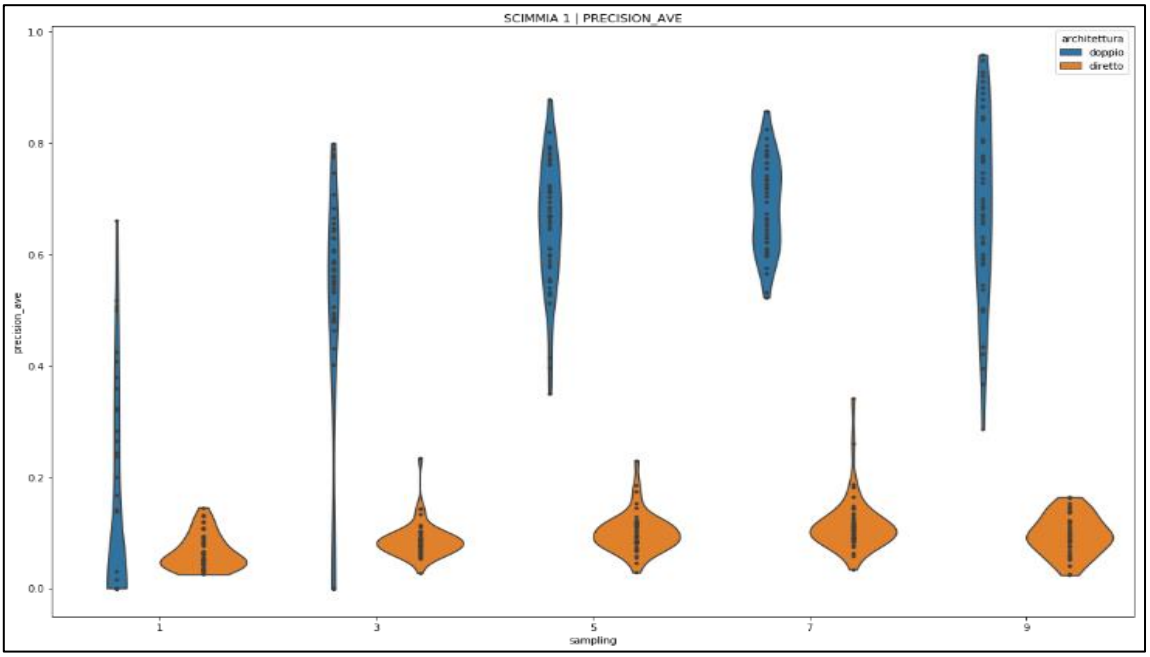


*Figura 44: PRECISIONs (medie) ottenute dai modelli "diretto" e "doppio" (composito) per ogni sampling k (scimmia 1).*

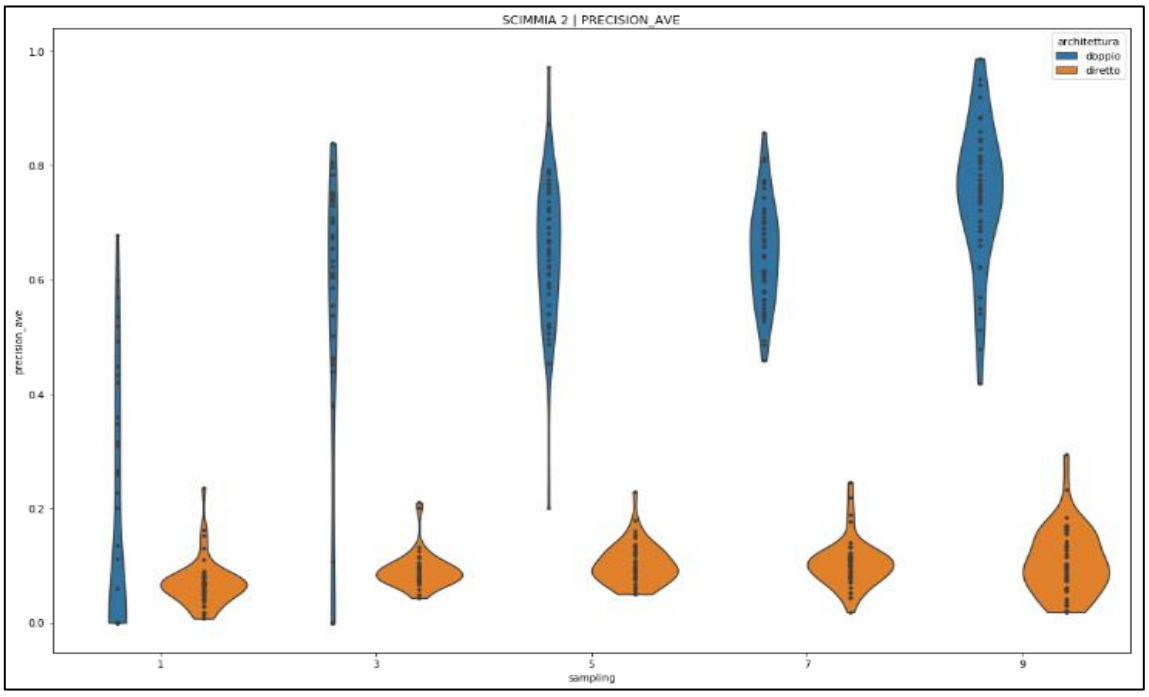


*Figura 45: PRECISIONs (medie) ottenute dai modelli "diretto" e "doppio" (composito) per ogni sampling k (scimmia 2).*

La precisione ottenuta dal modello composito sovrasta quella risultante dal modello diretto, mostrando contemporaneamente l'efficacia dell'approccio *ensemble* e l'incapacità dell'architettura *cnn* scelta di apprendere correttamente da un dataset tanto sbilanciato. La performance maggiore sembra arrivare al prezzo della robustezza, apparentemente confermando il contro della moltiplicazione del rumore di classificazione.

Recall (average)

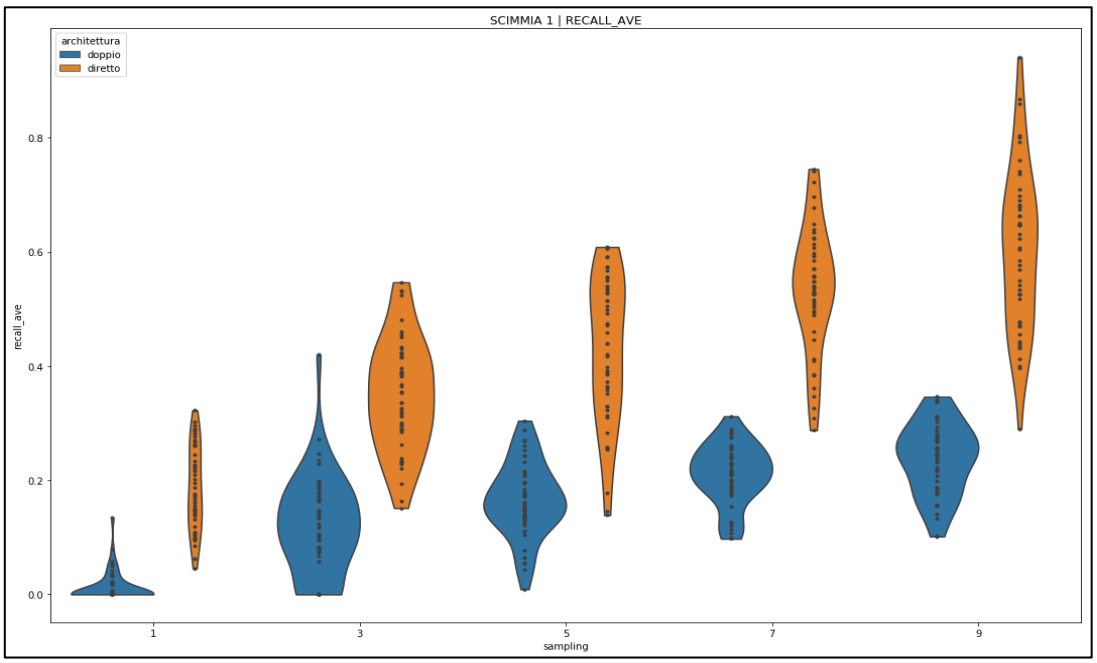


*Figura 46: RECALLs (medi) ottenuti dai modelli "diretto" e "doppio" (composito) per ogni sampling k (scimmia 1).*

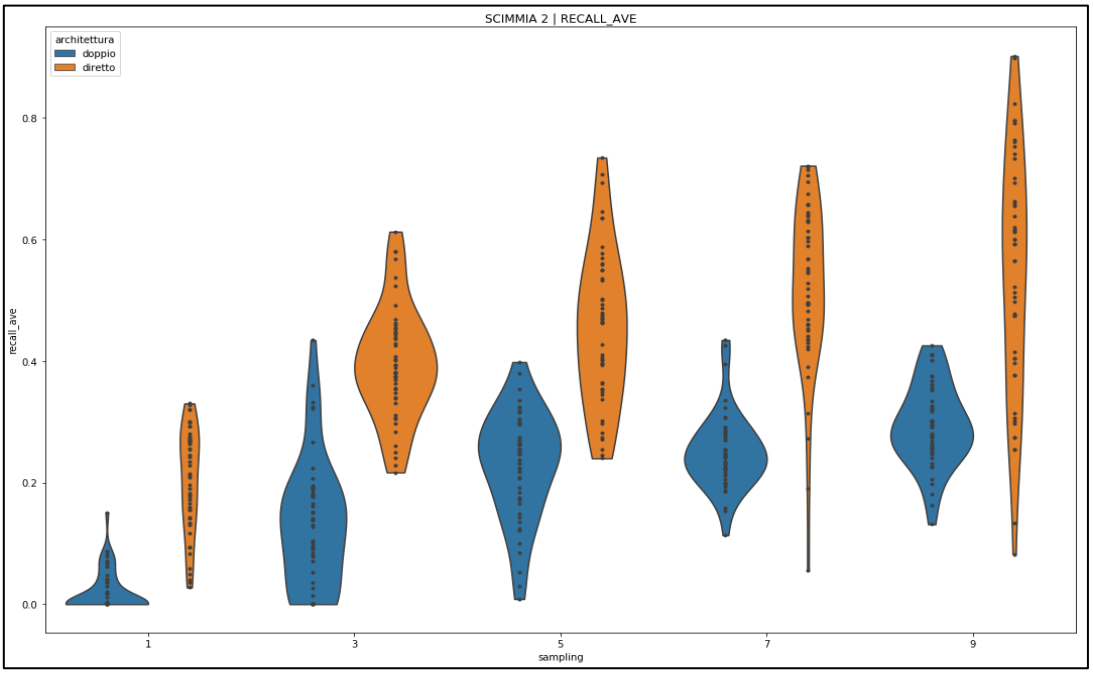


*Figura 47: RECALLs (medi) ottenuti dai modelli "diretto" e "doppio" (composito) per ogni sampling k (scimmia 2).*

Con il *recall* sembra verificarsi invece uno dei contro ipotizzati per il modello composito: l'incapacità della *cnn* di apprendere tutti i *patterns* componenti l'intervallo di 2 secondi intorno al key-up. Il modello diretto mostra una notevole (ma poco robusta) capacità di apprendimento, fallendo tuttavia nel comporli in una predizione efficace (mostrato nei plot di *precision*) – probabilmente avrebbe giovato un altro strato denso alla fine. Sembra insomma che il modello diretto abbia imparato molto sulle classi-oggetto, ma male – e che il composito abbia imparato poco, ma bene.

Si avvisa di accettare le precedenti riflessioni con cautela, sia in quanto si sta ragionando sulla performance mediata su tutte le classi, sia perché è davvero difficile analizzare correttamente aspetti così profondi del modello di apprendimento di un algoritmo a partire dalla distribuzione dei suoi risultati e senza esplorare accuratamente le variazioni architetturali ed applicative di ciascun approccio – studio che esula dall'obiettivo esplorativo del presente lavoro.

In particolare, si fa notare che l'estensione delle classi positive a 2 secondi intorno al key-up richiederebbe un nuovo esperimento 1 dedicato per validare la performance dell'architettura, adattata con Hyperas ai soli 500 ms prima del *flag* e quindi impreparata al compito. L'algoritmo composito dimostra quindi una performance notevole, considerando la mancanza di ottimizzazione.

F-Score (average)

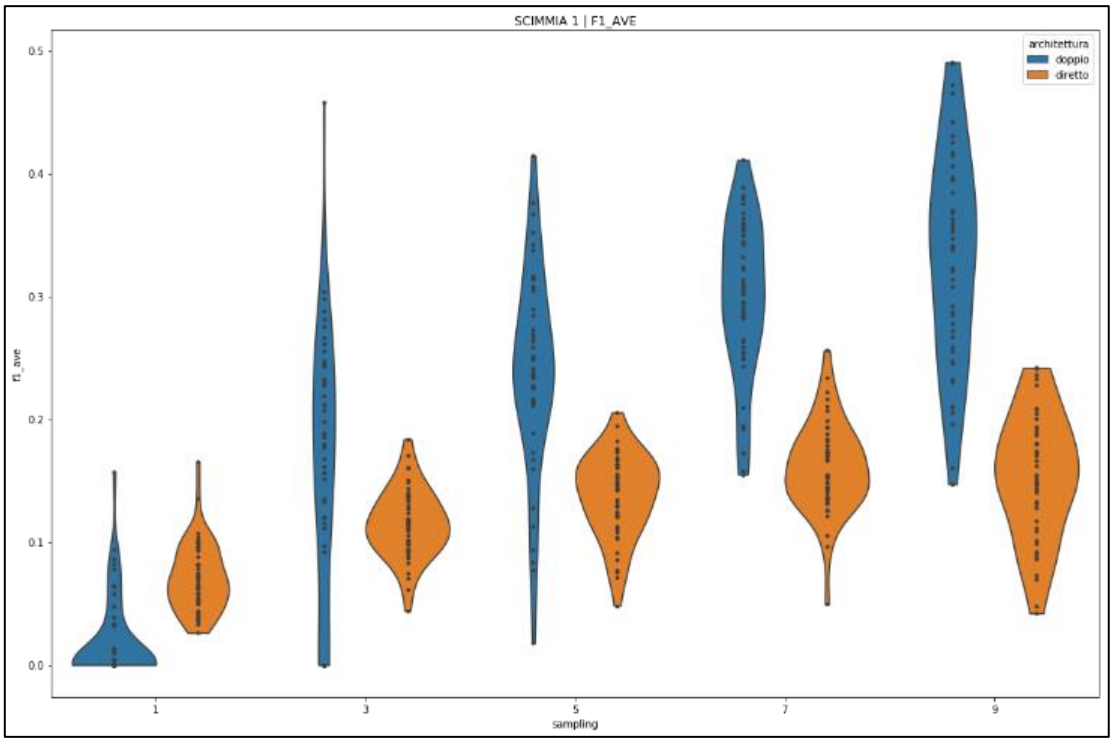


*Figura 48: F-SCOREs (medi) ottenuti dai modelli "diretto" e "doppio" (composito) per ogni sampling k. (scimmia 1)*

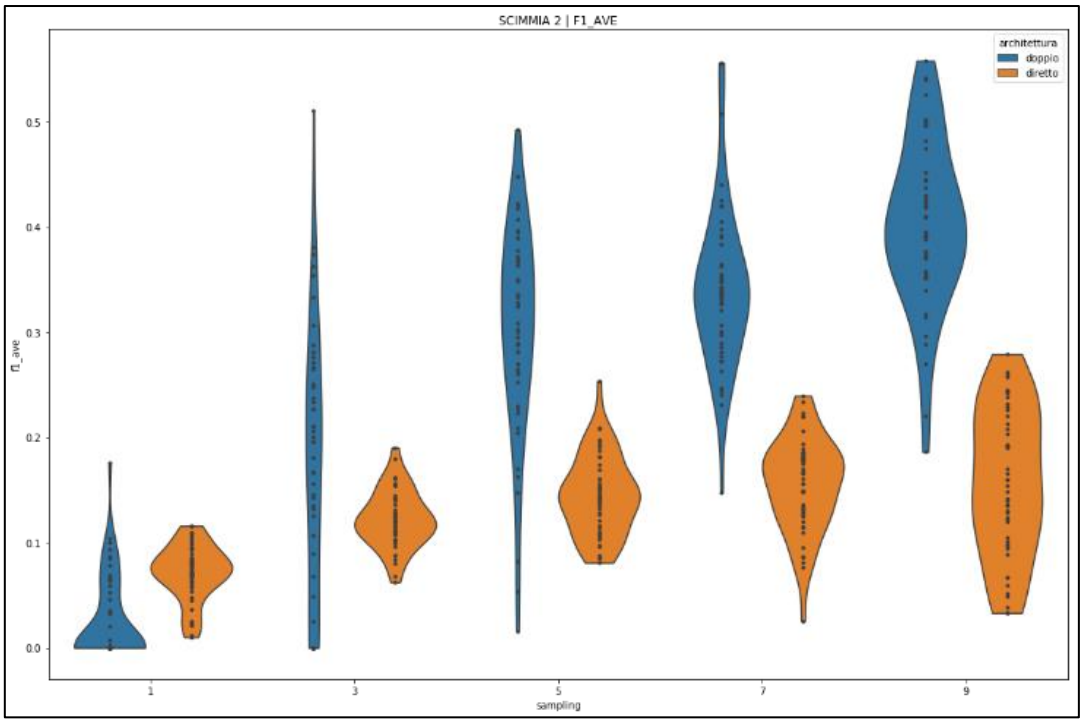


*Figura 49: F-SCOREs (medi) ottenuti dai modelli "diretto" e "doppio" (composito) per ogni sampling k. (scimmia 2)*

In *Figura 48-49* sono riportati con metrica F-score (media) i risultati ottenuti per i modelli diretto e doppio. il modello composito risulta complessivamente il vincitore in termini di performance, al prezzo di una notevole perdita di robustezza (compattezza dei *violin-plots*) rispetto al modello diretto. Malgrado questo, il massimo F-Score medio raggiunto risulta intorno a 0.5, un valore basso considerando la presenza di ben 9 trial di addestramento per ciascuna classe; la differenza tra le distribuzioni degli F-Score di ciascuna classe (mostrati in appendice) evidenzia ulteriormente una performance lontana dall'idealità. Quanto detto suggerisce una scelta architetturale scorretta o incompleta della rete *cnn* e/o una necessità maggiore di dati.

Bisogna inoltre ricordarsi che non è stato fatto un ri-adattamento Hyperas per il caso multiclasse, lasciando l'ottimizzazione ottenuta nel caso single-class: l'ipotesi alla base della mancanza era che la procedura avesse individuato gli iperparametri più adatti alla "struttura" del segnale e che questa fosse ovviamente indipendente dal numero di classi, ma è anche vero che il modo in cui gli iperparametri sono trovati dipende dal valore di loss calcolato rispetto alla/e classe/i forniti all'ottimizzatore. Bisognerebbe quindi ripetere l'esperimento anteponendo una ricerca iperparametrica e confrontando i risultati.

Malgrado i risultati non soddisfacenti, le reti mostrano comunque un corretto *trend* di apprendimento al crescere dei dati, indicando la corretta applicazione dei processi di addestramento e validando ulteriormente le *DNN* per il problema in oggetto.

Un eventuale prossimo esperimento dovrà revisionare l'architettura e probabilmente adottare metodi di *data augmentation.*

## 4.3. Esperimento 3 – Neuron Loss Analysis

L'ultimo esperimento vuole verificare la stabilità di classificazione di intenzione di movimento dell'architettura *DNN* scelta (*cnn*) nel tempo, simulando una progressiva perdita di neuroni acquisiti dal sistema a causa di processi di deterioramento dell'impianto intracorticale causati dalla risposta infiammatoria dell'organismo e successivo incapsulamento fibrotico del dispositivo (*Figura 50*, tratta da [64]).

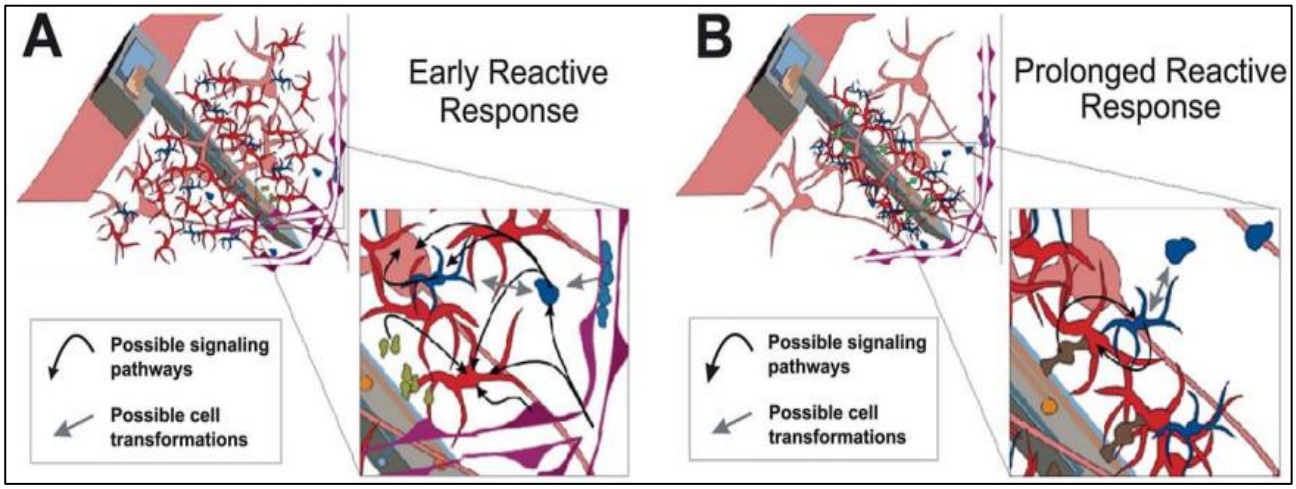


*Figura 50: Risposta fibrotica acuta (sx) e cronica (dx) all'impianto di un elettrodo: il completo avvolgimento dell'elettrodo da parte della cicatrice gliale può portare al suo totale isolamento elettrico, quindi al "neuron loss".*

Lo scopo è cercare di verificare la robustezza della codifica appresa dai modelli basati sull'architettura scelta, al contempo controllando che i risultati effettivamente dipendano dalle informazioni fornite in ingresso e non da errori, rumori o similari. Il modo con cui le *performance* si mostreranno in funzione del numero di neuroni perduti potrebbe anche fornire indicazioni sull'approccio dell'algoritmo all'apprendimento.

Tale scopo sarà perseguito ottenendo un grafico di F-Score medio per ciascun sampling in funzione del numero di neuroni persi a partire dal totale, sia per il caso single-class analizzato nell'E1 sia per la versione multiclasse diretta sviluppata in E2.

Non verrà analizzato il multiclassificatore composito in quanto si sta cercando, con questo esperimento, di verificare la robustezza della codifica dell'intenzione di movimento appresa dalla DNN, a prescindere da come essa possa essere composta in *ensembles*. Inoltre, il componente "preciso" corrisponde al single-class sviluppato in E1, mentre quello "esteso" è una rete multiclasse come nel caso diretto ma addestrato su più dati oltre quelli dell'intenzione, quindi al di fuori dell'analisi perseguita.

### 4.3.1. Descrizione della simulazione

La simulazione è stata costruita sotto le seguenti ipotesi semplificative:

1. L'impianto è fattualmente un array multielettrodo (si rimanda all'ipotesi fatta in metodi rispetto alla possibilità di considerare "acquisite insieme" tutte le *single electrode recordings*);
2. L'addestramento dell'algoritmo avviene con la calibrazione effettuata a dispositivo appena impiantato, quindi con tutti i neuroni correttamente acquisibili;
3. I segnali dei neuroni che non vengono persi non subiscono alcuna modifica (dovrebbero invece riflettere un incremento di impedenza dell'elettrodo, con sempre più spikes risultanti sotto-soglia e quindi una progressiva perdita di *flag* temporali);
4. Ciascun neurone ha una posizione assegnata e fissata nel tempo nella matrice di input all'algoritmo. Questo può realisticamente accadere se gli elettrodi sono numerati e lo *spike-sorting* viene eseguito ordinatamente su di essi con un criterio fissato.

Per ottenere i grafici si svolgerà una procedura di validazione simile a quelle impiegate in E1 ed E2 ma con ciascun test (per ogni cross-validazione, per ogni ripetizione, per ogni sampling) ripetuto sopra un numero $N_n$ decrescente di neuroni disponibili. Per simulare le numerosissime combinazioni possibili di neuroni persi per ciascun $N_n$, ciascun test per ciascun numero di neuroni rimasti è stato ripetuto 10 volte con una diversa scelta casuale di neuroni perduti.

Poiché per ogni ripetizione di sampling è necessario un tempo notevole (circa 4 ore sulla macchina descritta in materiali e metodi) si è scelto di ripetere ciascun sampling solamente 3 volte invece delle 10 usate dal resto degli esperimenti.

Quanto detto genererà per ogni scimmia, per ognuno dei due casi single- e multi- class, per ogni sampling k in K=[1, 3, 5, 7, 9], per ogni neurone casuale perso in più, 30 misure di F-Score. Queste saranno interpretate come attributi descrittori di modelli appartenenti approssimativamente alla stessa distribuzione statistica (la cui variabilità è funzione dei particolari neuroni perduti, della scelta dei trial di addestramento, dell'ordine con cui sono stati fornite le finestrature e dell'inizializzazione dei pesi della rete) e quindi impiegate per stimare media e *Standard Error of the Mean (SEM)* dell'F-Score per ciascun $N_n$.

Nota: lo *Standard Error* di uno stimatore, in questo caso la media, è definito come la sua deviazione standard; la metrica SEM, data dalla formula $\frac{std}{\sqrt{N_{campioni}}}$, descrive quindi dei "margini probabilistici" di oscillazione della media stimata.

### 4.3.2. Risultati dell'esperimento

#### Single Class

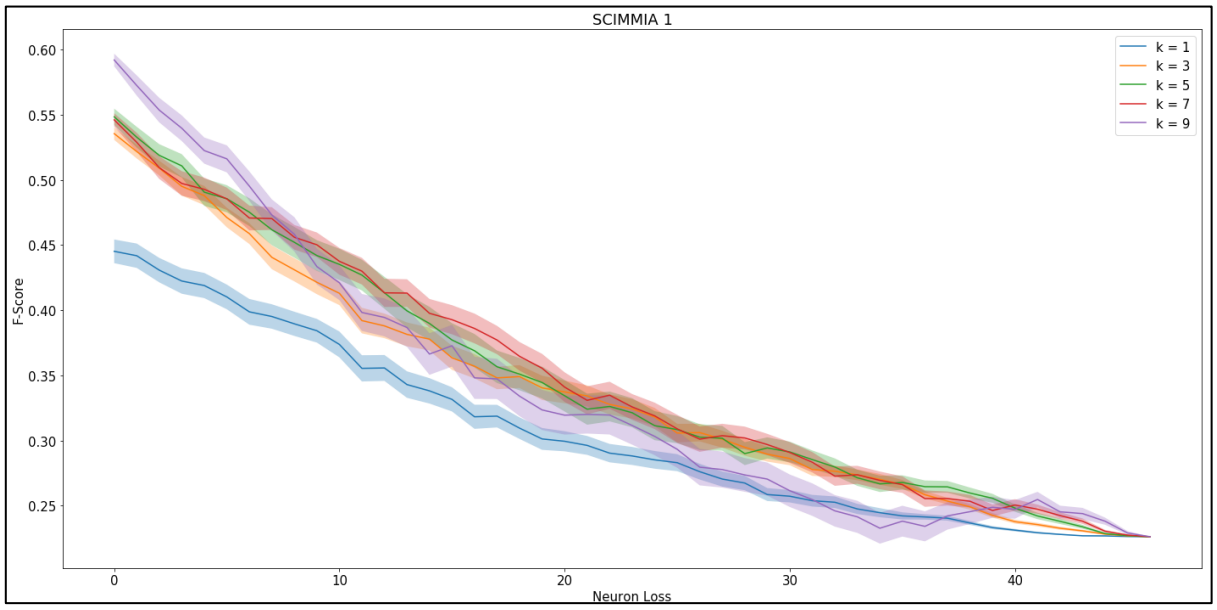


*Figura 51: F-Score (media+-SEM) per ogni sampling k all'aumentare dei neuroni persi. (scimmia 1)*

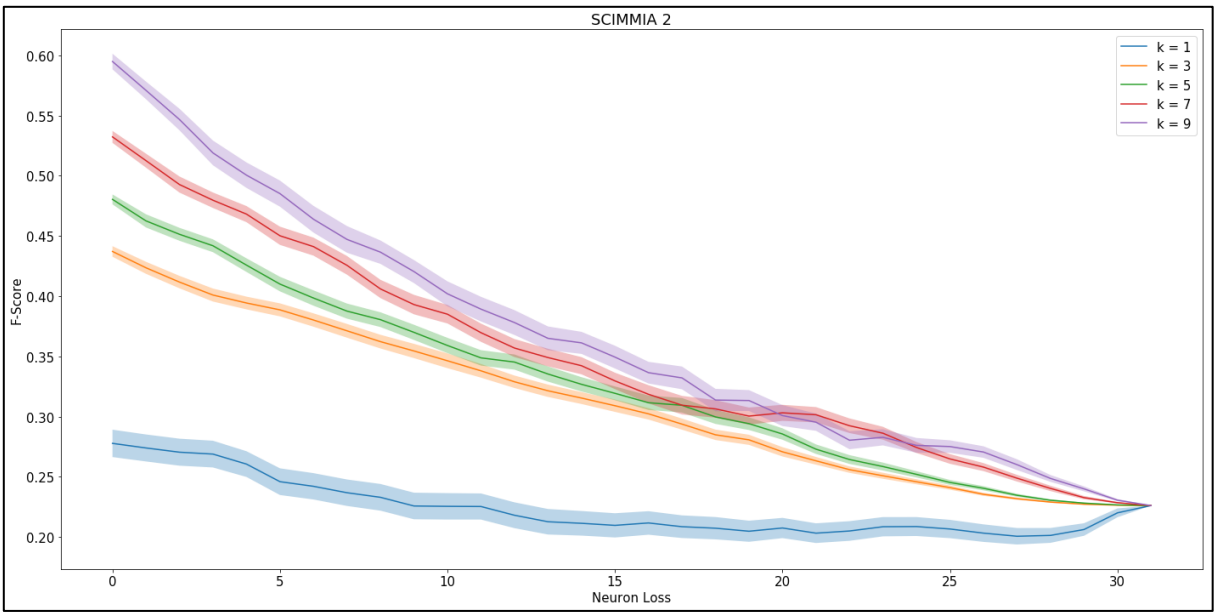


*Figura 52: F-Score (media+-SEM) per ogni sampling k all'aumentare dei neuroni persi. (scimmia 2)*

Le performance degradano in modo chiaro fino ad arrivare ad una media minima compresa tra 0.20 e 0.25, ottenuta dalla decodifica di un solo neurone casuale rimasto. La scimmia 1 sembra mostrare una minor dipendenza dalla quantità di trials campionati rispetto alla scimmia 2, anche se entrambe rilevano un particolare calo di performance nel caso di sampling k=1 – confermando ovviamente i dati dell'esperimento 1. In particolare, il caso k=1 per la scimmia 2 ottiene risultati minimali a prescindere dal numero di neuroni; difficile capire se la causa di ciò è da ricercarsi nell'addestramento con meno neuroni rispetto alla scimmia 1 (32 contro 47) o piuttosto in una particolare difficoltà di decodifica del segnale registrato dalla scimmia.

La forma del decadimento descrive un lieve ramo d'iperbole dall'andamento quasi lineare, mostrando che l'architettura proposta non soffre di tracolli di performance a causa della perdita dei canali, piuttosto questa tende a degradarsi progressivamente. Questo lascia intuire una codifica interna capace di estrarre informazione da tutti i neuroni senza lasciarsi influenzare troppo da ciascuno di essi; se così non fosse ed invece dipendesse unicamente da un piccolo particolare sottogruppo di neuroni, la performance media dovrebbe crollare improvvisamente dopo un certo numero di neuroni perduti – infatti la probabilità di aver eliminato qualche elemento di quel gruppo almeno in qualcuna delle estrazioni ripetute cresce rapidamente lungo il grafico.

Questo andamento medio caratteristico potrebbe consentire in fase di implementazione un facile sistema di controllo della salute dell'impianto, in aggiunta agli altri esistenti: dal decadimento quasi-lineare delle performance dell'algoritmo si potrebbe infatti misurare il grado di fibrosi.

## Multi Class

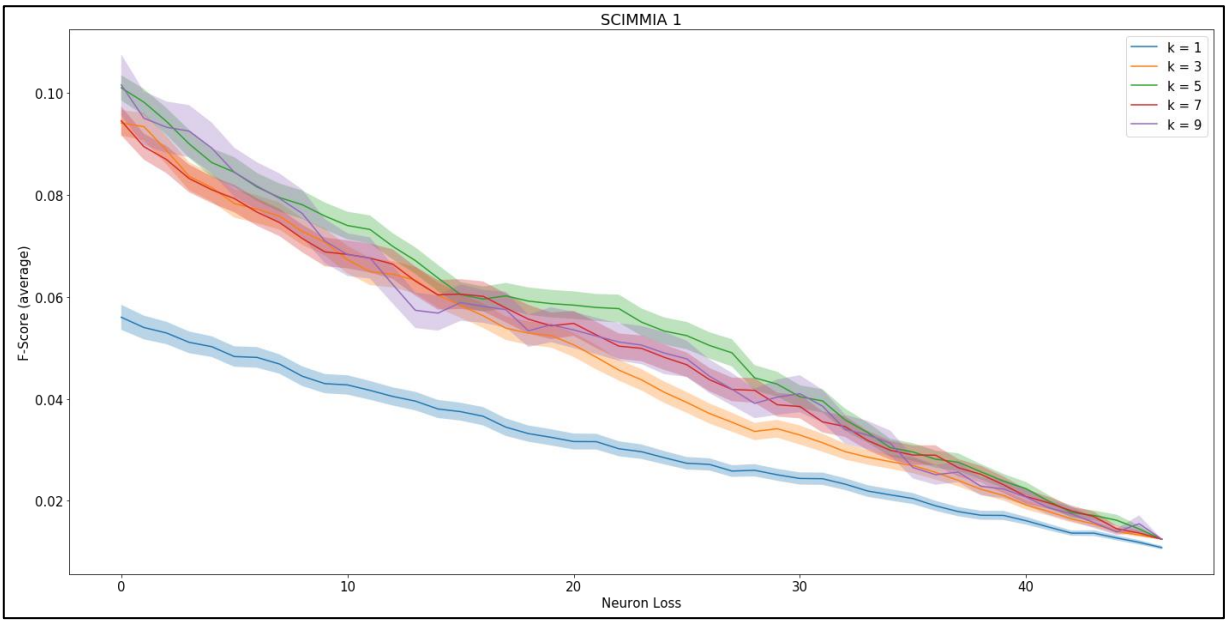


*Figura 53: F-Score medio interclasse (media+-SEM) per ogni sampling k all'aumentare dei neuroni persi. (scimmia 1)*

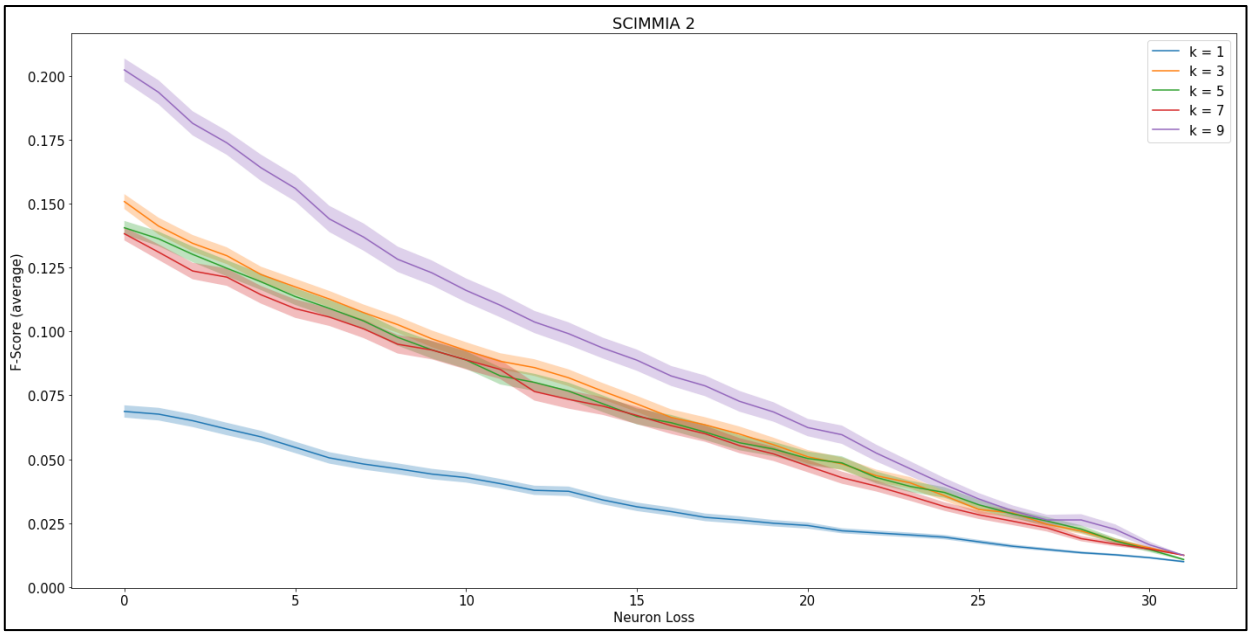


*Figura 54: F-Score medio interclasse (media+-SEM) per ogni sampling k all'aumentare dei neuroni persi. (scimmia 2)*

I risultati multiclasse (mediati tra le classi) sembrano tracciare un andamento complessivamente simile a quello a classe singola: la scimmia 1 mostra poca differenza tra i sampling se non con k=1 e la scimmia 2 mostra più varietà, anche se i sampling 3, 5, 7 sono molto più ravvicinati. E' però importante notare che:

1. I risultati ottenuti sono, come prevedibile visto E2, molto inferiori;
2. Le SEM associate alle medie sono più alte del caso single-class, specialmente nella scimmia 1.

In generale i risultati ottenuti in questo caso sono quindi meno affidabili, anche se sembrano confermare il trend visto nel caso precedente e quindi verificare ulteriormente la relazione stimata tra performance e *neuron loss*.

In conclusione, la *Neuron Loss Analysis* conferma un effettivo apprendimento delle reti attraverso una chiara relazione delle performance con il numero di neuroni acquisibili, che risulta essere un degrado quasi lineare. Questa conferma è avvenuta sia per il caso single-class che, seppur con inferiore attendibilità, per il caso multi-class.

Per verificare ulteriormente i risultati ottenuti bisognerebbe, oltre a revisionare l'approccio multi-classe, compiere in particolare due varianti della simulazione proposta:

1. Una che rimuova spikes casuali dai neuroni non rimossi, con probabilità di rimozione crescente con il numero di neuroni rimossi e quindi del tempo
2. Un'altra che addestri modelli e segua l'evoluzione particolare di ciascuno attraverso l'intera *Neuron Loss Analysis*, in modo da verificare sul singolo modello il decadimento quasi-lineare riscontrato in queste analisi.

# 5. CONCLUSIONI

Citando *verbatim* l'introduzione: lo scopo di questo lavoro è, sommariamente, quello di fornire un solido *benchmark* iniziale per lo sviluppo di algoritmi DL nell'ambito del *decoding real-time* di intenzioni di movimento per il controllo neuroprotesico.

Nel corso della dissertazione tale dichiarazione è stata chiarificata e sviluppata:

- Il "*benchmark*" è stato implementato attraverso una catena di processo capace di portare dal dato iniziale non formattato (*spikes* sortati) fino ad algoritmi addestrati e valutati in performance e robustezza su dati mai visti, a prescindere dal contesto del dato grezzo e dalle caratteristiche dell'algoritmo;
- Gli "algoritmi DL" usati nel lavoro sono stati specificatamente tre esempi notevoli di *Deep Neural Networks* – una *Feed-forward Dense Neural Network*, una *Convolutional NN*, una *Recurrent NN* (di tipo *Gated Recurrent Units*) – dalle cui performance è possibile ipotizzare le caratteristiche ideali di modelli futuri;
- Il *"decoding real-time"* è stato simulato attraverso la tecnica di *sliding-windowing*;
- Le "intenzioni di movimento" sono state associate alle caratteristiche espresse dalle *Single Unit Recordings* nei 500 ms precedenti l'inizio dei movimenti *reach-to-grasp*.

Gli esperimenti hanno evidenziato l'effettivo funzionamento dell'intero approccio, dimostrando la bontà della catena di pre-processing adottata e l'applicabilità delle *DNN* e del paradigma *Deep Learning* nelle sue forme più basilari al problema di decodifica neurale in oggetto.

Le performance non ottimali soprattutto nel caso multi-classe evidenziano tuttavia la necessità di proseguire la linea di ricerca iniziata da questo documento, ad esempio seguendo le strade elencate di seguito:

- Implementare architetture più profonde e complesse che integrino la capacità di *pattern-recognition* delle *CNN* e la memoria delle *RNN*, prendendo spunto dai modelli per *multi-channel time series* impiegati in altri ambiti (ad esempio la *"LSTM Fully Convolutional Networks for Time Series Classification"* di Karim et al. [65]);
- Impiegare tecniche di *data augmentation*, ad esempio miscelando i *timestamps* appartenenti alla stessa classe come fatto da Filippini in [17] o aggiungendo un appropriato *noise* random ai segnali (gaussiano discretizzato sul binning o di Poisson sugli spikes) per presentare più volte "versioni distorte" dello stesso trial e costringere l'algoritmo a differenziare maggiormente segnale e rumore;

- Addestrare le reti con metodi più complessi, ad esempio impiegando *cyclical learning rates* per velocizzare l'apprendimento e migliorarne la generalizzazione [66] o, nel caso di reti molto profonde, utilizzando l'approccio *Stochastic Depth* per addestrarle rapidamente eseguendo un *dropout* su interi strati della rete [67];
- Estendere l'approccio *ensemble* esplorato in E2 aggregando non solo reti costruite per compiti complementari, ma anche reti adibite allo stesso compito ma addestrate su *splits* diversi del *training set* o persino reti che hanno visto gli stessi dati ma inizializzate con pesi diversi ([68], ma l'*ensemble learning* è un argomento rintracciabile in qualsiasi libro intorno al *Machine Learning*)

Bisogna inoltre notare che per ottenere una vera decodifica real-time automatizzata del segnale neurale occorrerebbe impiegare come input dei modelli gli effettivi segnali grezzi rilevati dagli elettrodi: la procedura di spike-sorting, come detto nel capitolo dedicato alle *Single Unit Recordings* in background, è spesso effettuata manualmente (come nel *dataset* impiegato in questo studio) ed impedisce l'implementazione di una procedura automatica. Le riflessioni contenute nei lavori che approcciano il problema di scavalcare lo *spike-sorting*, ad esempio il lavoro di Valérie [20], forniscono tuttavia indicazioni importanti:

- I segnali degli elettrodi vengono modellati come combinazioni quasi-lineari dell'attività dei neuroni, con parametri pressoché costanti nel tempo nell'ipotesi di corretto fissaggio dell'impianto;
- Impianti multi-elettrodo con sufficiente densità di microelettrodi consentirebbero a più ricettori di acquisire gli stessi neuroni con diversi parametri nelle combinazioni quasi-lineari associate, consentendone quindi in linea di principio una dissociazione automatica con tecniche come la già citata PCA e la Independent Component Analysis (ICA) [69]

Poiché PCA ed ICA sono in effetti implementabili con reti neurali [70] queste considerazioni rendono immaginabile la costruzione di *DNNs* capaci di apprendere l'associazione tra segnali degli elettrodi e le attività del soggetto acquisito senza alcuna procedura manuale nella *pipeline*, nell'ipotesi che tali modelli vengano impiegati sopra sistemi effettivamente multi-elettrodo.

Questo lavoro esplorativo si pone, in fin dei conti, come apripista nel proporre metodi *Deep Learning* per la costruzione di algoritmi facilmente sviluppabili, implementabili su circuito in maniera energeticamente efficiente, totalmente automatici e quasi-*real-time* per il controllo di neuroprotesi. In tutta la letteratura analizzata è stato trovato un solo altro lavoro che implementi moderne reti neurali alla decodifica di *spiking* (*"Machine learning for neural decoding"* di Glaser et al. [71]), che affronta tuttavia un problema di regressione rispetto a quello di classificazione oggetto della tesi;

l’autore spera quindi di aver integrato i contributi e i risultati di tale studio con ulteriori prove della potenzialità delle *DNN* e del *Deep Learning* in generale, costruendo insieme ad esso un piccolo *tutorial* iniziale che possa facilitare a futuri ricercatori il difficile, multidisciplinare, affascinante compito di decodificare reti neurali biologiche con reti neurali sintetiche.

L’autore si augura, infine, che il lavoro sviluppato possa configurarsi come minuscolo ma funzionale tassello nello sforzo internazionale che tenta di riportare all’abilità invalidi e mutilati attraverso biorobotica e neuroprotesica, ambiziosi eredi della cibernetica di Wiener rappresentati in Italia da centri di fama mondiale come la Scuola Superiore Sant’Anna e il Centro Protesi (Inail) Vigorso di Budrio - centri ai quali potrebbe aggiungersi in futuro l’Ateneo di Bologna grazie all’attività del gruppo di lavoro intorno alla Prof.ssa Fattori, sotto il quale è stata portata avanti questa tesi.

Sono di metà febbraio le immagini della ragazza invalida con le braccia protesiche dell’eroina giapponese Alita (*Figura 55*), costruite da OpenBionics (openbionics.com) – oggetti belli e maneggevoli, ma sconnessi dalla mente. Grazie all’impegno cibernetico internazionale, forse da adulta potrà muovere oggetti simili come fossero da sempre le sue braccia.


*Figura 55: Tilly Lockey e le sue* Hero Arms *della Open Bionics (https://openbionics.com/bionic-heroes/tilly/)*

# APPENDICE – F-Scores per ciascuna classe

Si mostrano di seguito i *violin-plots* delle distribuzioni di F-Score per ciascuna classe-oggetto. Come nei capitoli precedenti, ogni grafico rappresenta una scimmia.

## C1 | Palla

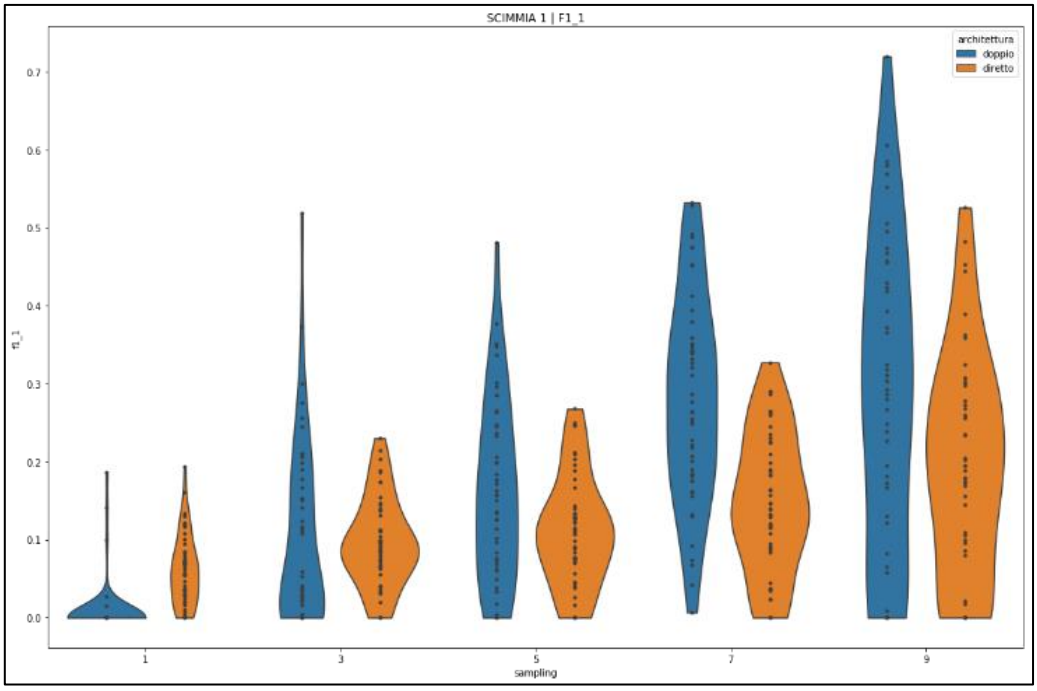


*Figura 56: F-SCORES (palla) ottenuti dai modelli "diretto" e "doppio" (composito) per ogni sampling k. (scimmia 1)*

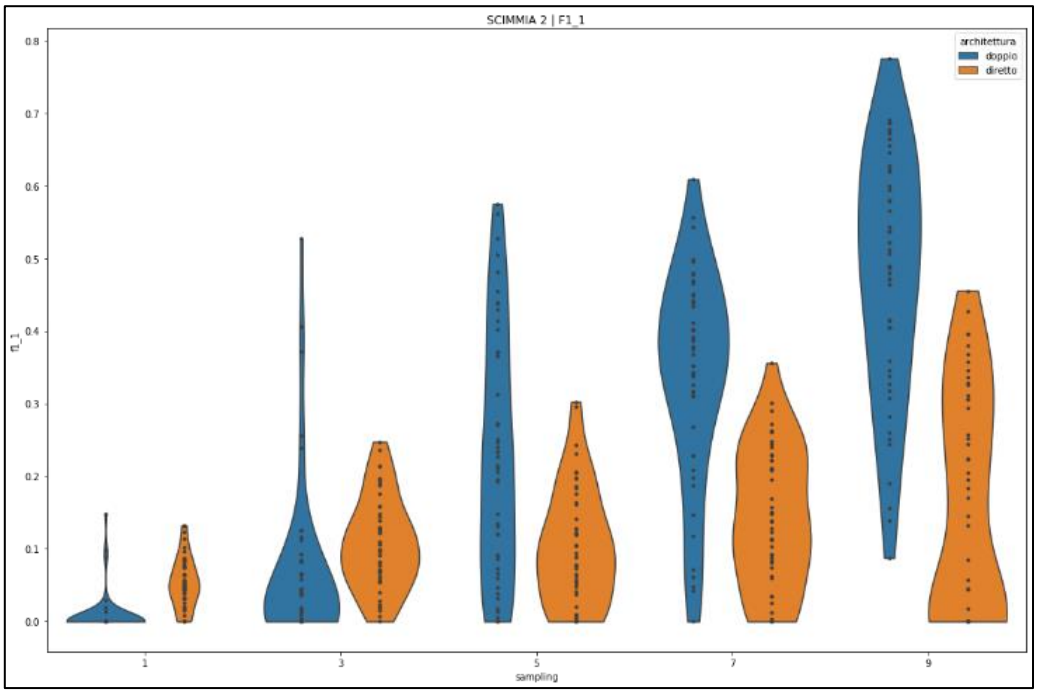


*Figura 57: F-SCORES (palla) ottenuti dai modelli "diretto" e "doppio" (composito) per ogni sampling k. (scimmia 2)*

## C2 | Maniglia

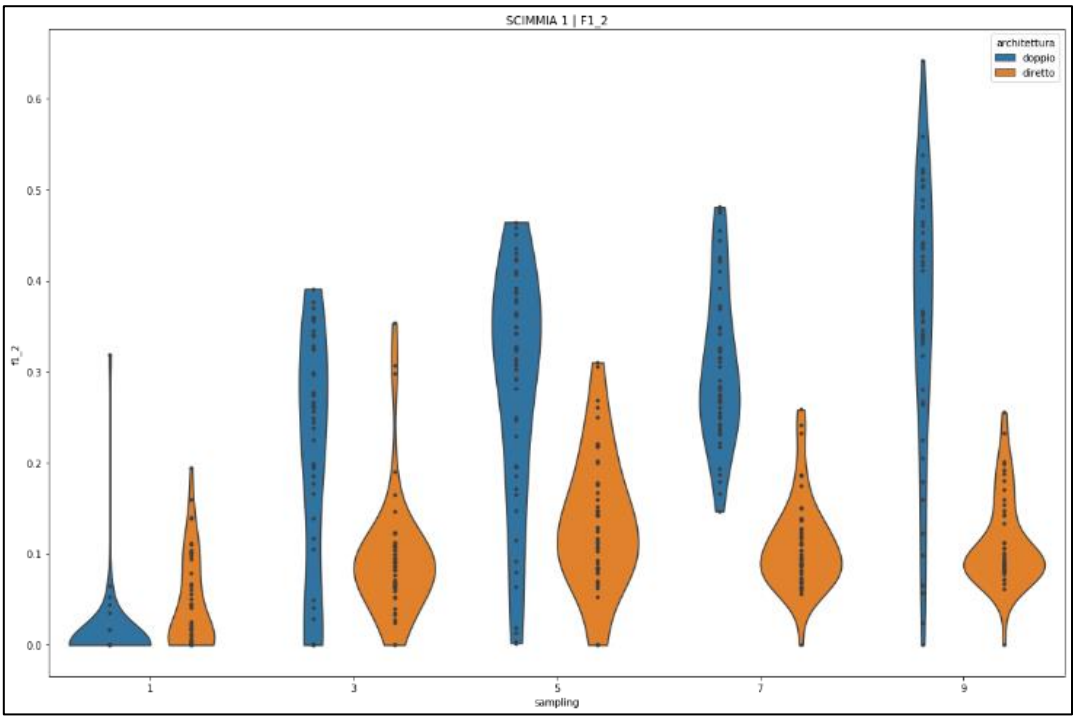


*Figura 58: F-SCORES (maniglia) ottenuti dai modelli "diretto" e "doppio" (composito) per ogni sampling k. (scimmia 1)*

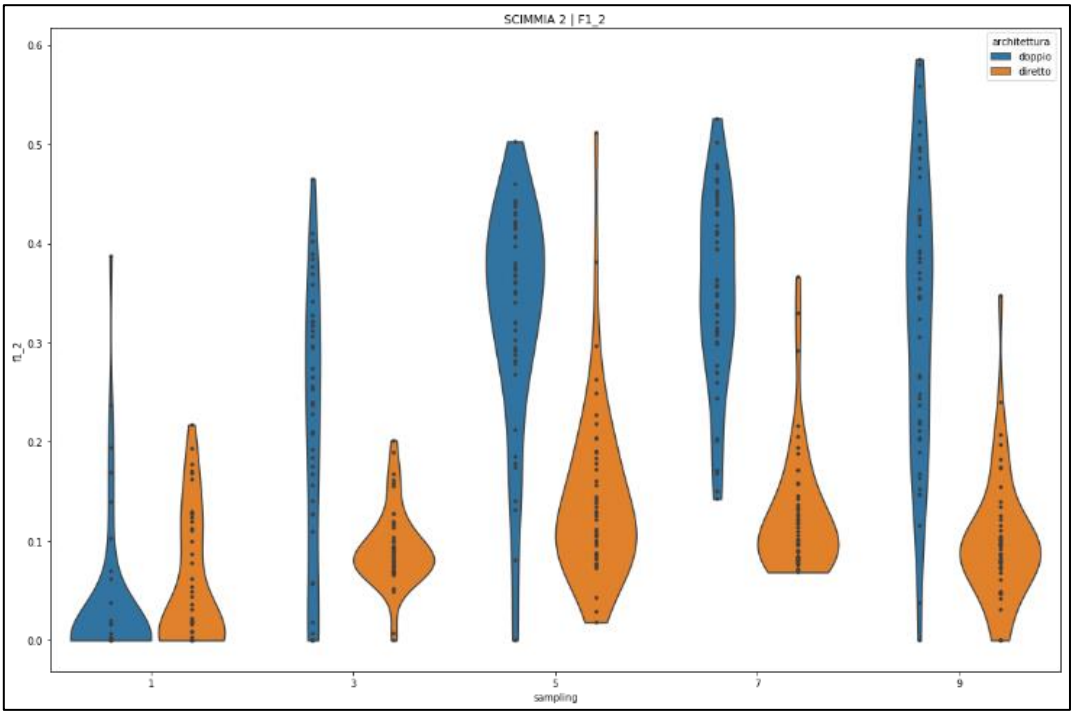


*Figura 59: F-SCORES (maniglia) ottenuti dai modelli "diretto" e "doppio" (composito) per ogni sampling k. (scimmia 2)*

## C3 | Anello

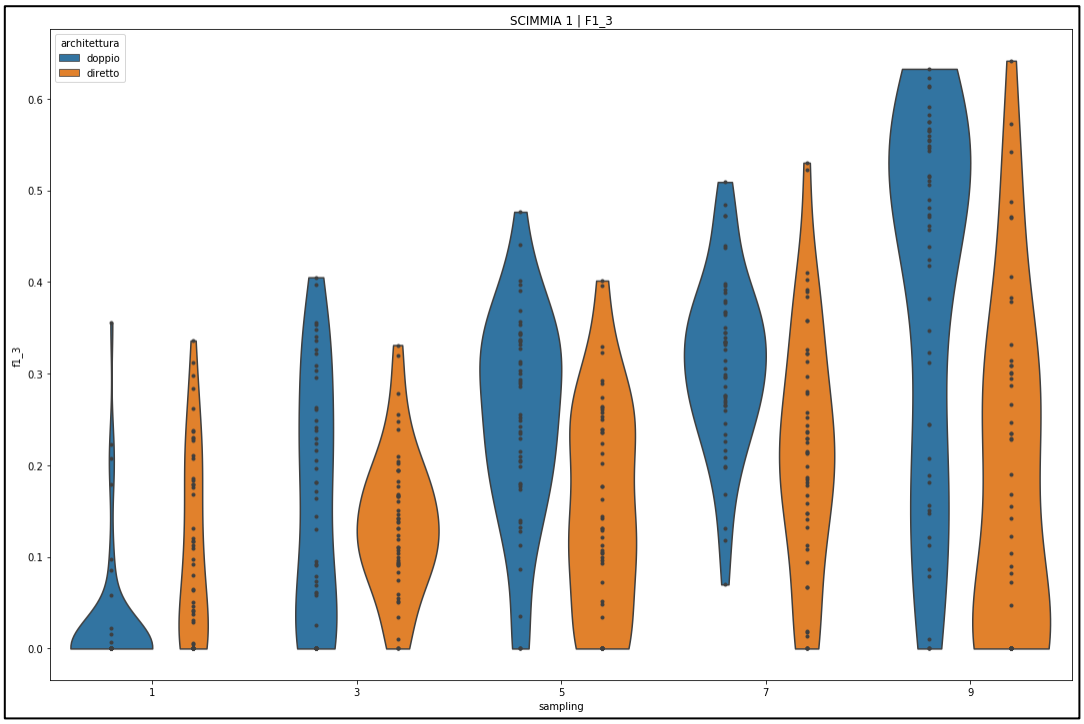


*Figura 60: F-SCORES (anello) ottenuti dai modelli "diretto" e "doppio" (composito) per ogni sampling k. (scimmia 1)*

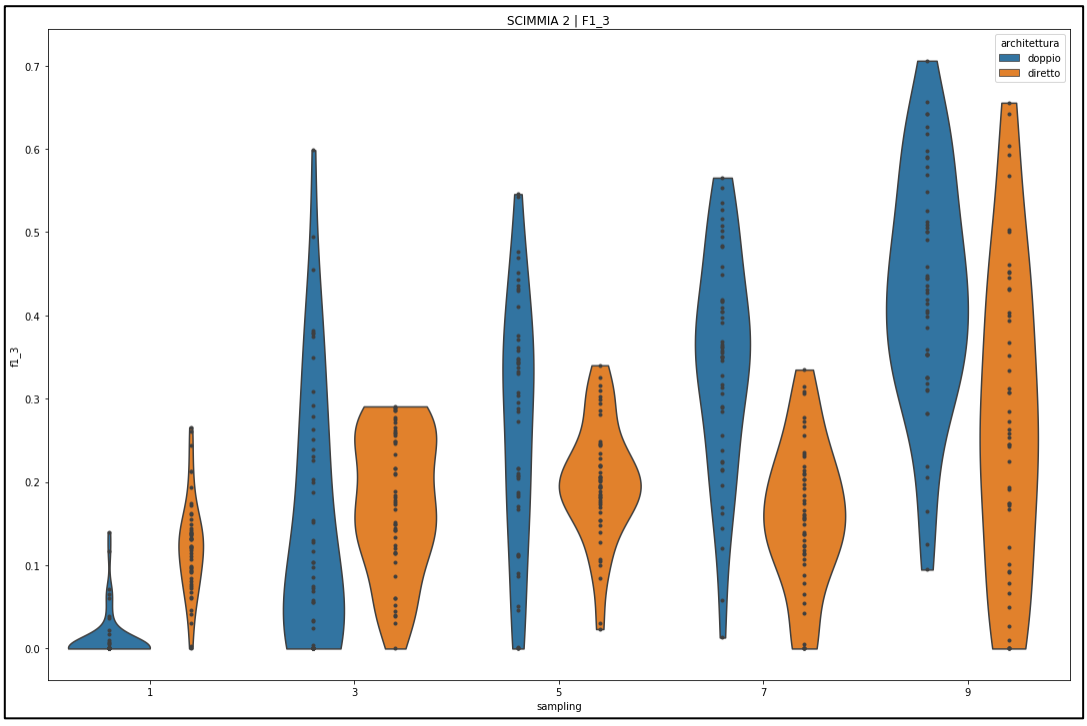


*Figura 61: F-SCORES (anello) ottenuti dai modelli "diretto" e "doppio" (composito) per ogni sampling k. (scimmia 2)*

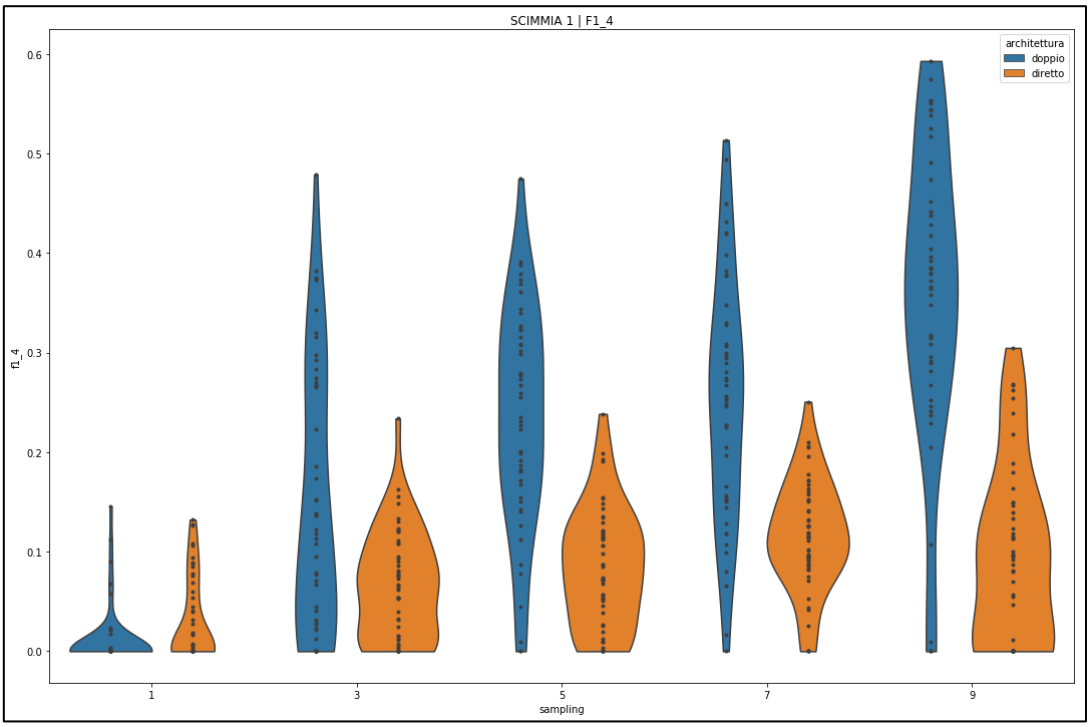


*Figura 62: F-SCORES (piastra) ottenuti dai modelli "diretto" e "doppio" (composito) per ogni sampling k. (scimmia 1)*

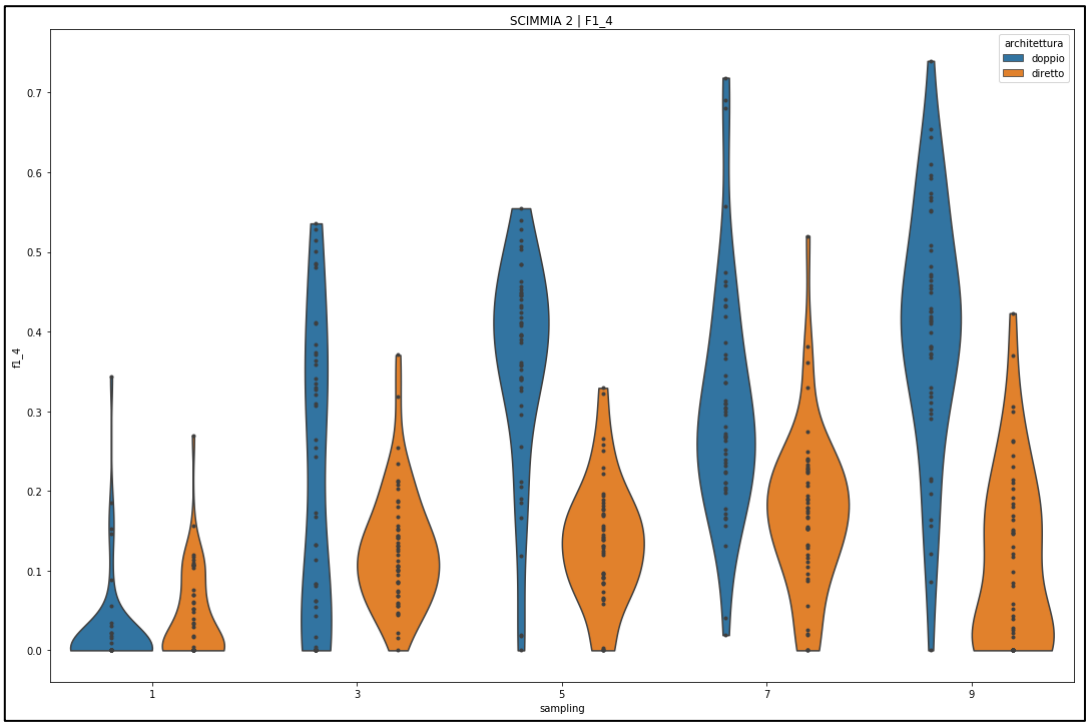


*Figura 63: F-SCORES (piastra) ottenuti dai modelli "diretto" e "doppio" (composito) per ogni sampling k. (scimmia 2)*

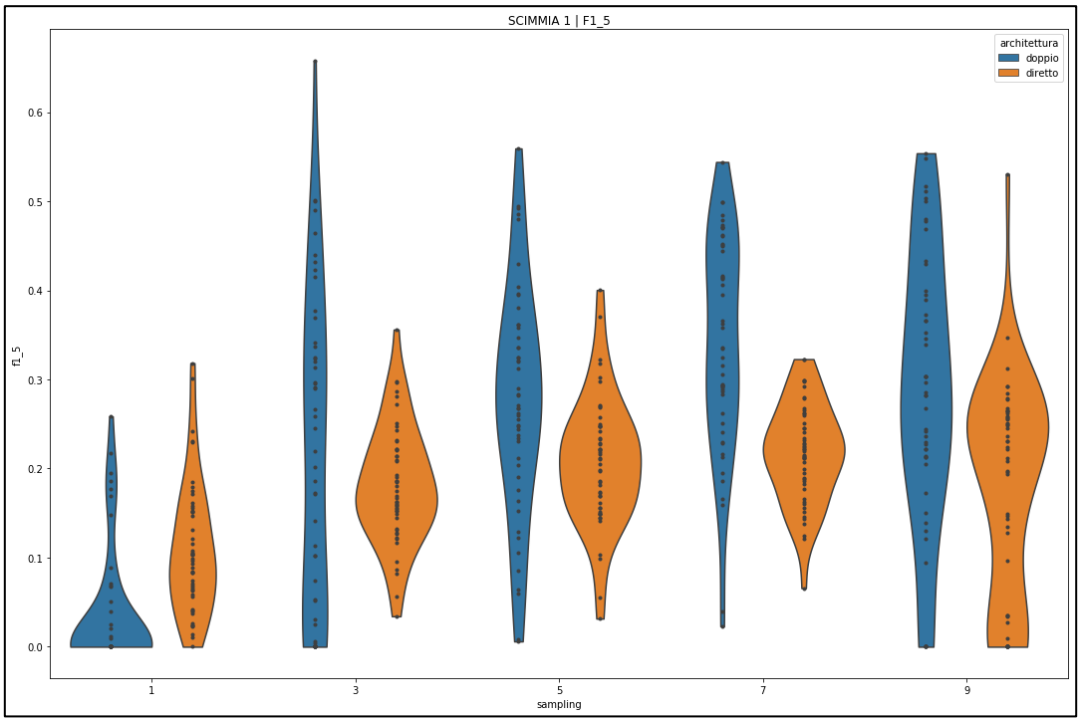


*Figura 64: F-SCORES (fessura) ottenuti dai modelli "diretto" e "doppio" (composito) per ogni sampling k. (scimmia 1)*

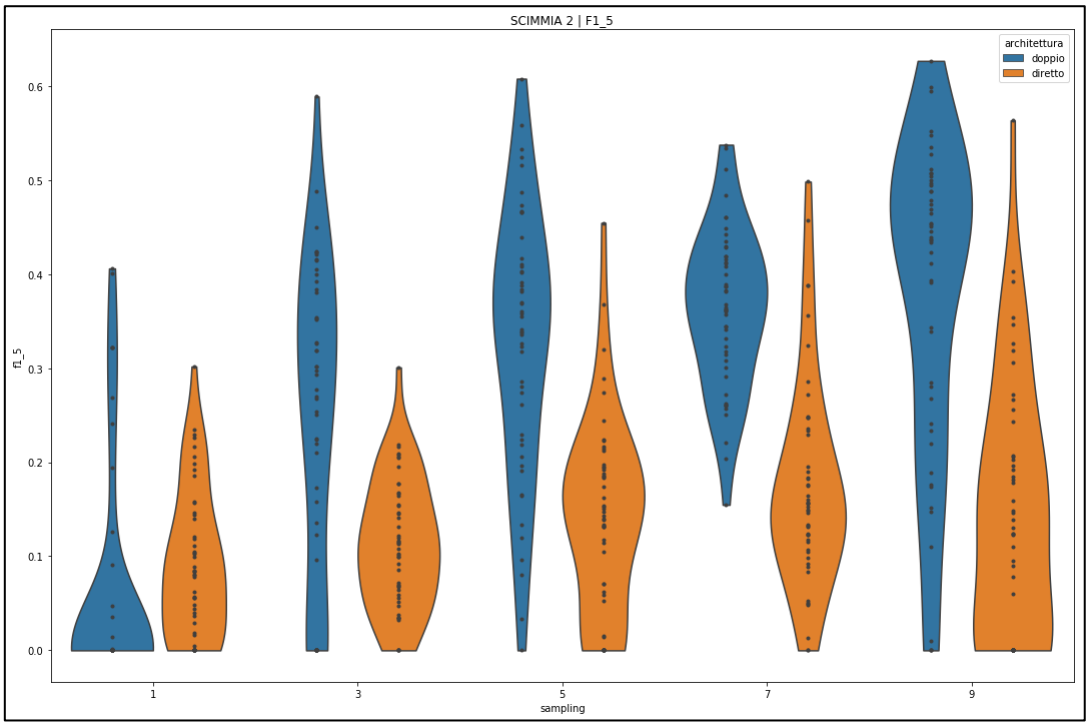


*Figura 65: F-SCORES (fessura) ottenuti dai modelli "diretto" e "doppio" (composito) per ogni sampling k. (scimmia 2)*

# BIBLIOGRAFIA


[1] J. J. Vidal, «Toward Direct Brain-Computer Communication», *Annu. Rev. Biophys. Bioeng.*, vol. 2, n. 1, pagg. 157–180, 1973.

[2] V. Guy, M.-H. Soriani, M. Bruno, T. Papadopoulo, C. Desnuelle, e M. Clerc, «Brain computer interface with the P300 speller: Usability for disabled people with amyotrophic lateral sclerosis», *Ann. Phys. Rehabil. Med.*, vol. 61, n. 1, pagg. 5–11, gen. 2018.

[3] C. Escolano, J. M. Antelis, e J. Minguez, «A telepresence mobile robot controlled with a noninvasive brain-computer interface», *IEEE Trans. Syst. Man Cybern. Part B Cybern.*, vol. 42, n. 3, pagg. 793–804, 2012.

[4] R. Chavarriaga, A. Sobolewski, e J. del R. Millán, «Errare machinale est: The use of error-related potentials in brain-machine interfaces», *Front. Neurosci.*, vol. 8, n. 8 JUL, pagg. 1–13, 2014.

[5] D. Achanccaray, K. Acuna, E. Carranza, e J. Andreu-Perez, «A virtual reality and brain computer interface system for upper limb rehabilitation of post stroke patients», *2017 IEEE Int. Conf. Fuzzy Syst. FUZZ-IEEE*, pagg. 1–5, 2017.

[6] K. M. Szostak, L. Grand, e T. G. Constandinou, «Neural Interfaces for Intracortical Recording: Requirements, Fabrication Methods, and Characteristics», *Front. Neurosci.*, vol. 11, dic. 2017.

[7] A. P. Georgopoulos, A. B. Schwartz, e R. E. Kettner, «Neuronal population coding of movement direction», *Science*, vol. 233, n. 4771, pagg. 1416–1419, set. 1986.

[8] L. R. Hochberg *et al.*, «Neuronal ensemble control of prosthetic devices by a human with tetraplegia», *Nature*, vol. 442, n. 7099, pagg. 164–171, lug. 2006.

[9] S. H. Scott, P. L. Gribble, K. M. Graham, e D. W. Cabel, «Dissociation between hand motion and population vectors from neural activity in motor cortex», *Nature*, vol. 413, n. 6852, pagg. 161–165, set. 2001.

[10] S. N. Flesher *et al.*, «Intracortical microstimulation of human somatosensory cortex», *Sci. Transl. Med.*, vol. 8, n. 361, pag. 361ra141, 19 2016.

[11] P. P. Battaglini, A. Muzur, C. Galletti, M. Skrap, A. Brovelli, e P. Fattori, «Effects of lesions to area V6A in monkeys», *Exp. Brain Res.*, vol. 144, n. 3, pagg. 419–422, giu. 2002.

[12] R. Breveglieri, D. F. Kutz, P. Fattori, M. Gamberini, e C. Galletti, «Somatosensory cells in the parieto-occipital area V6A of the macaque»:, *NeuroReport*, vol. 13, n. 16, pagg. 2113–2116, nov. 2002.

[13] P. Fattori, M. Gamberini, D. F. Kutz, e C. Galletti, «"Arm-reaching" neurons in the parietal area V6A of the macaque monkey», *Eur. J. Neurosci.*, vol. 13, n. 12, pagg. 2309–2313, giu. 2001.

[14] P. Fattori, R. Breveglieri, N. Marzocchi, D. Filippini, A. Bosco, e C. Galletti, «Hand orientation during reach-to-grasp movements modulates neuronal activity in the medial posterior parietal area V6A», *J. Neurosci. Off. J. Soc. Neurosci.*, vol. 29, n. 6, pagg. 1928–1936, feb. 2009.

[15] P. Fattori, R. Breveglieri, V. Raos, A. Bosco, e C. Galletti, «Vision for Action in the Macaque Medial Posterior Parietal Cortex», *Journal of Neuroscience*, vol. 32, n. 9, pagg. 3221–3234, feb. 2012.

[16] A. Bosco, R. Breveglieri, D. Reser, C. Galletti, e P. Fattori, «Multiple Representation of Reaching Space in the Medial Posterior Parietal Area V6A», *Cereb. Cortex*, vol. 25, n. 6, pagg. 1654–1667, giu. 2015.

[17] M. Filippini, R. Breveglieri, M. A. Akhras, A. Bosco, E. Chinellato, e P. Fattori, «Decoding Information for Grasping from the Macaque Dorsomedial Visual Stream.», *J. Neurosci. Off. J. Soc. Neurosci.*, vol. 37, n. 16, pagg. 4311–4322, apr. 2017.

[18] A. Biswas e A. P. Chandrakasan, «Conv-RAM: An energy-efficient SRAM with embedded convolution computation for low-power CNN-based machine learning applications», in *2018 IEEE International Solid - State Circuits Conference - (ISSCC)*, 2018, pagg. 488–490.
[19] V. S. Polikov, P. A. Tresco, e W. M. Reichert, «Response of brain tissue to chronically implanted neural electrodes», *J. Neurosci. Methods*, vol. 148, n. 1, pagg. 1–18, ott. 2005.
[20] V. Valérie, «Spike Train Decoding Without Spike Sorting», *Neural Comput.*, vol. 20, n. 4, pag. 923, 2008.
[21] G. W. Fraser, S. M. Chase, A. Whitford, e A. B. Schwartz, «Control of a brain–computer interface without spike sorting», *Journal of Neural Engineering*, vol. 6, n. 5, pag. 055004, ott. 2009.
[22] M. Kawato, «Internal models for motor control and trajectory planning», *Curr. Opin. Neurobiol.*, vol. 9, n. 6, pagg. 718–727, dic. 1999.
[23] D. Ha e J. ¨ Urgen Schmidhuber, «World Models».
[24] H. Cui, «Forward Prediction in the Posterior Parietal Cortex and Dynamic Brain-Machine Interface», *Frontiers in Integrative Neuroscience*, vol. 10, ott. 2016.
[25] F.-A. Savoie, F. Thénault, K. Whittingstall, e P.-M. Bernier, «Visuomotor Prediction Errors Modulate EEG Activity Over Parietal Cortex», *Sci. Rep.*, vol. 8, n. 1, dic. 2018.
[26] G. Rizzolatti e M. Gentilucci, «Motor and Visual-Motor Functions of the Premotor Cortex», pag. 16.
[27] M. J. Ahanshahi e M. Hallett, *The Bereitschaftspotential Movement -Related Cortical Potentials*. 2003.
[28] H. Shibasaki e M. Hallett, «What is the Bereitschaftspotential?», *Clin. Neurophysiol.*, vol. 117, n. 11, pagg. 2341–2356, nov. 2006.
[29] M. Desmurget, K. T. Reilly, N. Richard, A. Szathmari, C. Mottolese, e A. Sirigu, «Movement Intention After Parietal Cortex Stimulation in Humans», *Science*, vol. 324, n. 5928, pagg. 811–813, mag. 2009.
[30] L. Deng e D. Yu, «Deep Learning: Methods and Applications», *Found. Trends® Signal Process.*, vol. 7, n. 3–4, pagg. 197–387, 2014.
[31] W. McCulloch e W. Pitts, «A Logical Calculus of Ideas Immanent in Nervous Activity».
[32] Hodgkin Alan Lloyd, Huxley Andrew Fielding, e Eccles John Carew, «Propagation of electrical signals along giant nerve fibres», *Proc. R. Soc. Lond. Ser. B - Biol. Sci.*, vol. 140, n. 899, pagg. 177–183, ott. 1952.
[33] A. M. Turing, «I.—COMPUTING MACHINERY AND INTELLIGENCE», *Mind*, vol. LIX, n. 236, pagg. 433–460, ott. 1950.
[34] C. Shannon e W. Weaver, «The Mathematical Theory of Communication», pag. 131.
[35] N. Wiener, *Cybernetics Or Control and Communication in the Animal and the Machine*. MIT Press, 1965.
[36] F. Rosenblatt, *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Cornell Aeronautical Laboratory, 1957.
[37] F. Rosenblatt, *The Perceptron: A Probabilistic Model for Information Storage and Organization In The Brain*. 1958.
[38] D. E. Rumelhart, G. E. Hintont, e R. J. Williams, «Learning representations by back-propagating errors», pag. 4, 1986.
[39] H. J. Kelley, «Gradient Theory of Optimal Flight Paths», *ARS J.*, vol. 30, n. 10, pagg. 947–954, 1960.
[40] Stork, «Is backpropagation biologically plausible?», in *International 1989 Joint Conference on Neural Networks*, 1989, pagg. 241–246 vol.2.
[41] Y. Bengio, D.-H. Lee, J. Bornschein, T. Mesnard, e Z. Lin, «Towards Biologically Plausible Deep Learning», *ArXiv150204156 Cs*, feb. 2015.
[42] G. Cybenkot, «Approximation by superpositions of a sigmoidal function», pag. 12.

[43] K. Hornik, «Approximation capabilities of multilayer feedforward networks», *Neural Netw.*, vol. 4, n. 2, pagg. 251–257, gen. 1991.
[44] A. G. IVAKHNENKO, «The Group Method of Data of Handling ; A rival of the method of stochastic approximation», *Sov. Autom. Control*, vol. 13, pagg. 43–55, 1968.
[45] D. H. Hubel e T. N. Wiesel, «Receptive fields of single neurones in the cat's striate cortex», *J. Physiol.*, vol. 148, n. 3, pagg. 574–591, ott. 1959.
[46] J. J. Hopfield, «Neural networks and physical systems with emergent collective computational abilities», *Proc. Natl. Acad. Sci.*, vol. 79, n. 8, pagg. 2554–2558, apr. 1982.
[47] J. L. Elman, «Finding Structure in Time», *Cogn. Sci.*, vol. 14, n. 2, pagg. 179–211, 1990.
[48] J. Schmidhuber, «DISCOVERING PREDICTABLE CLASSIFICATIONS Technical Report CU-CS-626-92», pag. 9.
[49] S. Hochreiter e J. Schmidhuber, *Long Short-Term Memory*. .
[50] Y. LeCun, L. Bottou, Y. Bengio, e P. Ha, «Gradient-Based Learning Applied to Document Recognition», pag. 46, 1998.
[51] A. Krizhevsky, I. Sutskever, e G. E. Hinton, «ImageNet Classification with Deep Convolutional Neural Networks», in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, e K. Q. Weinberger, A c. di Curran Associates, Inc., 2012, pagg. 1097–1105.
[53] M. Filippini, «Messa a punto di un sistema di acquisizione e di analisi di scariche bioelettriche di cellule parietali di macaco.», Alma Mater Studiorum, Università di Bologna.
[54] D. P. Kingma e J. Ba, «Adam: A Method for Stochastic Optimization», *ArXiv14126980 Cs*, dic. 2014.
[55] M. D. Zeiler *et al.*, «On rectified linear units for speech processing», in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pagg. 3517–3521.
[56] K. Hara, D. Saito, e H. Shouno, «Analysis of function of rectified linear unit used in deep learning», in *2015 International Joint Conference on Neural Networks (IJCNN)*, 2015, pagg. 1–8.
[57] M. Pumperla, *Keras + Hyperopt: A very simple wrapper for convenient hyperparameter optimization: maxpumperla/hyperas*. 2019.
[58] J. Bergstra, D. Yamins, e D. D. Cox, «Hyperopt: A Python Library for Optimizing the Hyperparameters of Machine Learning Algorithms», pag. 8, 2013.
[59] D. Li, «Visualization of Deep Convolutional Neural Networks», pag. 66.
[60] S. Ioffe e C. Szegedy, «Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift», *ArXiv150203167 Cs*, feb. 2015.
[61] K. Cho *et al.*, «Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation», *ArXiv14061078 Cs Stat*, giu. 2014.
[62] J. Chung, C. Gulcehre, K. Cho, e Y. Bengio, «Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling», *ArXiv14123555 Cs*, dic. 2014.
[63] L. Rokach, «Ensemble-based classifiers», *Artif. Intell. Rev.*, vol. 33, n. 1, pagg. 1–39, feb. 2010.
[64] D. H. Szarowski *et al.*, «Brain responses to micro-machined silicon devices», *Brain Res.*, vol. 983, n. 1–2, pagg. 23–35, set. 2003.
[65] F. Karim, S. Majumdar, H. Darabi, e S. Chen, «LSTM Fully Convolutional Networks for Time Series Classification», *IEEE Access*, vol. 6, pagg. 1662–1669, 2018.
[66] L. N. Smith, «Cyclical Learning Rates for Training Neural Networks», in *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, Santa Rosa, CA, USA, 2017, pagg. 464–472.
[67] G. Huang, Y. Sun, Z. Liu, D. Sedra, e K. Weinberger, «Deep Networks with Stochastic Depth», arXiv:1603.09382 [cs], mar. 2016.

[68] M. P. Perrone, «When Networks Disagree: Ensemble Methods for Hybrid Neural Networks», pag. 17.
[69] A. Hyvärinen e E. Oja, «Independent component analysis: algorithms and applications», *Neural Netw.*, vol. 13, n. 4–5, pagg. 411–430, giu. 2000.
[70] R. Mutihac e M. M. Van Hulle, «Neural network implementations of independent component analysis», in *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, Martigny, Switzerland, 2002, pagg. 505–514.
[71] J. I. Glaser, R. H. Chowdhury, M. G. Perich, e L. E. Miller, «Machine learning for neural decoding», pag. 24.