Indice

1

INTRODUZIONE

Una rete neurale è un "modello matematico e informatico per l'elaborazione delle informazioni, denominato anche rete neurale artificiale, ispirato alla fisiologia e al funzionamento del cervello umano e del sistema nervoso in genere"[1]. Le reti neurali del cervello umano sono in grado di comprendere l'ambiente e, in base a come esso si presenta, trasmettono impulsi nervosi che ci consentono di affrontare le esigenze quotidiane.

Similarmente, le reti neurali artificiali sono state progettate con l'idea di un apprendimento capace di emulare quello che è alla base dell'intelligenza umana. Esse sono formate da neuroni artificiali comunicanti, che fungono da unità di processo e simulano i collegamenti sinaptici dei neuroni del cervello. Proprio come i neuroni biologici, tali unità elaborano dati in parallelo e contemporaneamente, a differenza dei sistemi in serie presenti nei calcolatori tradizionali.

Il sistema nervoso biologico impara grazie agli esempi nell'ambiente e, in accordo a come essi si presentano, utilizza la conoscenza appresa per adattarsi a situazioni future che mostrano una certa affinità con gli eventi precedenti, ma che risultano comunque diverse. Una rete neurale artificiale utilizza una serie limitata di campioni, i quali vengono utilizzati dal sistema per apprendere la risoluzione di compiti simili agli esempi su cui è stata addestrato. La fase di apprendimento corrisponde, dal punto di vista biologico, alla modifica dei collegamenti sinaptici, dopo la quale, in riferimento ai campioni usati durante l'addestramento, la rete appare in grado di fornire la soluzione col minimo errore.

Le reti neurali offrono quindi uno strumento molto potente per la risoluzione di problemi di classificazione e regressione, quali il riconoscimento di determinati oggetti in un'immagine o la previsione di fenomeni.

In questo lavoro di tesi triennale ci occupiamo di predire le condizioni psicologiche e i comportamenti di vari soggetti durante diversi periodi di tempo, in base a sequenze di misurazioni provenienti da sensori biometrici. Tale approccio sottintende l'impiego di serie temporali, che costituiscono il campione di riferimento. Le reti neurali *feed-forward* non possiedono memoria degli ingressi precedenti e ricevono in input informazioni statiche nel tempo. Questo permette loro, ad esempio, di riconoscere un oggetto in una foto, utilizzandone i pixel come esempi di training. Predire un comportamento, tuttavia, presuppone

[1] Treccani, Enciclopedia della Matematica, 2013.

un'analisi temporale dell'azione, mediante le informazioni raccolte in un determinato lasso di tempo.

Per questo motivo sono state scelte le reti neurali ricorrenti (*Recurrent Neural Networks*, RNN) [1-3]. Esse conservano memoria degli ingressi precedenti, tramite la presenza di cicli tra le unità. In sostanza, una RNN mantiene in memoria lo stato da un'iterazione alla successiva, producendo un output che poi rimanda indietro a se stessa come input.

Poiché puntiamo ad un approccio pratico, orientato ad un feedback immediato dell'utente mediante l'ausilio di dispositivi indossabili, la ricerca di efficienza costituisce il *focus* primario. L'individuazione di un compromesso ideale tra complessità e affidabilità dei risultati assume quindi fondamentale importanza.

In seguito verranno discussi nel dettaglio i modelli utilizzati e la divisione dei dati in sequenze temporali, oltre a proporre un'analisi comparativa tra i quattro tipi di RNN utilizzati (ESN [4-7], SimpleRNN [8-10], LSTM [8,10] e GRU [9,10]) per predire comportamenti e condizioni emotive di molteplici soggetti, sulla base di misurazioni provenienti da cinque dataset.

2

FONDAMENTI DI RETI NEURALI

In questo capitolo sono trattati i fondamenti teorici alla base delle reti neurali. In Sezione 2.1 vengono introdotte le reti neurali artificiali, delineandone architettura e funzionamento generale. Sono inoltre analizzate le principali funzioni di attivazione, le tipologie degli algoritmi di apprendimento e i *task* di classificazione, regressione e *clustering*. In Sezione 2.2 introduciamo le reti neurali ricorrenti, le quali, a differenza delle reti *feed-forward* precedentemente descritte, presentano una memoria interna grazie alla presenza di cicli tra le unità, garantendo quindi un'efficace analisi predittiva su sequenze temporali di dati. Successivamente, all'interno della stessa sezione, esponiamo brevemente la struttura e il funzionamento dei quattro modelli di reti neurali ricorrenti utilizzati, analizzandone proprietà, caratteristiche ed eventuali problemi.

2.1 Reti neurali artificiali

Le reti neurali artificiali (*Artificial Neural Networks*, ANN) [11,12] nascono dal tentativo di replicare le reti neurali biologiche. Esse consistono di un insieme di nodi (o neuroni) comunicanti e organizzati in diversi strati (*layer*). Ogni collegamento tra nodi si caratterizza per la presenza di parametri adattivi **w**, chiamati pesi. Combinando i segnali ricevuti dai nodi dello strato precedente con i relativi pesi e attraverso l'applicazione di una funzione di attivazione φ, ogni nodo fornisce allo strato successivo un valore in uscita: $y(x) = \varphi(\sum \mathbf{w}_i x_i)$.

Una rete neurale è composta da tre tipi di strati:

- di input: contiene i dati iniziali della rete.
- *hidden* (nascosti): tipo di layer intermedio, in cui viene appresa la rappresentazione dell'informazione di input più adatta a facilitare la risoluzione del compito predittivo.
- di output: produce il risultato dato l'input ricevuto.

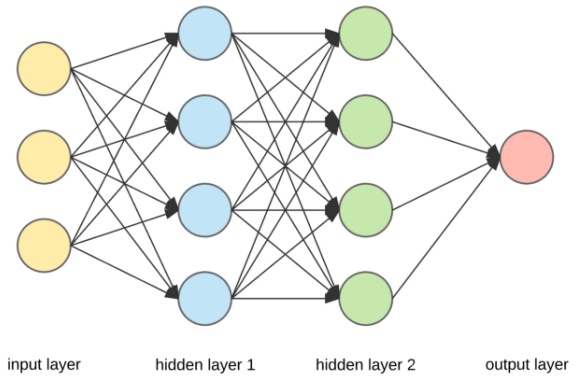
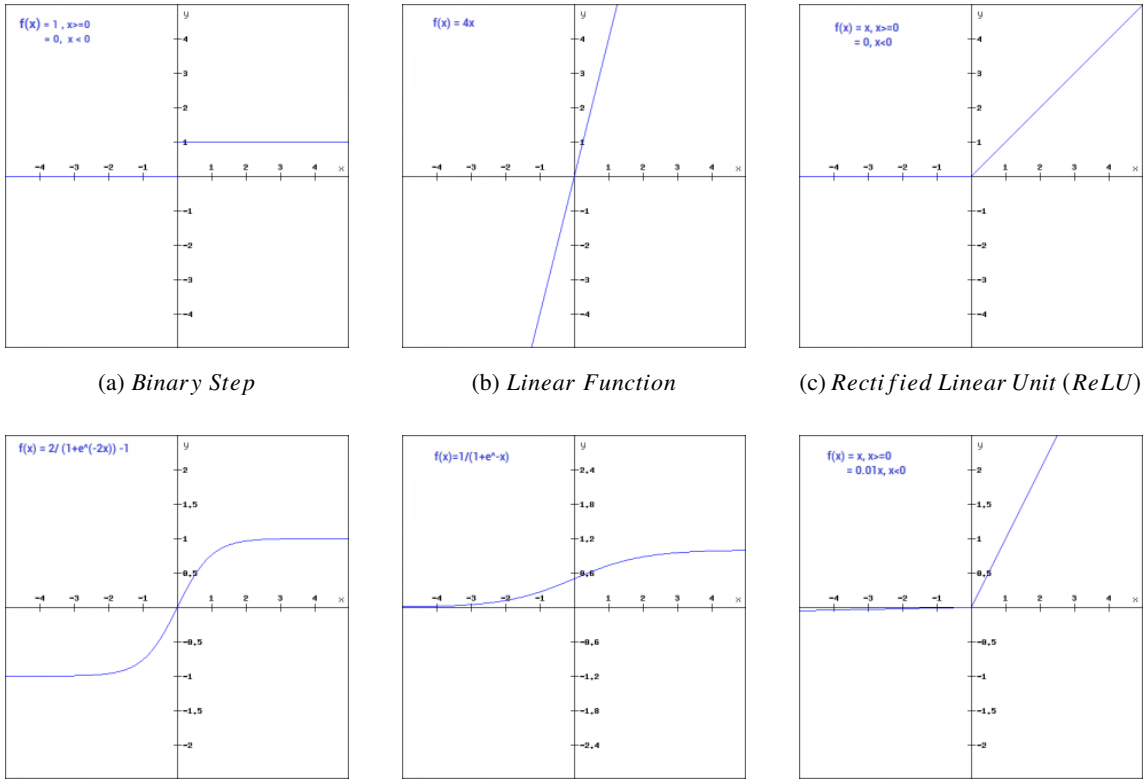


Figura 2.1: Architettura degli strati [13]

Le funzioni di attivazione definiscono quali nodi debbano essere o meno attivati, simulando il comportamento dei neuroni biologici. In generale, si può avere una funzione di attivazione diversa a seconda del *layer*. Le principali funzioni di attivazione sono rappresentate nella figura 2.2 e le relative equazioni nella tabella 2.1.



(a) *Binary Step* (b) *Linear Function* (c) *Rectified Linear Unit* (*ReLU*)

(d) *Hyperbolic Tangent* (*Tanh*) (e) *Sigmoid* (f) *Leaky ReLU*

Figura 2.2: Grafici delle principali funzioni di attivazione [14]

Una rete neurale può dunque essere vista come una funzione matematica che trasforma un insieme di variabili $\mathbf{x} = (x_1, ..., x_n)$ (input della rete) in un insieme di variabili $\mathbf{y} = (y_1, ..., y_m)$ (output della rete). Durante questa trasformazione, ogni ingresso viene moltiplicato per il relativo peso w. Alla somma di questi prodotti viene eventualmente aggiunto un *bias* (i.e. una soglia di attivazione del neurone). Al totale così ottenuto viene applicata la funzione di attivazione φ, il cui risultato costituisce l'output della rete. Lo scopo della fase di addestramento di una rete neurale è proprio quello di trovare i valori dei pesi e del *bias* di ogni neurone. Applicando alla rete un ingresso conosciuto, i pesi e il *bias* così calcolati devono garantire un'uscita il più possibile vicina al valore desiderato.

Tabella 2.1: Equazioni delle principali funzioni di attivazione

Funzione	**Equazione**	**Intervallo**
Binary Step	$f(x) = \begin{cases} 0 \; se \; x < 0 \\ 1 \; se \; x \leq 0 \end{cases}$	$\{0\} \bigcup \{1\}$
Linear	$f(x) = x$	$(-\infty, +\infty)$
ReLU	$f(x) = \begin{cases} 0 \; se \; x < 0 \\ x \; se \; x \leq 0 \end{cases}$	$[0, +\infty)$
Tanh	$f(x) = \frac{2}{1 + e^{-2x}} - 1$	$(-1, 1)$
Sigmoid	$f(x) = \frac{1}{1 + e^{-x}}$	$(0, 1)$
Leaky ReLU	$f(x) = \begin{cases} 0 \; se \; x < 0 \\ 0.01x \; se \; x \leq 0 \end{cases}$	$(-\infty, +\infty)$

L'addestramento di una rete neurale artificiale passa dalla scelta dell'algoritmo di apprendimento, che può essere di tre tipi:

- Apprendimento supervisionato: Tipo di apprendimento nel quale abbiamo a disposizione un insieme di dati comprendente esempi di input con gli output desiderati. La rete inferisce la relazione che lega dati di ingresso e dati di uscita, aumentando la sua capacità di previsione. Attraverso la tecnica della *backpropagation* [15,16] (retro-propagazione dell'errore), i dati sono utilizzati allo scopo di modificare i pesi ed altri parametri della rete stessa. Se l'addestramento ha successo, la rete risulta in grado di prevedere, dato un esempio di ingresso appartenente allo stesso set di dati utlizzato per il training, il relativo output. Questo algoritmo di apprendimento è utilizzato principalmente per problemi di classificazione e regressione.

- Apprendimento per rinforzo: In questo caso non vengono utilizzate le coppie di dati in forma ingresso-uscita, ma si fornisce esclusivamente un *feedback* sull'esito positivo o negativo di una risposta. In pratica è un tipo di apprendimento che si basa sulle informazioni riguardanti la vicinanza all'output desiderato.
- Apprendimento non supervisionato: A differenza dell'apprendimento supervisionato, il set di dati a disposizione è costituito dai soli dati di ingresso. Compito della rete sarà infatti quello di inferire proprietà comuni tra i vari dati, sulla base delle quali effettuare previsioni sugli input successivi. Naturalmente, la buona riuscita di questo tipo di addestramento dipende molto dai dati di ingresso. L'apprendimento non supervisionato è utilizzato, ad esempio, per risolvere problemi di *clustering*.

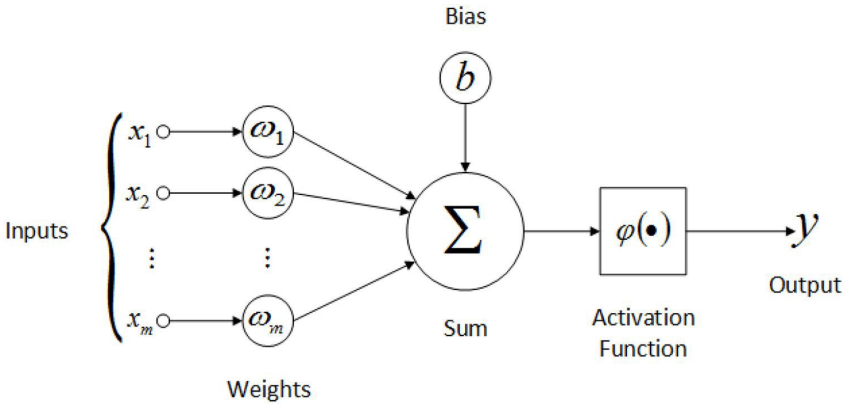


Figura 2.3: Il calcolo dell'attivazione del neurone [17]

A seconda del tipo di uscita che ci aspettiamo da un determinato sistema, possiamo individuare due principali categorie di *task* supervisionati: regressione e classificazione. Nel caso della regressione, puntiamo ad un output continuo, non limitato ad un insieme discreto. Al contrario, il dominio della classificazione è discreto: gli input sono suddivisi in più classi e l'output consiste nell'assegnamento di una classe, tra quelle disponibili, ai dati di ingresso. A questi si aggiunge il *clustering*, un *task* non supervisionato nel quale i dati con caratteristiche e proprietà simili vengono radunati in *cluster* (gruppi). Il dominio del *clustering* è costituito dai soli dati di ingresso, i cui *cluster* di appartenenza rappresentano gli output corrispondenti.

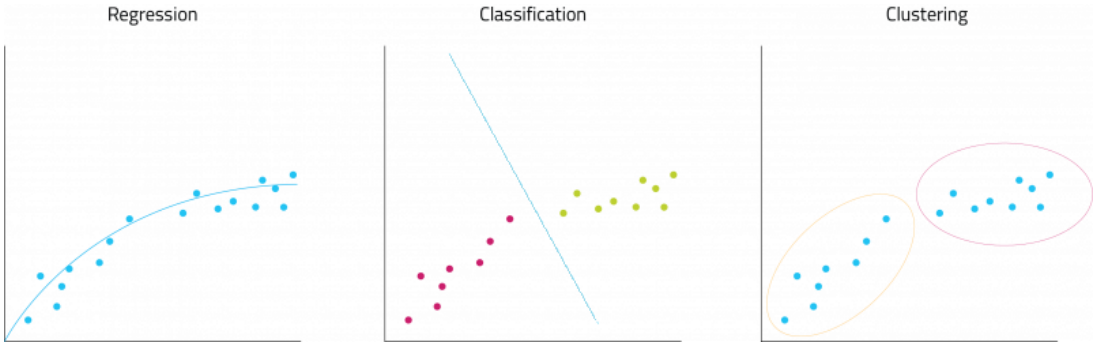


Figura 2.4: Compiti di regressione, classificazione e *clustering* [18]

Le reti neurali precedentemente descritte sono chiamate *feed-forward*, in virtù della loro natura "a catena aperta". Le connessioni tra i neuroni di tali reti non formano cicli, i.e. ogni connessione avanza dall'input all'output della rete senza mai tornare indietro. Di conseguenza l'uscita attuale dipende esclusivamente dall'ingresso attuale. In sostanza tale tipo di rete neurale non possiede memoria degli ingressi precedenti.

Le reti neurali *feed-forward* risultano quindi molto utili nel caso in cui i vari ingressi si presentino in modo indipendente, senza correlazione tra loro. L'impiego di queste reti è indicato nel caso di esempi e situazioni statiche, quali l'analisi e il riconoscimento di oggetti in una foto. Tuttavia, l'assenza di memoria risulta particolarmente problematica in caso di comportamenti "dinamici" e situazioni che hanno bisogno di essere contestualizzate (e.g. previsioni del meteo). Riconoscere un comportamento in un video o uno stato emotivo (grazie all'ausilio di sensori specifici) presuppone un'analisi temporale. Se esaminiamo la foto di una persona con un piede alzato, non possiamo sapere se stia correndo, camminando oppure abbia alzato un piede per un altro motivo. Allo stesso modo, non possiamo immaginare lo stato emotivo di un essere umano grazie ad una sola misurazione del battito cardiaco. Tuttavia, l'oscillazione di quest'ultimo, combinata ai dati di altri sensori, fornisce le condizioni e il contesto necessari per poter individuare uno specifico status psicologico del soggetto in questione. Ciò presuppone l'organizzazione dei dati in sequenze temporali, ovvero segmenti di dati comprendenti misurazioni effettuate nell'arco di un determinato numero di secondi.

A causa dell'assenza di memoria delle reti neurali artificiali *feed-forward*, è dunque necessario l'impiego di un diverso tipo di rete neurale, che possieda memoria degli ingressi precedenti a quello attuale: le reti neurali ricorrenti (*Recurrent Neural Networks*, RNN) [1-3].

2.2 Reti neurali ricorrenti

Una rete neurale ricorrente (*Recurrent Neural Network*, RNN) [1-3] è una rete neurale artificiale in cui sono presenti cicli tra i neuroni. I valori di uscita di uno strato di un livello superiore vengono utilizzati come valori d'ingresso per uno strato inferiore. Ciò permette di considerare uno degli strati come memoria di stato, rendendo dunque la rete in grado di adottare un comportamento dinamico sulla base delle informazioni ricevute in precedenza. In pratica una RNN riceve gli input e produce un output che rimanda indietro a se stessa. Questa caratteristica risulta molto importante per quanto riguarda l'analisi predittiva effettuata su sequenze temporali di dati.

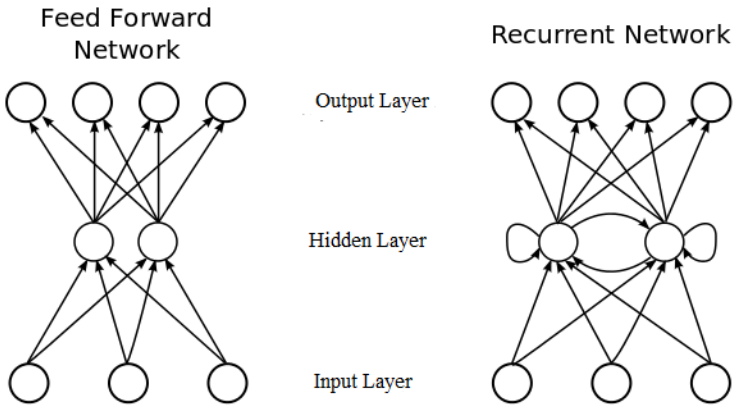


Figura 2.5: Reti neurali *feedforward* vs. ricorrenti [19]

Oltre alle reti neurali ricorrenti standard (*Vanilla RNN*) [1-3, 8-10], saranno impiegati altri tipi di unità RNN, ossia LSTM (*Long Short Term Memory*) [8,10], GRU (*Gated Recurrent Units*) [9,10] e ESN (*Echo State Network*) [4-7], descritte nelle prossime sezioni.

2.2.1 *SimpleRNN*

Le reti neurali ricorrenti condividono la stessa organizzazione. Ciò che le differenzia è la struttura delle unità. SimpleRNN, della libreria *Keras* [20], corrisponde alla classe delle reti neurali ricorrenti standard, la cui struttura è mostrata nella figura 2.6.

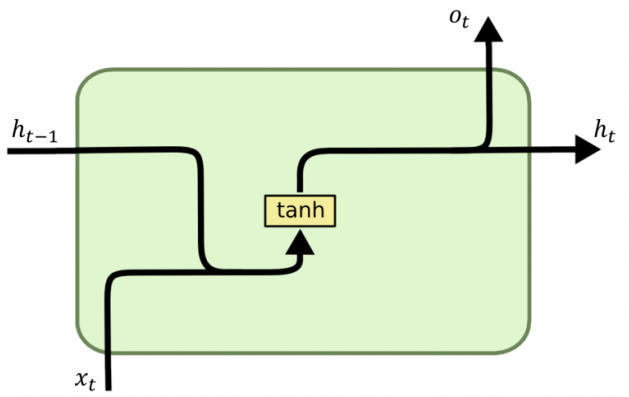


Figura 2.6: Architettura RNN [21]

Ad ogni istante t, l'input è frutto della combinazione tra l'ingresso corrente x_t e il precedente stato nascosto h_{t-1}. La funzione di attivazione (non necessariamente *tanh* come nella figura) produce quindi l'output o_t e lo stato nascosto h_t al tempo t. Quest'ultimo, combinato al prossimo ingresso x_{t+1}, costituirà l'input al tempo $t+1$. Per aggiornare, ad ogni istante t, i vari parametri dei neuroni (pesi e *bias*), viene utilizzato il metodo della *backpropagation*: ad ogni iterazione ogni peso della rete neurale viene aggiornato in modo proporzionale alla derivata parziale della funzione costo (*loss function*) rispetto al peso stesso. Nella figura 2.7 è rappresentato il meccanismo della *backpropagation* in una porzione di RNN *unfolded*, i.e. una parte di RNN rappresentata come una rete neurale di tipo *feed-forward*.

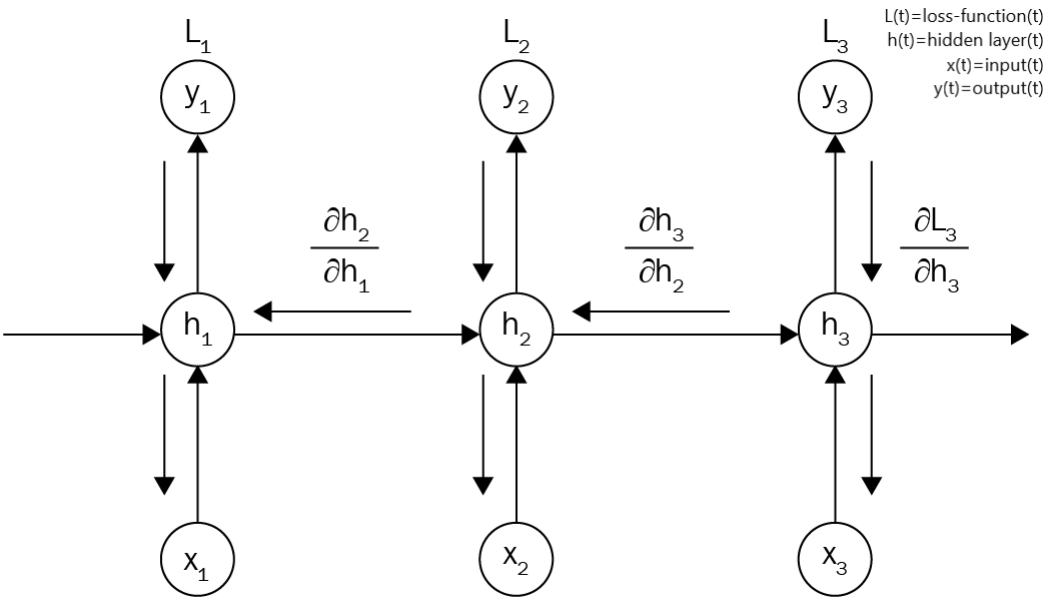


Figura 2.7: Schema *backpropagation* [22]

Consideriamo tre stati (h_1, h_2, h_3), tre input (x_1, x_2, x_3), tre output (y_1, y_2, y_3) e l'errore L_3. Quest'ultimo viene propagato all'indietro mediante il calcolo del gradiente ad ogni step, a partire dallo stato h_3: $\frac{\partial L_3}{\partial h_3}$. In una RNN standard dunque, l'addestramento avviene mediante la retro-propagazione dell'errore attraverso il tempo [23]. Ad ogni step viene calcolato il gradiente (derivata parziale della *loss function* rispetto al peso stesso), che viene utilizzato per aggiornare i pesi della rete. L'algoritmo di *backpropagation* prevede dunque la moltiplicazione in catena dei gradienti lungo gli strati. Questa propagazione all'indietro può tuttavia condurre a due problemi, a seconda della funzione di attivazione utilizzata: la scomparsa del gradiente e l'esplosione del gradiente.

Scomparsa del gradiente

Il problema del *vanishing gradient* [24-26] sorge in caso di derivate molto piccole. Se utilizziamo funzioni di attivazione del tipo sigmoidale o tangente iperbolica, i valori dei loro gradienti restano compresi tra 0 e 1. La moltiplicazione in catena di tali valori lungo i *layer* porta quindi il valore complessivo a diminuire velocemente, via via che il gradiente

viene propagato all'indietro. Valori più piccoli del gradiente influiscono in maniera molto marginale sull'aggiornamento dei pesi, perciò la rete non apprende adeguatamente dagli input precedenti.

Esplosione del gradiente

Il problema dell'*exploding gradient* [26,27] è opposto a quello del *vanishing gradient*. Esso sorge infatti in caso di valori dei gradienti superiori a 1, ad esempio se vengono utilizzate funzioni di attivazione come la ReLU o la lineare pura. La propagazione all'indietro di valori superiori a 1 può portare il valore complessivo del gradiente a crescere enormemente lungo la catena di neuroni, causando uno sproporzionato aggiornamento dei pesi.

Per limitare questi due effetti negativi dovuti alla retro-propagazione, possiamo utilizzare neuroni ricorrenti più complessi rispetto a quello standard, come i neuroni LSTM [8,10] (*Long Short Term Memory*) e GRU [9,10] (*Gated Recurrent Units*).

2.2.2 *LSTM*

Il neurone LSTM [8,10] presenta al suo interno diverse porte (*gate*) che, durante la fase di addestramento, agiscono sulla memorizzazione delle informazioni, sulla combinazione tra ingresso e stato interno e sulla restituzione dell'output.

- *Forget Gate*: Decide se l'informazione in ingresso debba essere mantenuta o dimenticata. Prende in input l'ingresso corrente x_t e il valore della *backpropagation* precedente h_{t-1}, ai quali viene applicata una funzione di attivazione di tipo sigmoidale $\sigma(x) = \frac{1}{1 + e^{-x}}$. σ, che restituisce un valore compreso tra 0 e 1. Un valore uguale ad 1 corrisponde a mantenere l'informazione completa in memoria, mentre un valore pari a 0 coincide con la sua totale eliminazione. Questa uscita viene successivamente moltiplicata al valore del precedente stato c_{t-1}.

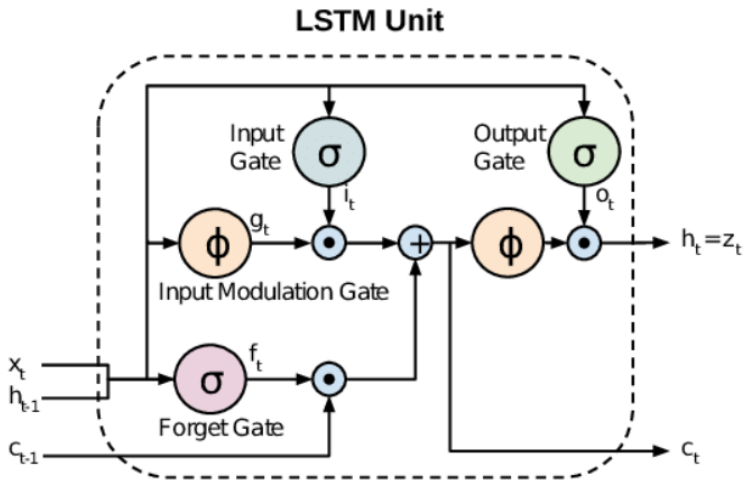


Figura 2.8: Architettura LSTM [28]

- *Input Gate*: Crea un vettore i_t di valori compresi tra 0 ed 1, che sceglie quali informazioni aggiungere al risultato uscente dal *forget gate*. Alle nuove informazioni viene cioè applicata una funzione di attivazione ϕ, dando luogo al vettore g_t. Questo viene poi moltiplicato a i_t, che dunque esegue una selezione delle informazioni. L'elaborazione finale dalle operazioni avvenute nell'*input gate* e nel *forget gate* diventa lo stato attuale del neurone c_t.
- *Output Gate*: Decide, in maniera analoga rispetto agli altri due *gate* (i.e. mediante l'applicazione di una funzione di attivazione sigmoidale), quali valori fornire in uscita. Esso determina lo stato nascosto successivo.

2.2.3 *GRU*

Il neurone GRU [9,10] risulta strutturalmente più semplice del neurone LSTM. Esso infatti gestisce solo due *gate*: l'*update gate* e il *reset gate*.

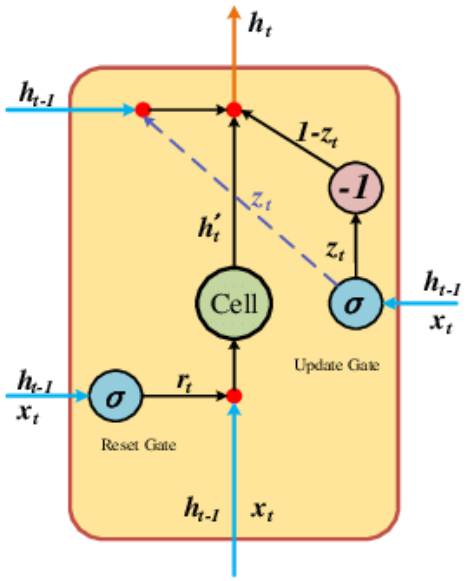


Figura 2.9: Architettura GRU [29]

- *Update Gate*: Applica la funzione di attivazione sigmoidale σ all'ingresso corrente x_t e al valore della *backpropagation* precedente h_{t-1}, per decidere se aggiornare o meno il valore dello stato corrente. Ne risulta un vettore z_t di valori compresi tra 0 e 1. Più valori si avvicinano a 1, più informazioni passate vengono mantenute, aggiornando quindi lo stato in misura minore.
- *Reset Gate*: Questa porta è utilizzata per decidere quanta dell'informazione passata dimenticare. Prende in input i vettori x_t e h_{t-1} e restituisce r_t, un vettore di valori compresi tra 0 e 1. Più valori sono vicini a 0, più informazioni del precedente *hidden state* vengono ignorate. La combinazione tra r_t, h_{t-1} e x_t fornisce il contenuto della memoria corrente h'_t, immagazzinato in *Cell*.

L'output finale h_t, che contiene le informazioni per l'unità corrente da trasmettere alla rete, è dato dalla seguente combinazione tra z_t, h_{t-1} e h'_t:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t,$$

dove $\odot$ è definito come il Prodotto di Hadamard. Il prodotto di Hadamard è un'operazione binaria che, date due matrici A,B di uguali dimensioni, restituisce un'altra matrice C della stessa dimensione. Ogni elemento i,j di C è il prodotto degli elementi i,j corrispondenti di A e B. Ad esempio:

$$A(3\times3) \odot B(3\times3) = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \odot \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} & a_{13}b_{13} \\ a_{21}b_{21} & a_{22}b_{22} & a_{32}b_{32} \\ a_{31}b_{31} & a_{32}b_{32} & a_{33}b_{33} \end{bmatrix}.$$

I *gate* dei neuroni LSTM e GRU sono quindi in grado di imparare a selezionare gli input più importanti, memorizzandoli all'interno dell'unità. Questo consente di passare l'informazione anche attraverso lunghe sequenze e ridurre sensibilmente il rischio di scomparsa/esplosione del gradiente.

2.2.4 *ESN*

Il modello *Echo State Network* (ESN) [4-7] è un tipo di modello RNN caratterizzato da un apprendimento efficiente con sequenze temporali, facente parte del paradigma *Reservoir Computing* (RC) [4-7]. Il paradigma RC evita i problemi dovuti alla *backpropagation*, effettuando una concettuale separazione tra la parte non lineare ricorrente (*reservoir*) e la parte *feed-forward* in uscita (*readout*). In questo modello vengono addestrate esclusivamente le connessioni al *readout*, mentre il *reservoir* (fig. 2.10) è generato inizialmente in modo random e non viene mai addestrato.

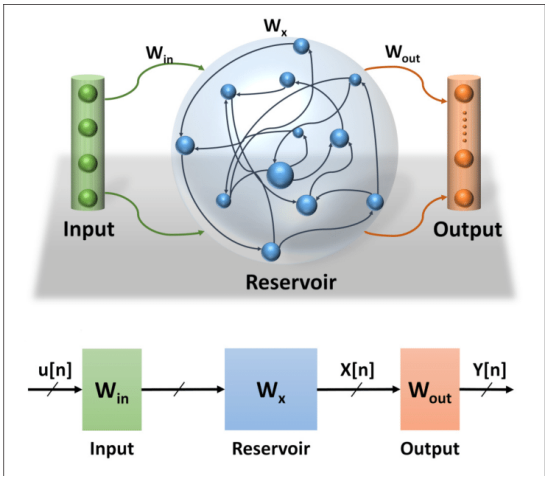


Figura 2.10: Architettura ESN *shallow* [30]

- *Reservoir*: *Layer* ricorrente non lineare creato in maniera casuale, non addestrato durante la fase di training. Il *reservoir* implementa la seguente funzione di transizione di stato:

$$x(n) = tanh(W_{\text{in}}(u(n)) + \theta + W_{\text{x}}x(n-1)),$$

  ove u(n) e x(n) rappresentano rispettivamente l'input e lo stato del *reservoir* al tempo n, W_{in} è la matrice dei pesi di ingresso, θ è il vettore dei pesi *bias* in ingresso e W_{x} è la matrice dei pesi ricorrente, relativa alle connessioni interne della rete. I valori dei pesi in W_{in} e θ vengono scelti da una distribuzione uniforme tra [-$scale_{in}$,$scale_{in}$], ove $scale_{in}$ rappresenta un iperparametro di *input scaling*. Partendo da un valore *bias* unitario, anch'esso viene dunque scalato in base al parametro di *input scaling*.
- *Readout*: *Layer* lineare addestrato, responsabile dell'output effettivo. L'uscita è data dalla seguente combinazione lineare:

$$y(n) = \varphi_{\text{out}}(W_{\text{out}}x(n) + \theta_{\text{out}}),$$

  in cui y(n) rappresenta l'output al tempo n, W_{out} è la matrice dei pesi in uscita dal *reservoir* e in input rispetto al *readout*, φ_{out} è una funzione di attivazione (e.g. *sigmoid*) e θ_{out} è il vettore dei pesi *bias-to-readout*.

Consideriamo da qui in avanti il modello "LI-ESN" (*Leaky Integrator Echo State Network*) [4], una variante di ESN a cui aggiungiamo l'iperparametro *a* (*leaky integrator*). Il valore di *a* si riferisce alla velocità di risposta del *reservoir* all'input. A valori maggiori di *a* corrisponde una reazione più veloce all'ingresso. La funzione di transizione viene dunque modificata:

$$x(n) = (1 - a)x(n - 1) + a \times tanh(W_{\text{in}}(u(n)) + \theta + W_{\text{x}}x(n - 1))$$

La formula LI-ESN risulta essere una generalizzazione della precedente. Infatti, sostituendo il valore 1 al parametro *a*, otteniamo l'equazione di stato ESN standard.

Echo State Property

La ESP [4-7] è una proprietà specifica del *reservoir*. Essa afferma che l'effetto di uno stato precedente x(n) e di un input precedente u(n) su un futuro stato x(n + k) dovrebbe gradualmente svanire col passare del tempo. Le dipendenze iniziali vengono dunque progressivamente perse.

Sono presenti due condizioni sulla matrice dei pesi ricorrente W_x:

- $\sigma_{\max}(W_x) < 1$,
  $\sigma_{\max}(W_x) = \|W_x\|_2$. (condizione sufficiente)
- $\rho(W_x) < 1$,
  $\rho(W_x) = max_i(|\lambda_i|)$. (condizione necessaria)

Per garantire il rispetto della seconda (necessaria) proprietà, W_x viene inizialmente generata come una matrice *random*, per poi essere riscalata sulla base della seguente formula:

$$W_x = W_{\text{random}} \frac{\rho'}{\rho(W_{\text{random}})},$$

ove ρ' è il raggio spettrale desiderato, iperparametro da noi stabilito.

Deep Echo State Network

A questo punto consideriamo una LI-ESN costituita da multipli *reservoir layer*. In questo modello, chiamato *DeepESN* [4], il primo strato è riempito dall'input esterno e si comporta come il *reservoir* di una ESN base. Ogni *layer* successivo ha invece, come input, l'output del precedente strato. La funzione di transizione di stato non cambia e ve n'è una per ogni *layer*. Da strato a strato possono perciò cambiare i parametri della rete e gli iperparametri, ma la struttura della funzione rimane la stessa. Per quanto riguarda il *readout*, la componente x(n) è data dalla combinazione lineare degli output di tutte le unità *reservoir* ($x^{(1)}(n)x^{(2)}(n)...x^{(N_L)}(n)$, N_L = numero di *layer reservoir*).

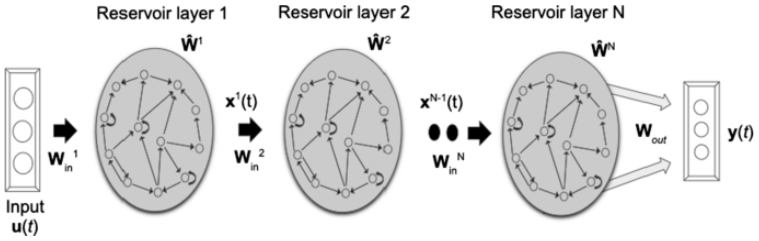


Figura 2.11: Architettura *DeepESN* [31]

3

RETI NEURALI RICORRENTI PER LA PREDIZIONE DELLO STATO FISICO ED EMOTIVO

Questo capitolo costituisce la parte centrale dello studio, in cui andiamo a delineare le tecniche e i metodi utilizzati per la predizione dello stato fisico ed emotivo. In Sezione 3.1 introduciamo il problema da analizzare, ovvero il riconoscimento dello stato fisico ed emotivo tramite l'impiego di dispositivi indossabili. Successivamente, in Sezione 3.2, è presente un'analisi degli approcci ricorrenti per questo tipo di problema. Vengono dunque descritti i sensori principalmente impiegati, le modalità di estrazione delle misurazioni biometriche e le tecniche di *pre-processing* dei dati maggiormente diffuse. La Sezione 3.3 riguarda le librerie adoperate durante la scrittura del codice, mentre nella Sezione 3.4 ci occupiamo di descrivere, per ogni dataset, i sensori adoperati, le modalità di estrazione dei campioni biometrici dai soggetti, le attività fisiche ed emotive registrate e l'organizzazione dei dati all'interno dei vari file e cartelle. Infine, nella Sezione 3.5, viene trattato il processo di *model selection* e la relativa selezione degli iperparametri, effettuata separatamente per ciascun modello e per ciascun dataset.

3.1 Descrizione del problema

Il riconoscimento dello stato fisico ed emotivo mira a rilevare varie misurazioni da una persona, e.g. attraverso accelerometro o pressione sanguigna, con l'obiettivo generale di garantire una vantaggiosa interazione tra uomo e macchina. Consideriamo di avere a che fare con dei dispositivi indossabili, come smartband e smartwatch, in quanto consentono un feedback immediato sullo stato fisico e psicologico del soggetto. D'altro canto, l'impiego di questa tecnologia presuppone una minore potenza di calcolo relativa all'hardware, motivo per il quale ci siamo concentrati sulla ricerca della massima efficienza, i.e. sul miglior compromesso tra la complessità dei modelli e l'affidabilità dei risultati.

Per riconoscere adeguatamente un movimento o una condizione psicologica di un soggetto (e.g. stress), dobbiamo considerare una suddivisione delle misurazioni in sotto-sequenze temporali. Ognuna di queste presenta determinati valori, provenienti da specifici sensori utilizzati, che indicano lo svolgimento di un'attività fisica o il sussistere di uno stato emotivo. E' quindi necessario l'impiego di una tipologia di reti neurali che tenga conto di questa struttura temporale: le reti neurali ricorrenti. La struttura ciclica delle connessioni (sez. 2.2) permette loro di adottare un comportamento dinamico, che prenda in considerazione anche le informazioni ricevute precedentemente.

Effettuiamo dunque un confronto bilanciato dei risultati raggiunti mediante l'impiego di quattro modelli di reti neurali ricorrenti, addestrati e valutati su dati provenienti da cinque dataset diversi, relativi all'attività sia fisica che emotiva. Tale confronto comprende anche i risultati conseguiti (sugli stessi dataset) da parte di altri autori, attraverso diversi metodi di *pre-processing* e di *model selection*.

In questa sezione sono descritte le tecniche di *model selection*, la composizione dei cinque dataset e le librerie utilizzate, mentre nella sezione successiva esponiamo il procedimento di *pre-processing* dei dati e i risultati raggiunti, con l'ausilio di tabelle e grafici.

3.2 Analisi degli approcci per la predizione dello stato fisico ed emotivo

L'espressione facciale, la voce, il linguaggio del corpo e diversi parametri biologici, quali l'attività elettrodermica e la temperatura, rappresentano molteplici modi (espliciti e non) di esprimere stati emotivi. Per la predizione di quest'ultimi impieghiamo proprio i segnali provenienti dai parametri biologici, rilevati mediante l'impiego di appositi strumenti. I sensori maggiormente adoperati per dataset di questo tipo sono elettrocardiogramma, termometro ed elettroencefalogramma, che risultano essere molto precisi. Tra l'altro, la frequenza cardiaca e la temperatura corporea sono ormai facilmente misurabili da bracciali e orologi smart e dunque ben si prestano per questa analisi. I soggetti sono messi in condizioni psicologiche tali da riuscire a campionare le relative misurazioni con un buon grado di certezza. Nel dataset WESAD, ad esempio, per indurre lo stato di stress, è stato chiesto ai partecipanti di preparare un discorso autodescrittivo, da recitare dinanzi ad un ristretto gruppo di persone. Nonostante si rilevino numerosi elementi in comune fra gli stessi stati psicologici indotti in soggetti diversi, ogni essere umano reagisce in modo differente agli stimoli esterni, motivo per il quale i modelli sono sempre testati su dati appartenenti a soggetti su cui è già stato effettuato l'addestramento.

L'attività fisica è principalmente riconoscibile mediante accelerometro e giroscopio, presenti sulla maggior parte dei dispositivi indossabili. L'accelerometro, in generale, risulta essere più preciso del giroscopio nel rilevamento delle attvità. Ciò nonostante, è la combinazione dei due a fornire la precisione maggiore. Nei dataset PAMAP2 e OPPORTUNITY l'accelerazione 3D e i dati del giroscopio fanno parte delle *features* misurate dai sensori IMU (*Inertial Measurement Units*), dispositivi elettronici molto utilizzati per dataset biometrici. I sensori IMU possono inoltre rilevare la temperatura corporea e, grazie al magnetometro, anch'esso presente su smartphone e smartwatch, persino la direzione del campo magnetico intorno al soggetto. Le principali attività analizzate sono quelle di tutti i giorni, come correre, camminare, stare in piedi, seduti o sdraiati. Queste rappresentano i movimenti principali compiuti da ciascuna persona durante l'arco della giornata. Ai soggetti di ogni dataset è stato perciò chiesto di eseguire tali azioni in modo naturale, campionando i movimenti mediante i sensori precedentemente citati.

Le tecniche di *pre-processing* utilizzate in questo e altri studi differiscono per l'elaborazione dei dati, ma non per quanto riguarda l'impostazione generale. Si predilige infatti un'organizzazione a sequenze temporali, ognuna delle quali si riferisce ad un determinato stato emotivo/fisico. Da questo punto di vista quindi, i vari approcci risultano assimilabili nella struttura a sottosequenze.

Studi effettuati in precedenza hanno scelto di adoperare, per questo genere di *task*, diversi tipi di modelli di reti neurali (e non). Sono stati confrontati vari algoritmi di apprendimento, quali *Decision Tree* [33,34,37], *Random Forest* (fig. 3.1) [32-34] e *k-Nearest Neighbors* [33,34,37,38]. Quest'ultimo in particolare risulta essere il più impiegato.

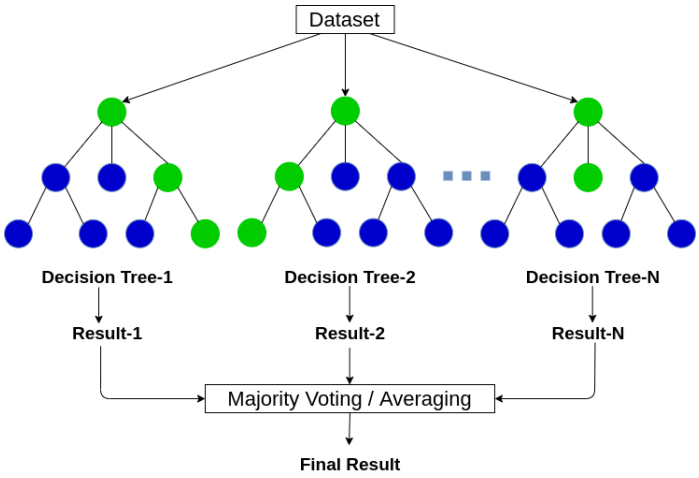


Figura 3.1: Classificatore *Random Forest* [42]

Inoltre, i precedenti studi in letteratura si sono concentrati sull'uso di reti neurali *feed-forward* [34] o dell'algoritmo SVM (*Support Vector Machine*) [34,41]. Sebbene quest'ultimo risulti più efficiente nel caso binario (e.g. considerando esclusivamente la divisione "stress/no-stress" nel dataset WESAD), è possibile estenderlo ad un *task* multi-classe, considerando un iperpiano separatore con $k > 2$ dimensioni (fig.3.2). Ciò è stato fatto, ad esempio, per i dataset HAR [34] e ASCERTAIN [41].

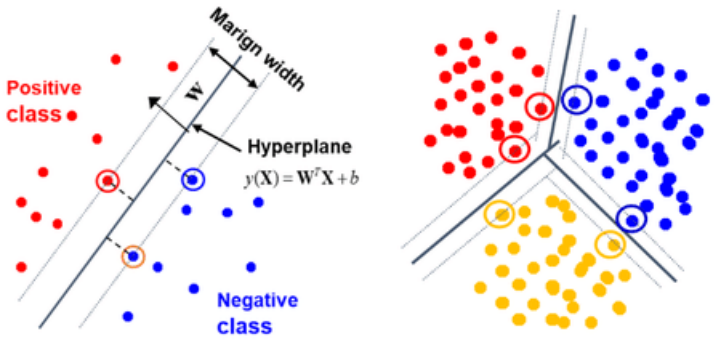


Figura 3.2: Classificatore SVM binario vs. multi-classe [43]

Tuttavia, le reti neurali sono state scarsamente sfruttate per i cinque dataset qui esaminati. In rari casi sono state analizzate le performance delle *Deep Neural Network* (DNN), reti neurali artificiali *feed-forward* con più di un *hidden layer*, ma in generale è stato preferito un approccio mediante algoritmi più "classici" nell'ambito del *Machine Learning*.

Relativamente ai cinque dataset trattati, le reti neurali (ricorrenti) sono state dunque impiegate in minima parte per la predizione dello stato fisico ed emotivo. Questo rappresenta l'aspetto di letteratura mancante che cerchiamo di colmare in questo studio.

Nella maggior parte dei precedenti lavori, inoltre, è stata adottata la tecnica della *Cross-Validation*[1] [44-47]. Sebbene essa comporti numerosi vantaggi in grado di aumentare l'efficacia e la precisione dei modelli di reti neurali (e.g. riducendo l'*overfitting*[2] [48]), la sua inclusione in questa ricerca avrebbe significato maggiore complessità e tempo di addestramento. Pertanto abbiamo adottato il metodo *Holdout* [47], suddividendo il dataset una sola volta tra dati di training (inclusi quelli per la validazione) e dati di test.

[1] Suddivisione del set di dati in k parti uguali. Ad ogni passo di addestramento, la k-esima parte dell'insieme di dati va a costituire il validation-set, mentre le restanti k-1 parti costituiscono il training-set.

[2] Un modello è soggetto a *overfitting* quando non è in grado di generalizzare, i.e. il modello si adatta troppo ai dati di training e non risulta efficace sui dati di test.

3.3 Librerie software usate

Per questo lavoro di tesi triennale sono state adoperate varie librerie scritte in linguaggio *Python* e *C++*, principalmente relative alle reti neurali. Quest'ultime sono tre: *TensorFlow* [49], *Keras* [20] e *Scikit-Learn* [50], impiegate per le fasi di *model selection* e training dei modelli. Esse offrono infatti numerosi strumenti per l'inizializzazione, l'addestramento e la valutazione delle performance dei vari modelli di RNN.

3.3.1 *TensorFlow*

Libreria *open-source* di *deep learning*, sviluppata e mantenuta da *Google*. Offre numerosi strumenti per *task* di machine learning, in vari linguaggi tra cui *Python*, *C++* e *Java*. Più specificatamente, *TensorFlow* è un framework che offre varie API, sia di alto (e.g. *Keras*) che di basso livello. In pratica, facciamo uso di *TensorFlow* nella misura in cui adoperiamo la libreria *Keras*, la quale offre una semplice interfaccia (basata per l'appunto su tale framework) per la rapida implementazione e l'addestramento di reti neurali.

3.3.2 *Keras*

Libreria *open-source* di reti neurali scritta in linguaggio *Python*, che gira su TensorFlow. E' appositamente designata per essere veloce e facile da usare ed è sviluppata da un ingegnere *Google*, François Chollet. *Keras* è utilizzata, in particolare, per l'inizializzazione e il training dei modelli di RNN, oltre che per la conversione dei vettori delle classi di output in corrispondenti matrici binarie.

Metodi e classi principali utilizzate:

- `Sequential()`: Metodo costruttore della classe omonima. Crea un'istanza di un modello *Keras*, al quale è possibile aggiungere vari tipi di *layer* (e.g. ESN, LSTM, Dense), tramite il metodo `add()`.
- `SimpleRNN`, `LSTM`, `GRU` e `Dense`: Classi che implementano i rispettivi tipi di *layer*. Per `Dense` sono specificati, come argomenti, il numero di classi, la funzione di attivazione e i parametri relativi alla regolarizzazione (come si può vedere nella figura 3.3), mentre per gli altri tipi di *layer* solo il numero di unità.

```
modelESN.add(tf.keras.layers.Dense(4, activation='softmax',
             kernel_regularizer = regularizers.l1_l2(l1 = 1e-6, l2 = 1e-6),
             bias_regularizer = regularizers.l1_l2(l1 = 1e-6, l2 = 1e-6),
             activity_regularizer = regularizers.l1_l2(l1 = 1e-6, l2 = 1e-6)))
```

Figura 3.3: Aggiunta di uno strato `Dense()` ad un modello

- `Regularizers`: Classe contenente tre funzioni di regolarizzazione (`l1`, `l2` e `l1_l2`).
- `Model`: Classe della quale utilizziamo i seguenti metodi:
  - `summary()`: Stampa un riepilogo della rete.
  - `compile()`: Configura il modello per l'addestramento. Specifichiamo tre argomenti: il nome dell'ottimizzatore usato, la *loss function* applicata e la metrica da valutare durante le fasi di addestramento e test (fig.3.4).

```
modelESN.compile(loss = 'categorical_crossentropy',
                 optimizer = opt, metrics = ['accuracy'])
```

Figura 3.4: Metodo `compile()`

  - `fit()`: Addestra il modello per un determinato numero di epoche. Gli argomenti specificati sono i dati di input, i dati target (la matrice binaria delle classi), il numero di epoche massimo, i dati sulla base dei quali valutare *loss* e *accuracy* del modello alla fine di ogni epoca (i.e. `validation_data`) e la *callback* (`EarlyStopping`, sez.3.5.1) applicata (fig. 3.5).

```
modelESN.fit(X_trainWES, y_trainWES, epochs = 200,
             validation_data = (X_valWES, y_valWES), callbacks = [es])
```

Figura 3.5: Metodo `fit()`

  - `evaluate()`: Restituisce *loss* e *accuracy* sul test-set del modello addestrato. Al metodo vengono passati due argomenti: i dati di input e i dati target del test-set.
- `to_categorical()`: Metodo che converte un vettore di numeri interi in una matrice di classi binarie. Prende come argomenti un vettore di interi e il numero di classi.

3.3.3 *Scikit-Learn*

Altra libreria *open-source* che fornisce strumenti per il *fitting* dei modelli, il *pre-processing* dei dati, la *model selection* e numerose altre *utilities* relative alle reti neurali. Per questo lavoro di tesi sono stati principalmente utilizzati i seguenti metodi:

- `train_test_split()`: Suddivide vettori o matrici di dati in training-set e test-set. Come argomenti passiamo i vettori/matrici su cui effettuare lo *splitting* e le relative proporzioni (fig.3.6).

```
#Splitting data into training(80%), validation(10%) and test-set(10%)
(X_WES, X_testWES, y_WES, y_testWES) = train_test_split(X_WES, y_WES, test_size = 0.1, train_size = 0.9)
(X_trainWES, X_valWES, y_trainWES, y_valWES) = train_test_split(X_WES, y_WES, test_size = 0.1, train_size = 0.9)
```

Figura 3.6: Metodo `train_test_split()`

- `ParameterSampler()`: Generatore casuale di combinazioni di parametri, data una distribuzione di quest'ultimi. Specifichiamo due argomenti: un dizionario che ha come chiavi i nomi degli iperparametri da testare e come valori le rispettive distribuzioni, più il numero di iterazioni (i.e. il numero di combinazioni da provare).
- `KerasClassifier()`: Costituisce l'implementazione in *Keras* delle API del classificatore *scikit-learn*. In pratica è un *wrapper* che consente la compatibilità tra i metodi di *Scikit-Learn* e i modelli *Keras*. Prende come argomento `build_fn()`, una funzione che costruisce un modello *Keras* specificato, al quale possiamo poi applicare i seguenti metodi di *Scikit-Learn* per la selezione degli iperparametri:
  - `set_params()`: Metodo utilizzato all'interno del generatore `ParameterSampler()`. Per ogni iterazione del generatore, setta la corrispondente combinazione di iperparametri.
  - `score()`: Funzione che restituisce l'*accuracy* media su un determinato set di dati. Impiegata per valutare ciascuna combinazione di iperparametri e scegliere la migliore. Come argomenti passiamo i dati di input e di output appartenenti al validation-set.

3.3.4 *Altre librerie*

Per aprire i file ".pkl" abbiamo utilizzato `load()` della libreria *Pickle*. Per lavorare sui *dataframe* contenenti le misurazioni dei soggetti è stata usata la libreria *Pandas*. Il *resampling* dei dati è stato effettuato attraverso la funzione `resample()` della libreria *SciPy*. Infine, la libreria *NumPy* è servita per l'esecuzione di varie operazioni matematiche (e.g. concatenazione di matrici).

3.4 Descrizione dei dataset

Per lo scopo di questa ricerca sono stati utilizzati cinque dataset contenenti informazioni di tipo biometrico, raccolte da diversi soggetti attraverso strumenti quali accelerometro, elettrocardiogramma, giroscopio ecc. I dati sono stati successivamente pre-processati e usati per l'addestramento e il testing dei modelli. In generale, poiché ci occupiamo di *task* di classificazione, ad ogni serie temporale di misurazioni è associata una classe, un numero corrispondente ad un determinato stato emotivo (e.g. stato di stress o di piacere) o ad una particolare condizione fisica del soggetto (e.g. sdraiato, seduto o in piedi). Il *pre-processing* dei dati verrà descritto in dettaglio nella Sezione 4.1.

3.4.1 *WESAD*

WESAD (*WEarable Stress and Affect Detection*) [32,33] è un dataset contenente informazioni fisiologiche appartenenti a 15 soggetti, registrate grazie a due dispositivi, uno messo intorno al torace (RespiBAN[3]) e uno indossato al polso (Empatica E4[4]). Sono incluse le seguenti misurazioni: pulsazioni del sangue (BVP), elettrocardiogramma (ECG), attività elettrodermica (EDA), elettromiogramma (EMG), respirazione (RESP), temperatura corporea (TEMP) e accelerazione su tre assi (ACC).

I dati provenienti dal torace sono stati campionati ad una frequenza di 700 Hz e comprendono tutte le misurazioni citate in precedenza. Il dispositivo Empatica E4 è stato indossato

[3] `http://www.biosignalsplux.com/en/respiban-professional`
[4] `http://www.empatica.com/research/e4/`

dai soggetti al polso della mano non dominante e contiene i seguenti dati, registrati a frequenze diverse tra loro: pulsazioni del sangue (64 Hz), accelerazione su tre assi (32 Hz), attività elettrodermica (4 Hz) e temperatura (4 Hz).

Originariamente, i dati erano relativi a 17 soggetti ma, a causa di un malfunzionamento dei sensori, i soggetti 1 e 12 sono stati scartati.

L'obiettivo di questo studio era quello di provocare tre differenti stati emotivi nei partecipanti, denominati *baseline* (condizione neutrale), *stress* e *amusement* (condizione di divertimento). Tali stati sono inoltre intervallati da due sequenze di meditazione guidata (*meditation I, meditation II*). Nel dettaglio:

- **Baseline**: Una volta equipaggiati i partecipanti con i sensori, sono stati registrati venti minuti di misurazioni al fine di indurre uno stato emotivo neutrale. Per suscitare questa condizione, ai soggetti sono state fornite delle riviste da leggere, seduti o in piedi.
- **Amusement**: Per suscitare questo stato, sono stati mostrati undici video divertenti, separati da brevi sequenze neutrali di cinque secondi l'una, per un totale di 392 secondi.
- **Stress**: Per ottenere la condizione di stress, i soggetti sono stati esposti al TSST (*Trier Social Stress Test*) [51], che consiste in un discorso pubblico e un calcolo aritmetico a mente. In particolare, ai partecipanti è stato richiesto di preparare, entro tre minuti, un discorso di cinque minuti riguardante le proprie caratteristiche personali, cercando di dare la miglior impressione possibile ad un gruppo di tre persone. Dopo il monologo, il gruppo ha chiesto ai soggetti di contare da 2023 a zero entro cinque minuti, mediante step di 17. In caso di errore, essi dovevano ricominciare da capo. Seguono dieci minuti di riposo.
- **Meditation**: Le due fasi precedenti sono seguite da un periodo di meditazione guidata, lungo sette minuti. Lo scopo è quello di far tornare i soggetti alla condizione neutrale, attraverso esercizi di respirazione controllata.

Le due condizioni di *stress* e *amusement*, al fine di evitare conseguenze relative all'ordine ripetuto, sono state intercambiate tra differenti soggetti (tab. 3.1).

Versioni	**Condizioni**					
Versione A	Baseline	Amusement	Meditation I	Stress	Rest	Meditation II
Versione B	Baseline	Stress	Rest	Meditation I	Amusement	Meditation II

Tabella 3.1: Le due versioni dell'ordine delle condizioni

I dati sono organizzati in 15 cartelle, ognuna relativa a un soggetto e contenente un file con estensione ".pkl". Il file in questione è un dizionario, composto dalle seguenti chiavi:

- **'subject'**: ID del soggetto.
- **'signal'**: Include tutte le misurazioni, suddivise in due campi:
  - *'chest'*: Dati relativi al dispositivo RespiBAN (ACC, ECG, EDA, EMG, RESP, TEMP).
  - *'wrist'*: Dati relativi al dispositivo Empatica E4 (ACC, BVP, EDA, TEMP).
- **'label'**: ID della condizione del soggetto, campionato a 700 Hz e associato alle misurazioni contenute in 'signal'. Sono forniti 8 ID, ma solo gli ID 1,2,3 e 4, come specificato dagli autori del dataset, sono da considerare:
  - 0: Non definito.
  - 1: *Baseline condition*.
  - 2: *Stress condition*.
  - 3: *Amusement condition*.
  - 4: *Meditation*.
  - 5/6/7: Da ignorare.

I dati sono stati ricampionati ad una frequenza di 32 Hz per ridurre il numero di campioni utilizzati, troppo elevato alla frequenza massima (700 Hz). Suddividendo il dataset in sequenze di circa tre secondi (corrispondenti a 100 misurazioni) e considerando solo le classi oggetto di studio (i.e. 1, 2, 3, 4), si ottengono 14.229 sottosequenze, delle quali verrà poi scelto un prefisso, al fine di ridurre ulteriormente l'eccessiva mole di dati.

3.4.2 *HAR*

Il dataset *Heterogeneity Activity Recognition* (HAR) [34,35] utilizza il giroscopio e l'accelerometro presenti su smartphone e smartwatch per classificare una determinata attività fisica compiuta da un soggetto.

I device utilizzati sono quattro modelli di smartphone (Samsung Galaxy S3 mini, Samsung Galaxy S3, LG Nexus 4, Samsung Galaxy S+) e due tipi di smartwatch (LG watches, Samsung Galaxy Gears). Il campionamento dei dati viene fatto alla frequenza più alta possibile raggiunta dai vari dispositivi, tra i 50 Hz e i 200 Hz (la frequenza massima dagli smartwatch è inferiore rispetto a quella degli smartphone).

I soggetti partecipanti allo studio sono 9, ad ognuno dei quali è stato richiesto di eseguire 6 attività fisiche (senza alcun ordine predefinito), mentre portava con sé i vari dispositivi.

I dati sono suddivisi in quattro file ".csv", a seconda del device (smartphone o smartwatch) e del sensore (giroscopio o accelerometro) impiegato e condividono la stessa organizzazione in colonne:

- **'Index'**: Numero di riga.
- **'Arrival_Time'**: Tempo in cui la misurazione è arrivata all'applicazione di rilevamento.
- **'Creation_Time'**: *Timestamp* che il sistema operativo associa al campione rilevato.
- **'X',' Y', 'Z'**: Valori forniti dal sensore per i tre assi xyz.
- **'User'**: Utente da cui proviene il campione di dati (i nomi dei soggetti vanno dalla "a" alla "i").
- **'Model'**: Modello di smartphone/smartwatch usato per la serie di misurazioni.
- **'Device'**: Device specifico utilizzato.
- **'Gt'**: Attività svolta dal soggetto, associata alle misurazioni proveninenti dal giroscopio/accelerometro. Può assumere i seguenti valori:
  - 0: Non definito (*null*).
  - 1: Soggetto in piedi (*stand*).
  - 2: Soggetto seduto (*sit*).
  - 3: Soggetto che cammina (*walk*).
  - 4: Soggetto che sale le scale (*stairsup*).
  - 5: Soggetto che scende dalle scale (*stairsdown*).

– 6: Soggetto che pedala in biciletta (*bike*).

La presenza di valori nulli è dovuta alla mancanza di alcune annotazioni riguardanti l'attività svolta dal soggetto. Inoltre, a causa di qualche problema di campionamento, alcuni utenti sono provvisti di un minor numero di misurazioni, ad esempio i soggetti "h" e "i".

Escludendo i soggetti "h" e "i", ricampionando i dati ad una frequenza più bassa (accelerometro dello smartwatch, circa 70 Hz) e considerando due secondi come lunghezza delle sequenze temporali, si ottengono 87.141 sottosequenze. Tuttavia, tenendo presente il fatto che le varie attività vengono ripetute più volte dai soggetti durante lo svolgimento della sessione di misurazioni, è possibile applicare un filtraggio in modo tale da ridurre la mole di dati (vedi sez. 4.1.2).

3.4.3 *PAMAP2*

PAMAP2 (*Physical Activity Monitoring Data Set*) [36,37] contiene dati biometrici di 9 soggetti. Per il campionamento delle informazioni sono stati usati un monitor per il battito cardiaco (9 Hz) e tre sensori wireless IMU (*Inertial Measurement Units*, 100 Hz), posti rispettivamente sul polso della mano dominante, sul petto e sull'anca del soggetto. Le registrazioni dei sensori IMU comprendono dati relativi alla temperatura e alle misurazioni effettuate grazie a due accelerometri, un giroscopio e un magnetometro.

Ai soggetti considerati, di età media di circa ventisette anni, è stato chiesto di effettuare 18 differenti attività (di durata compresa tra uno e tre minuti), di cui 12 principali e 6 facoltative, le quali sono state eseguite solo da alcuni partecipanti. Poiché la maggioranza degli esercizi è stata svolta solo da una parte dei soggetti, sono state scelte solo le attività compiute dal più alto numero di partecipanti. Di queste, sono state selezionate le quattro con più informazioni a disposizione (i.e. dalla durata di tre minuti) e maggiormente attinenti alla quotidianità, ovvero *lying*, *sitting*, *standing* e *walking*.

Molte attività sono mancanti a causa di problemi con i sensori wireless (e.g. perdita di connessione). In particolare, il soggetto 9 è stato scartato per via di una maggiore carenza di misurazioni. Per una panoramica delle attività svolte da ogni soggetto si faccia riferimento alla tabella 3.2.

	Soggetti								
Attività	1	2	3	4	5	6	7	8	9
Lying	✓	✓	✓	✓	✓	✓	✓	✓	
Sitting	✓	✓	✓	✓	✓	✓	✓	✓	
Standing	✓	✓	✓	✓	✓	✓	✓	✓	
Walking	✓	✓	✓	✓	✓	✓	✓	✓	
Running	✓	✓		✓	✓	✓	✓		
Cycling	✓	✓		✓	✓	✓	✓	✓	
Nordic Walking	✓	✓		✓	✓	✓	✓	✓	
Watching TV	✓								
Computer Work					✓	✓		✓	✓
Car Driving	✓								
Ascending Stairs*	✓	✓	✓	✓	✓	✓	✓	✓	
Descending Stairs*	✓	✓	✓	✓	✓	✓	✓	✓	
Vacuum Cleaning	✓	✓	✓	✓	✓	✓	✓	✓	
Ironing	✓	✓	✓	✓	✓	✓	✓	✓	
Folding Laundry	✓					✓		✓	✓
House Cleaning	✓				✓	✓		✓	✓
Playing Soccer								✓	✓
Rope Jumping**	✓	✓			✓	✓		✓	✓

Tabella 3.2: Attività svolte dai soggetti nel dataset PAMAP2
(*: 1 minuto, **: 2 minuti, -: 3 minuti)

I dati provenienti dal monitor e dai sensori IMU sono sincronizzati in 9 file ".dat", uno per ogni soggetto, strutturati in 54 colonne:

- **Timestamp** (1).
- **ID dell'attività** (2): Compreso tra 0 e 18 (0: attività transiorie).
- **Battito cardiaco** (3).
- **Sensore IMU sulla mano** (4-20).
- **Sensore IMU sul petto** (21-37).
- **Sensore IMU sull'anca** (38-54).

In particolare, le misurazioni provenienti da ogni sensore IMU sono suddivise in 17 colonne, nel seguente ordine:

- **Temperatura** (1).
- **Accelerometro 1** (2-4).
- **Accelerometro 2** (5-7).
- **Giroscopio** (8-10).
- **Magnetometro** (11-13).
- **Orientamento del dispositivo** (14-17, da ignorare).

Sono presenti valori nulli all'interno del dataset, indicati con "NaN", a causa dell'impiego di dispositivi wireless. Inoltre, il secondo accelerometro non è calibrato in modo preciso con il primo, per cui gli autori consigliano di usare solo quest'ultimo.

Valutando esclusivamente le quattro attività principali, escludendo il soggetto nove e rimuovendo le righe contenenti valori nulli, otteniamo 2285 sottosequenze dalla lunghezza di dieci secondi l'una (corrispondenti a 350 misurazioni). Le misurazioni del battito cardiaco, poiché campionate ad una frequenza più bassa, vengono ricampionate a 100 Hz, sostituendo i valori nulli con il rilevamento effettuato in precedenza. Per evitare il problema del *Sampling Bias*[5] [52], verrà applicato un ulteriore filtraggio, descritto nella Sezione 4.1.3.

3.4.4 *OPPORTUNITY*

Il dataset *Opportunity Activity Recognition* [38-40] è anch'esso una raccolta di misurazioni, provenienti da vari accelerometri, raccolte durante l'esecuzione di diverse attività da parte di quattro soggetti. I dati sono registrati all'interno di un ambiente appositamente predisposto e sia i soggetti che l'ambiente stesso sono dotati di vari sensori.

Tuttavia, poiché i sensori ambientali non fanno riferimento alle quattro attività svolte dai soggetti, bensì ai movimenti degli oggetti su cui sono collocati i sensori, sono stati considerati solo i dispositivi posti sui soggetti stessi (fig.3.7), ovvero 12 accelerometri e 7 IMU, i quali offrono le seguenti misurazioni:

- *3D acceleration*.

[5] Il *Sampling Bias* si verifica quando i dati in una ricerca statistica hanno probabilità diverse di essere campionati, non rappresentando accuratamente la reale situazione di ricerca.

- *3D rate of turn.*
- *3D magnetic field.*
- *Sensor's orientation.*

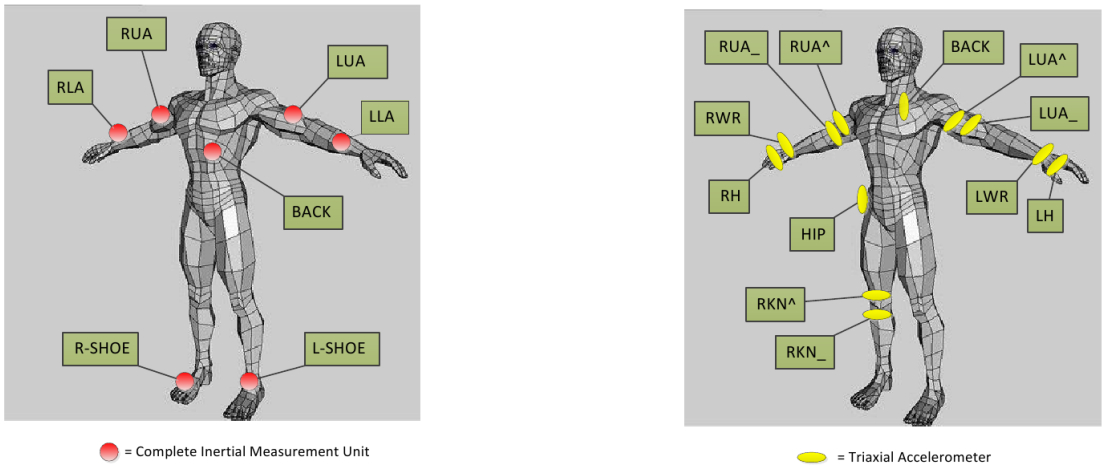


Figura 3.7: Ubicazione dei sensori su ogni soggetto. A sinistra, in rosso, è indicato il posizionamento dei 7 sensori IMU, mentre a destra è evidenziata in giallo la collocazione dei 12 accelerometri [38]

I campionamenti sono stati tutti effettuati alla frequenza di 30 Hz. Inoltre, poiché gli accelerometri sono wireless, è lecito aspettarsi una maggiore perdita di dati da parte di questi sensori.

Sono state registrate quattro attività motorie dei soggetti all'interno della stanza, ovvero *standing*, *walking*, *lying* e *sitting*. Più nello specifico, i partecipanti hanno svolto vari compiti nel modo più naturale possibile, attraverso sei "corse" (*run*), vale a dire sei sequenze di azioni da eseguire in risposta a determinate situazioni. Una delle corse, chiamata *drill-run*, consiste in venti ripetizioni di nove attività che coinvolgono specificatamente gli oggetti presenti all'interno della stanza, per cui ci concentreremo sulle rimanenti cinque, che consistono in nove semplici situazioni quotidiane dispiegate nel tempo:

- 1: Si parte da sdraiati, per poi alzarsi.
- 2: Muoversi per la stanza per controllare che tutti gli oggetti siano al loro posto.
- 3: Uscire fuori e camminare intorno all'edificio.

- 4: Preparare il caffè.
- 5: Sorseggiare il caffè camminando.
- 6: Preparare un panino.
- 7: Mangiare il panino.
- 8: Pulire la stanza e rimettere ogni cosa al suo posto.
- 9: Infine sdraiarsi di nuovo.

Vengono dunque campionate misurazioni relative ai movimenti dei soggetti, che comprendono sedersi, camminare, sdraiarsi e stare in piedi.

I dati sono divisi per *run* e per soggetti, i.e. le misurazioni di ogni soggetto sono suddivise in sei file ".dat": cinque *run* e la *drill-run*. Ogni file è organizzato in 250 colonne:

- **Timestamp** (1).
- **Accelerometri del soggetto** (2-37).
- **Sensori IMU del soggetto** (38-134).
- **Sensori degli oggetti** (135-243).
- **ID dell'attività svolta dal soggetto** (244). Può assumere i seguenti valori:
  - 0: Non definito (*null*).
  - 1: Soggetto in piedi (*stand*).
  - 2: Soggetto che cammina (*walk*).
  - 4: Soggetto seduto (*sit*).
  - 5: Soggetto sdraiato (*lie*).
- **ID delle attività concernenti l'ausilio degli oggetti presenti nella stanza** (245-250).

Sono presenti diversi valori nulli, per lo più nelle colonne 34-37 (accelerometro posto sulla mano destra). Inoltre, il soggetto quattro è stato scartato poiché, oltre a soffrire di numerosi dati nulli, non possiede sufficienti misurazioni per quanto riguarda l'attività con ID = 5 (*lie*).

Una volta escluso il soggetto quattro e rimosse le righe con all'interno valori nulli, si ottengono 4516 sequenze temporali lunghe tre secondi (corrispondenti a 75 misurazioni). I dati relativi all'attività *lie* (ID = 5) si presentano in numero inferiore, perciò nella Sezione 4.1.4 è descritta un'ulteriore elaborazione per evitare il *Sampling Bias* e diminuire il numero di sottosequenze utilizzate.

3.4.5 *ASCERTAIN*

Il dataset ASCERTAIN [41] è diviso in due parti: "ASCERTAIN_Features", contenente statistiche riguardo alle varie misurazioni e "ASCERTAIN_raw", che racchiude invece i dati veri e propri, che costituiscono il fulcro del dataset.

I dati derivano da segnali provenienti dall'elettrocardiogramma (ECG, 256 Hz), dall'elettroencefalogramma (EEG, 32 Hz), dall'attività elettrodermica (GSR, 128 Hz) e dai movimenti facciali di 58 soggetti. Tuttavia considereremo solo ECG, EEG e GSR, in quanto i movimenti facciali sono stati ripresi da una telecamera e non fanno parte dei segnali biometrici registrati con sensori.

Ogni soggetto è stato sottoposto alla visione di 36 clip video, dalla durata compresa tra 51 e 128 secondi, durante le quali sono stati registrati i segnali di cui sopra.

Terminati i campionamenti, ai partecipanti è stato chiesto di compilare un questionario, composto da 50 aggettivi (e.g. calmo, ansioso, superficiale), per valutare l'attinenza di quest'ultimi con la propria personalità. Successivamente, è stato calcolato un punteggio riguardante cinque caratteristiche particolari: estroversione, accondiscendenza, coscienza, tendenza ad emozionarsi e apertura mentale. Tuttavia questi punteggi sono costituiti da valori decimali compresi tra 1 e 7, rendendo il dataset più adatto per un *task* di regressione.

I soggetti hanno compilato ulteriori questionari, uno dopo la visione di ogni clip, indicando un valore da 0 a 6 per *arousal* (eccitazione) e un valore da -3 a 3 per *valence* (valenza). Si può allora classificare il quadrante di figura 3.3, in cui *arousal* e *valence* sono i due assi di un particolare modello emotivo bidimensionale e il centro del cerchio rappresenta una valenza neutra e un livello medio di eccitazione [53]:

- 0: $arousal > 3$ & $valence > 0$.
- 1: $arousal > 3$ & $valence \leq 0$.
- 2: $arousal \leq 3$ & $valence > 0$.
- 3: $arousal \leq 3$ & $valence \leq 0$.

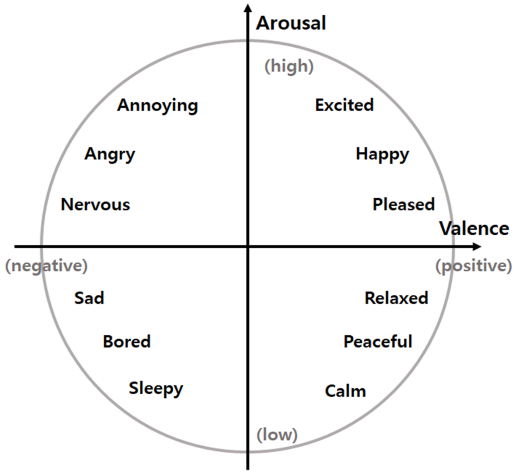


Figura 3.8: Le quattro combinazioni di *arousal* e *valence* [41]

Sono presenti tre cartelle relative ai segnali (ECG, EEG, GSR), con all'interno 58 ulteriori cartelle contenenti 36 file ".mat", che corrispondono alle misurazioni biometriche effettuate durante la visione delle clip. I questionari su arousal e valence sono invece racchiusi nel file "Dt_SelfReports.mat", sotto forma di due matrici 36×58.

I file di ECG, EEG e GSR sono organizzati in colonne nel seguente modo:

- **ECG**:
  - *Timestamp* (1).
  - Dati dell'accelerometro (2-4).
  - Dati ECG, mano destra (5).
  - Dati ECG, mano sinistra (6).
- **EEG**:
  - *Timestamp* (1).
  - Dati EEG (2-8).
- **GSR**:
  - *Timestamp* (1).
  - Dati dell'accelerometro (2-4).
  - Dati GSR (5).

I dati dei soggetti 1-36 sono stati registrati ad una minore frequenza, mentre quelli dei soggetti 44, 52, 56, 57 e 58 sono poco precisi, come dichiarato dagli autori stessi nel file "Data_Quality_Evaluation.xlsx". Inoltre, i dati ECG dei soggetti 1-8 includono esclusivamente le colonne 1, 5 e 6. Considerata anche l'imponente mole di misurazioni a disposizione, sono stati selezionati i 17 soggetti più precisi e campionati alla frequenza minore (37-55, con l'esclusione del 44 e del 52).

Utilizzando solo i dati provenienti dai 17 soggetti di cui sopra, rimuovendo le righe contenenti valori nulli ed effettuando il ricampionamento alla frequenza dell'elettrocardiogramma (32 Hz), abbiamo 39.329 sequenze temporali di 5 secondi (equivalenti a 160 misurazioni). Per ridurre questa quantità eccessiva verranno selezionate solo delle parti dei campioni biometrici, tenendo ben presente il problema del *Sampling Bias* (vedi sez. 4.1.5).

3.5 Model selection

Per ogni dataset è stata realizzata una *model selection*, attraverso la valutazione di vari iperparametri, separatamente per ciascun modello di apprendimento utilizzato. Ciascuno dei quattro modelli (ESN, SimpleRNN, LSTM e GRU) è addestrato su diverse combinazioni di iperparametri e, una volta trovata la combinazione migliore (i.e. quella che garantisce la *loss* minore sul validation-set), questa viene impiegata per il *fitting* e la successiva valutazione delle prestazioni. Infine, confrontiamo tra loro le performance sul test-set (sez. 4.2). I risultati sono basati sulla media dell'*accuracy* raggiunta in cinque iterazioni dell'addestramento. In pratica, si creano cinque istanze per ciascuno dei quattro tipi di modello RNN, ognuna delle quali viene addestrata sul medesimo training-set. Il punteggio finale è dato dalla media delle cinque percentuali di *accuracy* sul test-set.

Prima di essere addestrato, ogni modello dev'essere compilato, tramite la funzione `compile` di *Keras*. Questa prende tre argomenti: la *loss function* (per tutti `categorical_crossentropy`, da utilizzare in caso di due o più classi binarie da predire), l'*optimizer* e la *metrics* per giudicare la performance del modello (per tutti *accuracy*).

Ogni modello presenta un *layer* di output `Dense` (libreria *Keras*), del quale specifichiamo il numero di classi da predire, la funzione di attivazione e i parametri relativi alla regolarizzazione.

Allo scopo di rispettare le necessità hardware dei dispositivi indossabili su cui è basato questo lavoro, si è cercato di mantenere su livelli modesti le dimensioni dei dataset e la complessità dei modelli, provando ad ottenere il miglior compromesso tra complessità e affidabilità dei risultati.

3.5.1 *Selezione degli iperparametri*

Al fine di testare le varie combinazioni degli iperparametri, è stato impiegato `KerasClassifier`, un *wrapper* della libreria *Scikit-Learn*. La classe `KerasClassifier` prende un argomento `build_fn`, che corrisponde al nome della funzione da chiamare per ottenere il modello di RNN su cui vogliamo testare le varie combinazioni di iperparametri. Bisogna infatti indicare una funzione, da passare come argomento a `KerasClassifier`, che crei, compili e ritorni il modello in questione. Sono state quindi definite quattro funzioni, una per ogni modello, che creano, compilano e ritornano, rispettivamente, i modelli ESN, SimpleRNN, LSTM e GRU.

Una volta ritornato il modello, realizziamo la distribuzione degli iperparametri. Assegniamo un *range* di valori ai parametri che intendiamo testare, per poi aggiungerli ad un dizionario `param_distr` che ha come chiavi i nomi degli iperparametri. Tale dizionario costituisce il primo argomento da passare a `ParameterSampler` della libreria *Scikit-Learn*, in grado di generare combinazioni casuali di iperparametri dalle distribuzioni di quest'ultimi. Questo generatore prende, come secondo argomento, il numero di iterazioni, i.e. il numero totale di combinazioni da testare. Per evitare un numero troppo alto di combinazioni, gli iperparametri sono stati suddivisi in parti. Vengono dunque testate prima le combinazioni degli iperparametri specifici per ESN, poi quelle relative all'ottimizzazione di tutti i modelli, dunque quelle riguardanti la regolarizzazione e infine il numero di unità e *layer*. Al fine di assicurare una condizione di parità nell'addestramento dei diversi modelli, si è cercato di mantenere il solito numero di *trainable parameters* per ognuno di questi, modificando il numero di unità di conseguenza. Ad un maggior numero di unità del modello ESN, ad esempio, ne corrisponde uno minore del modello LSTM, ma il numero esplorato di *trainable parameters* è circa lo stesso per entrambi.

Gli iperparametri vengono testati all'interno del seguente ciclo:

```
for g in ParameterSampler(param_distr, n_iter)
```

Dopo aver settato la combinazione corrente di iperparametri tramite il metodo `set_params`, il modello viene addestrato sul training-set (`fit`) e valutato sul validation-set (`score`). Se lo *score* ottenuto è migliore dei precedenti (in termini di *accuracy*), viene salvato nella variabile `best_score` e i relativi valori degli iperparametri sono inseriti nella lista `best_params`. A questo punto il ciclo ricomincia con la successiva combinazione di iperparametri. Una volta finito, i valori migliori vengono stampati e verranno successivamente impiegati per l'addestramento del modello a cui si riferiscono.

Quello appena descritto (mediante *random search* [54]) rappresenta lo schema generale di selezione degli iperparametri.

Iperparametri ESN

Inizialmente viene effettuata la ricerca dei valori più efficaci per tre iperparametri specifici del modello ESN. Questi, descritti nella Sezione 2.2.4, sono:

- *input_scaling* $\in [0.5, 2.0]$
- *leaky* $\in [0.3, 1.0]$
- *spectral_radius* $\in [0.5, 1.5]$

Vengono generate, all'interno dei *range* prefissati, cento combinazioni diverse di tali iperparametri, in modo casuale. La combinazione ottima è poi impiegata per il training del modello ESN.

Ottimizzazione

Adoperiamo, per ogni modello, l'ottimizzazione *Adam*, un metodo di discesa del gradiente stocastico, basato sulla stima del primo e del secondo momento del gradiente, "computazionalmente efficiente, con pochi requisiti di memoria [...] e adatto per problemi con un gran numero di dati/parametri" [55]. Il momento è un termine che, sommato al gradiente, concorre nell'aggiornamento dei parametri e dipende dall'iterazione precedente. *Adam*

adatta l'apprendimento sulla base del primo momento medio (la media mobile dei gradienti) e della media dei secondi momenti (*uncentered variance*, la media mobile del quadrato dei gradienti). *Adam* è una classe di `Keras.optimizer` e presenta tre argomenti su cui effettuiamo una ricerca per trovare i valori più efficienti:

- *learning_rate* $\in [0.001, 0.01]$: Rappresenta la velocità alla quale il modello "impara", i.e. stabilisce di quanto i pesi debbano essere aggiornati ad ogni iterazione.
- *beta_1* $\in [0.8, 0.99]$: Controlla il tasso di decadimento della media mobile del primo momento.
- *beta_2* $\in [0.98, 0.999]$: Controlla il tasso di decadimento della media mobile del secondo momento.

Anche in questo caso generiamo cento combinazioni diverse di iperparametri, ognuno di questi all'interno del *range* sopra indicato. Ottenuta la combinazione migliore, sostituiamo i valori ai rispettivi parametri di *Adam* e compiliamo il modello con l'ottimizzatore risultante.

Funzione di attivazione

Come funzione di attivazione per lo strato di output (`Dense`) di ogni modello utilizziamo *softmax* [56], una funzione esponenziale normalizzata che trasforma un vettore x di valori reali arbitrari e dimensione N in un vettore $\sigma(x)$, anch'esso di dimensione N, di valori compresi nell'intervallo (0, 1) e la cui somma è 1: $\sigma(\vec{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$. Questa funzione risulta indicata per problemi di classificazione multi-classe, assegnando una determinata probabilità ad ognuna di queste. Quindi viene scelta la classe con valore di probabilità maggiore.

Regolarizzazione

Per ogni modello aggiungiamo la regolarizzazione [57] relativa al *layer* `Dense` di output, al fine di mitigare problemi di *overfitting*. Adoperiamo la funzione `l1_l2` di `Keras.regularizers`, che applica una combinazione di due differenti penalità alla *loss function*:

- $f_{\text{loss}} = f_{\text{loss}} + \lambda \sum_{x=0}^{N} |W_{\text{x}}|$ Regolarizzazione *l1*
- $f_{\text{loss}} = f_{\text{loss}} + \lambda \sum_{x=0}^{N} {W_{\text{x}}}^2$ Regolarizzazione *l2*

l1 e *l2* aggiungono, rispettivamente, il valore assouluto e il quadrato dei pesi W_x alla funzione *loss*. λ rappresenta l'iperparametro su cui effettuare la ricerca del valore più efficiente, il fattore di regolarizzazione. Mediante `l1_l2`, *Keras* applica entrambe le regolarizzazioni, sommandole alla *loss function*. Applichiamo dunque la regolarizzazione `l1_l2` allo strato di output, considerando i tre argomenti *regularizer* di `Dense`:

- *kernel_regularizer* $\in [e^{-6}, e^{-5}, e^{-4}]$
- *bias_regularizer* $\in [e^{-6}, e^{-5}, e^{-4}]$
- *activity_regularizer* $\in [e^{-6}, e^{-5}, e^{-4}]$

Vengono così generate, per ogni modello, ventisette combinazioni di iperparametri relativi alla regolarizzazione.

Unità e layer

Per quanto riguarda il numero di unità, l'obiettivo è un compromesso ottimo tra complessità e accuratezza dei modelli. In questo caso è stata data priorità al modello ESN *single-layer*, ricercando il numero di unità in un intervallo compreso tra 500 e 1500. Il principio seguito per questa selezione è relativo all'efficienza: se, partendo dal minimo (500) e aumentando il numero di unità, l'*accuracy* sul validation-set incrementa sensibilmente, allora proseguiamo la ricerca col numero di unità successivo. Viceversa, se ad un aumento di tale numero non corrisponde un significativo miglioramento dell'*accuracy* sul validation-set, la ricerca si ferma al numero di unità precedente.

Il totale di unità ESN impiegate definisce anche un determinato numero esplorato di *trainable parameters*[6]. Sulla base di questo "adattiamo" gli altri modelli, assegnando ad ognuno di essi un numero di unità tale da avvicinarsi il più possibile alla stessa quantità di parametri addestrabili. Un procedimento equivalente viene ripetuto anche per il caso

[6] Parametri della rete soggetti all'addestramento.

multi-layer: ogni modello (compreso ESN) presenta un numero di unità regolato dalla quantità di *trainable parameters* esplorati dal modello ESN nel caso *single-layer*.

Per trovare il miglior numero di *hidden layer*, tenendo conto della necessità di non incrementare eccessivamente la complessità di ogni modello, consideriamo [1-5] come intervallo di ricerca e testiamo ogni configurazione sul validation-set. Il numero di *layer* che garantisce le prestazioni maggiori viene usato per il training finale del modello.

Per una visione più completa, nella Sezione 4.2 sono riportati i risultati degli esperimenti sia per il caso *single-layer*, sia per quello *multi-layer*.

Epoche

Il numero di epoche di *fitting* viene deciso in base al momento in cui ogni modello finisce di apprendere e va in *overfitting*. Se la *loss* relativa al validation-set inizia a salire in maniera continua dopo un certo numero di epoche o rimane stabile, l'addestramento si conclude. Dal punto di vista del codice, questo metodo viene implementato tramite `EarlyStopping` [58], *callback* della libreria *Keras* da passare come argomento al metodo `fit`. Questa termina l'addestramento qualora un parametro specificato (in questo caso `val_loss`) smetta di migliorare. E' possibile specificare anche due ulteriori parametri all'interno della definizione della *callback*: `patience` e `restore_best_weights`. Il primo è un intero che rappresenta il numero di epoche senza miglioramenti prima che venga fermato l'addestramento ed è utile in quanto possono verificarsi delle oscillazioni della *loss* sul validation-set, dopo le quali essa riprende a migliorare. Il secondo argomento invece, settato a `True`, ripristina i pesi relativi all'epoca con il miglior valore del parametro monitorato.

Caching ESN

Generalmente, l'output delle reti neurali ricorrenti viene calcolato nel seguente modo:

$$stati = RNN(input)$$
$$output = OutputLayer(stati)$$

Le matrici dei pesi all'interno della RNN sono modificate ad ogni epoca, perciò è necessario ricalcolare tutti gli stati associati ad ogni sequenza di input.

Per quanto riguarda ESN invece, gli stati vengono calcolati solo all'inizio, in quanto non viene addestrato il *reservoir*, ma esclusivamente il *readout*. Ricalcolare gli stati ad ogni epoca è quindi inutile, motivo per il quale, una volta inizializzati all'interno del *reservoir*, essi vengono salvati e riutilizzati per tutte le epoche successive. Il tempo di addestramento del modello ESN, dovendo dunque addestrare solo il *readout*, diminuisce drasticamente.

4

RISULTATI

In questo capitolo vengono delineate le tecniche di *pre-processing* impiegate in questo lavoro, confrontandole con diversi studi in letteratura, per poi andare ad illustrare e commentare, con l'ausilio di grafici e tabelle, i risultati raggiunti. La Sezione 4.1 è dedicata al *pre-processing*, con una sottosezione specifica per ogni dataset. In ciascuna sottosezione definiamo le modalità di elaborazione e di suddivisione di un determinato dataset. L'ultima parte della Sezione 4.1 è invece relativa alle tecniche di *pre-processing* adoperate e ai risultati raggiunti in letteratura per ogni dataset. Infine, in Sezione 4.2, sono rappresentate tabelle e grafici riguardanti i risultati ottenuti. L'esito degli esperimenti viene perciò commentato, discusso e confrontato con le prestazioni raggiunte dagli altri modelli di apprendimento non-RNN.

4.1 Pre-processing dei dati

Nonostante la sostanziale differenza tra i vari dataset, essi condividono la possibilità di essere suddivisi in sequenze temporali, al fine di poter addestrare i nostri modelli di RNN. In generale, l'input *shape* di ogni dataset è definita da una tripla (Numero di sottosequenze × Lunghezza delle sottosequenze × Numero di *features*[1]), mentre l'output *shape* è data da (Numero di sottosequenze × Numero di classi). I dataset si differenziano quindi per quanto riguarda la tipologia dei dati, la frequenza delle misurazioni e la struttura interna, ma il fulcro del *pre-processing* rimane la suddivisione in sottosequenze temporali. Per questo sussistono numerosi elementi in comune tra le elaborazioni dei cinque dataset.

4.1.1 *WESAD*

I dati dei soggetti, salvati in quindici file ".pkl", vengono inizialmente caricati in un dizionario attraverso la funzione `load` della libreria *Pickle*. Successivamente, viene effettuato il *resampling* dei dati alla frequenza di 32 Hz, mediante la funzione `resample` della libreria *SciPy*. Parallelamente le *features*, ovvero le misurazioni biometriche provenienti dai vari dispositivi, vengono unite in un'unica matrice attraverso `concatenate`, della libreria *NumPy*. In pratica, ogni chiave del dizionario (i.e. ogni soggetto) ha, come valore, una matrice di dati N×F, con N = numero di misurazioni e F = numero di *features* (elettrocardiogramma, accelerometro, ecc.).

I dati dei soggetti vengono poi uniti in un'unica matrice X N×F (N = numero di misurazioni totali, F = numero di *features*) e standardizzati[2], mediante la formula

$$X' = \frac{X - \mu}{\rho} ,$$

dove μ = media dei valori delle *features* e ρ = deviazione standard dei valori delle *features*.

Successivamente si crea un vettore Y che ha come elementi gli ID delle attività, sincronizzato con la matrice di input X, per cui alla riga X[0] corrisponde l'ID dell'attività svolta

[1] Numero di misurazioni per ogni sottosequenza temporale (e.g. accelerometro, elettrocardiogramma).

[2] In questo caso specifico, aggiungendo la standardizzazione„i risultati finali migliorano almeno del 10%.

per prima. Inoltre, poiché gli ID 0, 5, 6 e 7 non sono da considerare, vengono rimossi dal vettore Y, insieme alle corrispondenti righe della matrice X.

A questo punto Y è costituito da varie sequenze di interi compresi tra uno e quattro, mentre X contiene le misurazioni associate alle attività (sincronizzate con Y, fig. 4.1).

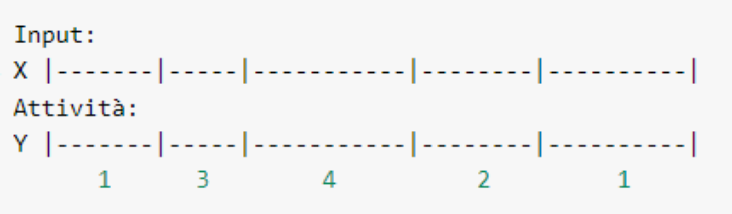


Figura 4.1: Sottosequenze temporali di X e Y sincronizzate

Quindi si calcola la lunghezza di ogni segmento e la si inserisce in una lista, chiamata `LenSubsequences`, che servirà per suddividere i dati in diverse serie di misurazioni, ognuna con associata la propria attività. Infatti, basandosi sulle lunghezze dei segmenti salvate in `LenSubsequences`, possiamo scandire, mediante un `for`, la matrice X e il vettore Y.

Per evitare il problema del *Sampling Bias* e per diminuire il numero di dati da elaborare, prendiamo un prefisso per ogni segmento, i.e. sedici sottosequenze da cento misurazioni (corrispondenti a circa tre secondi) l'una, che vengono inserite in una lista `SubsequencesX`. Le diverse attività, nonostante siano eseguite in tempi diversi, si presentano in egual numero, salvo *meditation*, che compare due volte per ogni soggetto (dopo le sessioni di *amusement* e *stress*). Per questo, se l'ID dell'attività è uguale a 4, prendiamo otto sottosequenze, invece di sedici.

Per ogni serie di misurazioni (i.e. ogni cento campioni di dati), salviamo in una lista `SubsequencesY` anche il numero di attività relativo, prendendolo dal vettore Y.

`SubsequencesX` e `SubsequencesY` sono dunque due liste della stessa lunghezza, uguale al numero di sottosequenze scelte. La prima è composta da matrici di dimensione 100×14 (numero di misurazioni × numero di *features*), la seconda contiene gli interi relativi alle attività svolte.

Tramite la funzione `to_categorical` di *Keras*, Y viene convertita in una matrice binaria con tante classi (colonne) quante sono le attività (in questo caso quattro, fig. 4.2).

```
   Y          to_categorical(Y, num_classes = 4)

[ 1.0                  [ 1   0   0   0
  ...                   ... ... ... ...
  1.0                    1   0   0   0
  2.0                    0   1   0   0
  ...                   ... ... ... ...
  2.0                    0   1   0   0
  4.0                    0   0   0   1
  ...                   ... ... ... ...
  4.0                    0   0   0   1
  3.0                    0   0   1   0
  ...                   ... ... ... ...
  3.0 ]                  0   0   1   0 ]
```

Figura 4.2: Applicazione di `to_categorical` a Y

La *shape* finale di X è dunque (Numero di sottosequenze: 952 × Lunghezza delle sottosequenze: 100 × Numero di *features*: 14), mentre quella di Y è (Numero di sottosequenze: 952 × Numero di classi: 4).

Infine i dati vengono divisi tra training-set, validation-set e test-set, attraverso la funzione `train_test_split` della libreria *Scikit-Learn*, nel seguente modo:

- 771 (81%) ⇒ Training-set.
- 86 (9%) ⇒ Validation-set.
- 95 (10%) ⇒ Test-set.

4.1.2 *HAR*

Il dataset si compone di quattro file ".csv", rispettivamente "Phones_accelerometer", "Phones_gyroscope", "Watch_accelerometer" e "Watch_gyroscope", caricati in memoria mediante la funzione `read_csv` della libreria *Pandas*. A differenza degli altri dataset, le

attività svolte sono sei, relative alle misurazioni effettuate tramite accelerometro e giroscopio di smartphone e smartwatch. Come spiegato nella Sezione 3.4.2, i soggetti sono nove, ma gli ultimi due possiedono un minor numero di campionamenti, motivo per il quale non sono stati presi in considerazione.

La divisione del dataset è stata effettuata considerando sei soggetti e assegnando un'attività ad ognuno di questi. Tale suddivisione ha permesso di incrementare la precisione sul test-set del 15/20%. Più nello specifico, lo *splitting* è stato eseguito nel seguente modo:

- **S1**: *stairsup* (ID = 4)
- **S2**: *stairsdown* (ID = 5)
- **S3**: *stand* (ID = 1)
- **S4**: *sit* (ID = 2)
- **S5**: *walk* (ID = 3)
- **S6**: *bike* (ID = 6)

Le attività sono state assegnate seguendo il criterio della quantità di dati a disposizione (ad esempio, il soggetto 1 possiede molti più dati riguardanti l'attività 4 rispetto agli altri).

Una volta caricati i soggetti, si effettua il *resampling* dei dati (`resample` di *SciPy*) alla frequenza dell'accelerometro dello smartwatch (circa 70 Hz), che rappresenta un ragionevole compromesso tra la frequenza dello smartphone (200 Hz) e la frequenza del giroscopio dello smartwatch (50 Hz). Parallelamente, i dati dei quattro sensori vengono concatenati verticalmente, andando a costituire una matrice, formata da tante colonne quante sono le *features* (in tutto dodici) e tante righe quante sono le misurazioni del soggetto a cui si riferisce.

Poiché le attività sono scritte non in forma di intero, bensì come stringhe, è necessario effettuare una scansione del dataset per trasformare i nomi in numeri (e.g. *stand* = 1).

Successivamente possiamo unire i dati dei soggetti, così come è stato fatto per il dataset WESAD, in una matrice X N×F (N = numero di misurazioni totali, F = numero di *features*), per poi standardizzarli. Vengono dunque estratti gli ID delle attività e salvati in un vettore Y, di lunghezza uguale al numero di righe di X (ad ogni riga contentente le misurazioni corrisponde un'attività svolta).

Una volta rimosso il target 0 (i.e. il numero di attività non definito) e le righe corrispondenti di X, possiamo dividere i dati in sottosequenze temporali, ognuna delle quali si compone di cento misurazioni (con il relativo target), equivalenti a circa due secondi. Per non aumentare eccessivamente la mole di dati a disposizione ed evitare il *Sampling Bias*, prendiamo 16.000 campioni per ogni attività. Ognuno dei sei soggetti contribuisce quindi per 160 sottosequenze. E' possibile fare questo attraverso una semplice scansione `for` della matrice X, durante la quale vengono salvate 160 sottosequenze (con relativa attività) ogni volta che incontriamo un nuovo soggetto, per un totale di 160×6 = 960.

Y viene quindi convertita in una matrice binaria (`to_categorical`), di dimensione (Numero di sottosequenze: 960 × Numero di classi: 6), mentre X è una tripla (Numero di sottosequenze: 960 × Lunghezza delle sottosequenze: 100 × Numero di *features*: 12)

Le 960 sottosequenze sono infine "smistate" in tre diverse liste nel seguente modo:

- 768 (80%) ⇒ Training-set.
- 96 (10%) ⇒ Validation-set.
- 96 (10%) ⇒ Test-set.

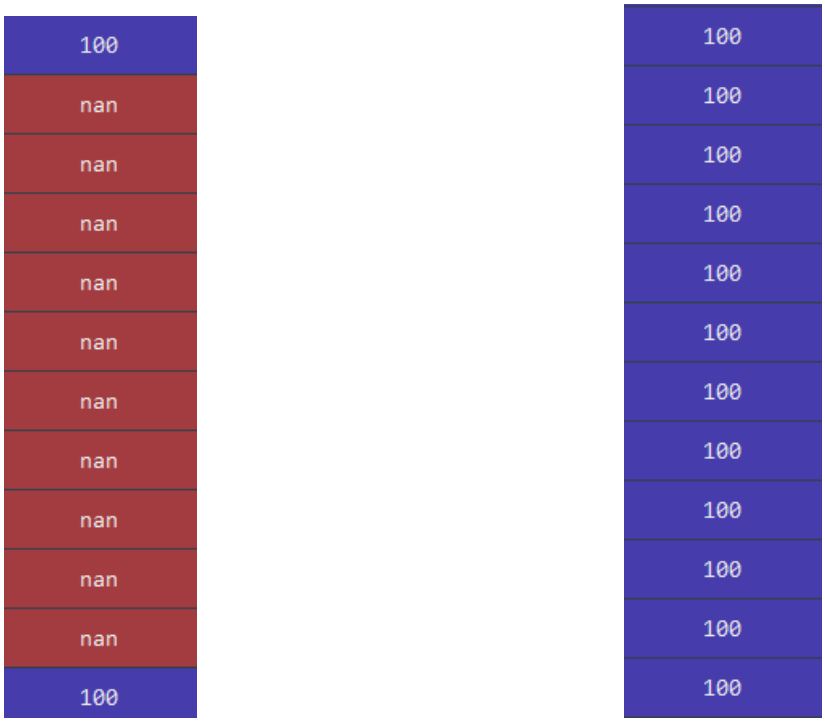
4.1.3 *PAMAP2*

Sono presenti nove file ".dat", ognuno di essi riguardante un soggetto. Tuttavia, solo i primi otto sono stati presi in considerazione, in quanto l'ultimo possiede pochissimi dati (vedi sez. 3.4.3). I soggetti vengono caricati in un dizionario, che ha come chiavi gli identificativi dei partecipanti ("S1"-"S8") e come valori le misurazioni biometriche effettuate.

Inizialmente si esegue un filtraggio dei dati:

- Si rimuovono i dati relativi alle attività diverse da *lying* (ID = 1), *sitting* (ID = 2), *standing* (ID = 3) e *walking* (ID = 4).
- Si rimuovono le *features* riguardanti il secondo accelerometro (difettoso) e l'orientamento del dispositivo (poco utile ai fini della ricerca).
- La frequenza alla quale è stato campionato il battito cardiaco dei soggetti è assai inferiore rispetto alla frequenza degli altri sensori utilizzati. Nonostante ciò, questo tipo di dati è stato comunque sincronizzato temporalmente con gli altri dispositivi, lasciando tuttavia

numerosi spazi vuoti all'interno della colonna in questione, riempiti da valori "NaN". Abbiamo quindi un valore non nullo ogni circa dieci righe di misurazioni. Per tale motivo, le righe nulle del battito cardiaco sono sostituite dal valore precedente non nullo, come si può vedere nella figura 4.3.



(a) prima (b) dopo

Figura 4.3: Colonna relativa al battito cardiaco

- Sono presenti alcune (poche) righe contenenti valori "NaN", a causa della natura wireless dei dispositivi adoperati. Esse sono state rimosse, senza tuttavia impattare sulla sincronizzazione temporale dei dati, per via del numero molto esiguo (circa un migliaio a fronte di più di 800.000 campioni).

La frequenza dei vari campionamenti è ora uguale per tutti i sensori e dovremmo aspettarci 100.000 misurazioni per ogni soggetto. Tuttavia, i dati sono sempre di poco inferiori o superiori a 100.000 (± 5000 al massimo), motivo per il quale è stato effettuato

un *resampling* di ogni soggetto a 100.000 righe (ogni riga corrisponde ad una misurazione).

Successivamente i dati di ogni soggetto sono concatenati orizzonatalmente in una matrice X di dimensione 800.000×31 (misurazioni totali × numero di *features*), per poi essere standardizzati. Allo stesso modo viene creato un vettore Y contenente gli 800.000 ID delle attività, associati alle righe di X. Le trentuno colonne di X sono formate da:

- **Battito cardiaco** (1).
- **Temperatura di anca, mano e petto** (3)
- **Accelerometro su anca, mano e petto** ($xyz \times 3$)
- **Giroscopio su anca, mano e petto** ($xyz \times 3$)
- **Magnetometro su anca, mano e petto** ($xyz \times 3$)

A questo punto i dati (racchiusi in X e Y) devono essere suddivisi in sottosequenze temporali. Per fare ciò, bisogna scorrere il vettore Y per determinare la lunghezza dei segmenti contenenti valori (ID) uguali. Un segmento inizia e finisce quando il numero (ID) dell'attività cambia (fig. 4.1). Applichiamo dunque lo stesso metodo usato per il dataset WESAD, inserendo le lunghezze nella lista `LenSubsequences` e utilizzandole per prendere, da ogni segmento, un prefisso corrispondente a trenta sottosequenze temporali di dieci secondi (circa 350 misurazioni) l'una. Inoltre, i target (ID 1, 2, 3, 4) sono presenti in egual numero, per cui ricavando un prefisso da ogni segmento siamo sicuri di non incorrere in un eventuale problema di *Sampling Bias*. Naturalmente ogni sottosequenza estratta da X è collegata alla sua attività, ricavata da Y.

Le matrici X e Y hanno ora il medesimo numero di righe, equivalente al numero di sottosequenze scelte. Dopo averla convertita in una matrice binaria costituita da quattro classi (funzione `to_categorical`), Y assume la seguente forma: (Numero di sottosequenze: 930 × Numero di classi: 4). X, invece, è una matrice tridimensionale di dimensione (Numero di sottosequenze: 930 × Lunghezza delle sottosequenze: 350 × Numero di *features*: 31).

Infine suddividiamo i dati tramite `train_test_split` di *Scikit-Learn*:

- 753 (81%) ⇒ Training-set.
- 84 (9%) ⇒ Validation-set.
- 93 (10%) ⇒ Test-set.

4.1.4 *OPPORTUNITY*

Mediante `load_txt` e `concatenate` di *NumPy*, i dati delle cinque *run* (vedi sez.3.4.4) vengono rispettivamente caricati e uniti tra loro (i.e. concatenati orizzontalmente, così che le corse di ogni soggetto siano contenute in un'unica matrice), andando a costituire i valori di un dizionario (`ds`) che ha come chiavi i nomi dei tre soggetti considerati ("S1", "S2", "S3"). Dopo di che vengono estratti gli ID delle attività dalla colonna 244 delle matrici contenenti i campioni biometrici, collocandoli come valori in un dizionario a parte (`labels`), che ha anch'esso i nomi dei soggetti come chiavi.

Si esegue poi un primo filtraggio dei dati, andando ad eliminare le righe contenenti valori nulli (e i target ad esse associati) e le colonne delle matrici che rappresentano *features* da ignorare. Alla fine, avremo un totale di 130 *features*, 33 di esse relative a undici accelerometri e le restanti 97 provenienti da sette sensori IMU.

Vengono poi uniti tutti i dati dei soggetti, formando un'unica matrice X di dimensione N×F (N = numero di misurazioni totali, F = numero di *features*) e un vettore di interi Y contenente gli ID delle attività (lungo N = numero di misurazioni totali).

La frequenza alla quale sono stati eseguiti i campionamenti è la stessa per ogni sensore (30 Hz), per cui non c'è bisogno di effettuare un *resampling*.

Poiché gli ID delle attività sono rappresentati dagli interi 1, 2, 4 e 5, gli ID 4 e 5 sono stati cambiati in 3 e 4, al fine di semplificare il lavoro successivo. Inoltre, i dati riguardanti l'ID 0 (i.e. attività non definita) sono stati rimossi.

Seguendo lo stesso procedimento impiegato durante il *pre-processing* dei dataset WESAD e PAMAP2, la divisione in sottosequenze avviene salvando nella lista `LenSubsequences` le lunghezze dei segmenti con gli stessi ID. X viene dunque diviso in sottosequenze di 75 misurazioni (corrispondenti a circa tre secondi), ognuna con la propria attività associata.

Sono tuttavia presenti 1320 misurazioni per l'attività 1, 600 per la 2, 840 per la 3 e sole 120 per la 4. Per evitare un rilevante problema di *Sampling Bias*, viene applicata una selezione per quanto riguarda le prime tre attività. Più nello specifico, da ognuna di esse vengono prese esclusivamente 120 misurazioni, così da avere un numero bilanciato di dati per ciascun target. Questo viene fatto estraendo una riga ogni undici per quanto concerne la

classe 1, una ogni cinque per la 2 e una ogni sette per la 3. I dati della 4 vengono ovviamente estratti tutti.

Avendo quindi 120 righe per ciascuna delle quattro attività, si giunge ad un totale di 120×4 = 480 sottosequenze da 75 campionamenti ciascuna. La *shape* finale di X sarà quindi (Numero di sottosequenze: 480 × Lunghezza delle sottosequenze: 75 × Numero di *features*: 130).

Y, invece, una volta trasformata mediante `to_categorical`, è una matrice binaria con lo stesso numero di righe di X (Numero di sottosequenze: 480 × Numero di classi: 4).

I dati vengono infine suddivisi, mediante `train_test_split` di *Scikit-Learn*, tra training-set, validation-set e test-set:

- 389 (81%) ⇒ Training-set.
- 43 (9%) ⇒ Validation-set.
- 48 (10%) ⇒ Test-set.

4.1.5 *ASCERTAIN*

I dati sono separati in tre cartelle, relative ai tre differenti sensori utilizzati: EEG (Elettroencefalogramma), ECG (Elettrocardiogramma) e GSR (*Galvanic Skin Response*). Ogni cartella presenta al suo interno tante altre cartelle quanti sono i soggetti. Ciascuna di queste contiene le misurazioni riguardanti la visione di trentasei clip, organizzate in matrici N×F (N = Numero di misurazioni, F = Numero di *features*).

I dati di *arousal* e *valence* sono invece racchiusi nel file "Dt_SelfReports.mat" e si presentano come due matrici 17×36, le cui righe indicano il soggetto e le colonne il numero della clip. Ciascuna cella contiene un numero rappresentante la valutazione di ogni singola clip da parte del soggetto. In particolare, le valutazioni vanno da -3 a 3 per quanto riguarda *valence* e da 0 a 6 per quanto invece concerne *arousal*. I target sono dunque determinati dalla combinazione delle due condizioni:

- 0: *arousal* > 3 & *valence* > 0.
- 1: *arousal* > 3 & *valence* ≤ 0.
- 2: *arousal* ≤ 3 & *valence* > 0.

- 3: *arousal* ≤ 3 & *valence* ≤ 0.

e salvati in una matrice Y, di dimensione 17×36.

A questo punto vengono rimosse le righe contenenti valori "NaN" da ogni matrice di dati EEG, ECG e GSR.

Dopo aver effettuato un *resampling* di ECG (256 Hz) e GSR (128 Hz) alla frequenza dell'elettroencefalogramma (EEG, 32 Hz), tramite `resample` di *SciPy*, le *features* (i.e. le colonne) di ogni clip, provenienti dai tre sensori, vengono concatenate lungo l'asse *y*, formando tante matrici N×F (N = Numero di misurazioni della clip in questione, F = Numero totale di *features*) quante sono le clip, ovvero trentasei. Ciascuna di queste matrici va a comporre il valore di un dizionario che ha come chiavi i nomi delle trentasei clip ("Clip1"-"Clip36"). Quest'ultimo costituisce il valore di un ulteriore dizionario, le cui chiavi sono i nomi dei soggetti ("S1"-"S17"). Ogni soggetto è dunque la chiave di un dizionario che ha come valori trentasei ulteriori dizionari relativi ai campionamenti effettuati durante la visione delle clip (fig 4.4). Le *features* totali (i.e. EEG+ECG+GSR) sono 17.

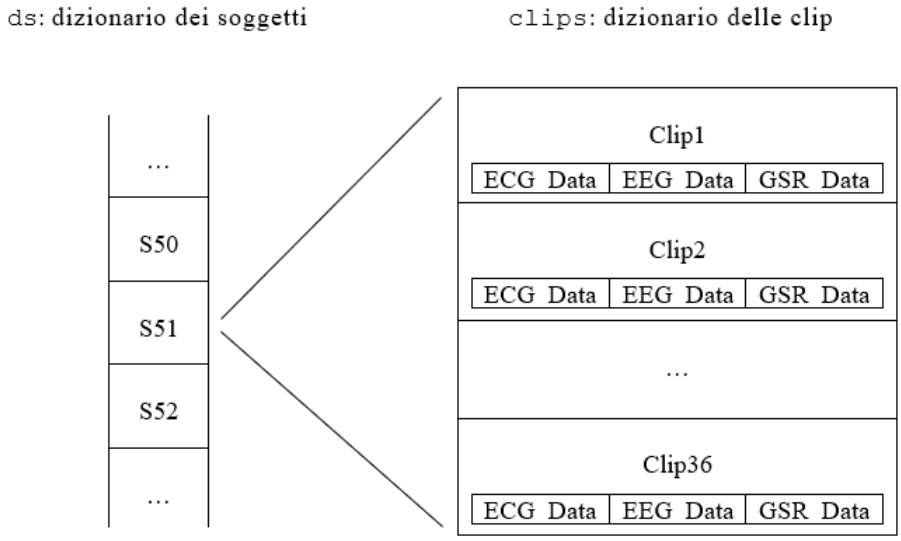


Figura 4.4: Organizzazione delle misurazioni biometriche del dataset ASCERTAIN

Adesso possiamo suddividere il dataset in sottosequenze temporali di cinque secondi ciascuna (corrispondenti a circa 160 misurazioni). Per ridurre la mole eccessiva di dati a disposizione, è necessario effettuare una selezione "equilibrata". Scegliamo dunque le ultime dieci sottosequenze per ogni clip (il soggetto risulta più coinvolto verso la fine dei filmati), bilanciando il numero dei target per evitare il problema del *Sampling Bias*. Più specificatamente, il numero di clip che hanno un target equivalente a 2 (in tutto 84) è inferiore agli altri. Per questo motivo, scegliamo 84 clip per ogni classe, per un totale di 84×4 = 336 clip, che corrispondono a 336×10 = 3360 sottosequenze da 160 misurazioni ciascuna. Ogni sottosequenza viene salvata in un vettore X, che alla fine avrà dimensione (Numero di sottosequenze: 3360 × Lunghezza delle sottosequenze: 160 × Numero di *features*: 17). I target, uno per ogni sottosequenza presa in esame, vengono invece salvati in un vettore Y, poi convertito, tramite la funzione `to_categorical` di *Keras*, in una matrice binaria con quattro classi. La dimensione finale di Y è quindi (Numero di sottosequenze: 3360 × Numero di classi: 4).

Allo stesso modo dei dataset WESAD, PAMAP2 e OPPORTUNITY, le sottosequenze vengono infine distribuite, mediante `train_test_split`, tra training-set, validation-set e test-set:

- 2722 (81%) ⇒ Training-set.
- 302 (9%) ⇒ Validation-set.
- 336 (10%) ⇒ Test-set.

4.1.6 *Altri studi*

Nonostante l'organizzazione dei dati in sottosequenze, comune tra i vari studi, sussistono numerose differenze nel *pre-processing* dei cinque dataset, ad esempio relative alla lunghezza delle sottosequenze e all'elaborazione delle *features*.

WESAD

In "The Effect of Person Specific Biometrics in Improving Generic Stress Predictive Models" [32] viene effettuato un *downsampling* dei dati in modo completamente diverso. Vengono infatti scartate molte misurazioni in modo da bilanciare le classi, senza usare alcuna

funzione di *resampling*. Inoltre, le classi *amusement* e *meditation* non sono considerate, riducendo il problema alla situazione binaria "stress/no-stress". Gli autori non utilizzano tutte le *features* disponibili, ma applicano varie trasformazioni ai valori biometrici provenienti dall'elettrodermografo (EDA) e dall'HRV (*Heart Rate Variability*), quest'ultima derivata dai dati dell'elettrocardiogramma. Il totale finale di campioni è 102.388, ognuno composto da 95 *features*. La struttura temporale permane, ma in questo caso è stata adottata una finestra mobile di cinque minuti per HRV e dieci minuti per EDA. In pratica, i dati vengono processati cinque/dieci minuti alla volta, mediante *shift* della finestra temporale. I modelli testati sono addestrati mediante *10-Folds-Cross-Validation*, utilizzando l'algoritmo *Random Forest*. Relativamente al caso binario, l'*accuracy* massima raggiunta dagli autori si aggira tra il 90 e il 95%.

In un altro studio [33] viene operata una suddivisione temporale diversa, applicando una finestra mobile di un minuto con *shift* di 0,25 secondi, eccezion fatta per i dati provenienti dall'accelerometro (ACC), per il quale la finestra è di cinque secondi. Vengono applicate varie elaborazioni alle *features*, quali il calcolo di media e deviazione standard per ACC, l'individuazione dei picchi relativi all'elettrocardiogramma (ECG) e il calcolo dei massimi e dei minimi. Il totale di finestre temporali generate, assumendo 0,25 secondi come *shift* temporale, è di circa 133.000, il 53% delle quali relative alla condizione *baseline*, il 30% a *stress* e il restante 17% ad *amusement*. I modelli scelti per i due *task* di classificazione (binario e con tre classi) sono *Decision Tree*, *Random Forest*, *Ada-Boost Decision Tree* (AB-DT), *Linear Discriminant Analysis* (LDA) e *k-Nearest Neighbors*, addestrati mediante LOSO (*Leave-One-Subject-Out*) CV. Nel complesso, le migliori performance in termini di *accuracy* sono da attribuirsi ad AB-DT per quanto riguarda il problema a tre classi (80,34 %) e a LDA per il caso binario (93,12 %).

HAR

Allan Stisen e altri [34] hanno proposto una suddivisione del dataset in finestre temporali di due secondi con un 50% di *overlap* tra due finestre consecutive. Il *resampling* dei dati è testato a quattro frequenze diverse: 25 Hz, 50 Hz, 100 Hz, e 200 Hz. Dalle misurazioni dei due accelerometri e dei due giroscopi vengono estratte ulteriori *features*, quali la radice della somma dei quadrati delle componenti ($\sqrt{x^2 + y^2 + z^2}$) e la somma normalizzata di x,

y e *z*. Non viene eseguito un filtraggio dei dati volto a ridurre la dimensione del dataset per cui, anche effettuando il *downsampling* a 25 Hz, tempo di addestramento e complessità di esecuzione incrementano notevolmente. Come modelli di confronto vengono impiegati *C4.5 Decision Tree*, *Support Vector Machines*, *k-Nearest Neighbors* (k-NN) e *Random Forest*, addestrati tramite LOSO-CV e 10-Folds-CV. Quest'ultima garantisce risultati migliori, con un *F1-score*[3] medio del 91%, se combinata con l'algoritmo k-NN o *Random Forest*.

Gli autori di "Human Activity Recognition Using Federated Learning" [35] hanno utilizzato un approccio diverso: il *federated learning*, tipo di apprendimento nel quale il *task* è compiuto da una "federazione" di dispositivi partecipanti che generano i dati di training. L'obiettivo principale è quello di addestrare un modello sulla base di un sottoinsieme di client. In pratica, ogni client ha il proprio modello, addestrato mediante SGD (*Stochastic Gradient Descent*) sui dati raccolti localmente. Quando tutti i client sono stati addestrati tramite diverse iterazioni di SGD, il server aggiorna il modello globale, tenendo conto di quelli locali. La finestra temporale applicata è di tre secondi, con un'*overlap* del 50%, dalla quale vengono estratte media, deviazione standard, massimo e minimo. Il dataset è stato ripartito in tre client, ad ognuno dei quali sono state assegnate due differenti attività. Per ogni dataset locale, il *resampling* viene effettuato a 65 Hz e i dati sono suddivisi in 90% training-set e 10% test-set, mentre le proporzioni del dataset generale sono 80% training-set e 20% test-set. Come tipologia di modello è stata scelta una rete neurale *feed-forward*, più specificatamente una *Deep Neural Network* (DNN). Prima dell'addestramento, è stata eseguita una ricerca degli iperparametri relativa al miglior numero di unità e *hidden layer*. La DNN più efficiente consiste di due *hidden layer* con cento neuroni ciascuno, per un totale di ben 24.006 *trainable parameters*. Utilizzando l' *Early Stopping*, gli autori sono giunti ad un'*accuracy* massima dell'87% per quanto riguarda il *federated learning*, mentre mediante apprendimento centralizzato il picco raggiunto è del 93%.

[3] Metrica diversa da *accuracy*, che si concentra sull'individuazione di falsi positivi e falsi negativi. Risulta utile in caso di poco bilanciamento tra le classi.

PAMAP2

Il *pre-processing* compiuto da Attila Reiss e Didier Stricker [37] si preoccupa innanzitutto del bilanciamento dei dati, mediante *resampling* delle misurazioni dei vari sensori alla medesima frequenza (100 Hz). Il dataset viene poi segmentato grazie alla suddivisione in finestre temporali da 5,12 secondi l'una, con *shift* di un secondo tra due finestre consecutive. Successivamente, vengono estratte ulteriori *features* dai dati provenienti dai sensori IMU, quali media, mediana e deviazione standard, per poi eseguire una standardizzazione dei campioni relativi al battito cardiaco. I modelli confrontati sono cinque: *C4.5 Decision Tree*, *Boosted C4.5 Decision Tree*, *Bagging C4.5 Decision Tree*, *Naive Bayes* e *k-Nearest Neighbors* (k-NN). L'addestramento avviene tramite LOSO-CV e 9-Folds-CV. Quest'ultimo garantisce risultati migliori. Vengono considerate sei attività (*lying*, *sitting/standing*, *walking*, *running* e *cycling*), ma *sitting* e *standing* sono unificate, data la maggiore difficoltà di distinzione dovuta ai sensori utilizzati. La percentuale di *accuracy* raggiunta da tutti i modelli, impiegando 9-Folds-CV, è molto alta, in particolare per k-NN, che supera il 99%.

OPPORTUNITY

Hesam Sagha e altri [38] hanno applicato, in fase di segmentazione, una finestra temporale di 0,5 secondi, con *overlap* del 50%. Gli esperimenti sono stati condotti utilizzando il valore medio, la media e la varianza dei campionamenti dei sensori. Le classi relative all'attività motoria del soggetto sono quattro: *stand*, *sit*, *walk* e *lie*, per il cui riconoscimento sono state confrontate le performance di quattro tipologie di classificatori: *k-Nearest Neighbors* (k-NN), *Nearest Centroid Classifier*, *Linear Discriminant Analysis* e *Quadratic Discriminant Analysis*. Non viene eseguito alcun bilanciamento delle classi, nè filtraggio dei dati per ridurre la dimensione del dataset. La percentuale di *accuracy* più alta viene raggiunta da k-NN (83%), mentre gli altri modelli si aggirano intorno al 60%. Senza considerare i dati rumorosi del soggetto 4, si nota un incremento notevole dell'*accuracy* (circa +5% per tutti i modelli tranne k-NN, che registra un aumento del 2%).

ASCERTAIN

Gli autori di "ASCERTAIN: Emotion and Personality Recognition using Commercial Sensors" [41] hanno preso, come campioni di dati, gli ultimi 50 secondi di ogni clip, per predire le quattro combinazioni di *arousal* e *valence* (HAHV - *High Arousal-High Valence*, LAHV - *Low Arousal-High Valence*, LALV - *Low Arousal-Low Valence* e HALV - *High Arousal-Low Valence*). Le *features* utilizzate comprendono dati e statistiche (quali media e deviazione standard) relative ai tre sensori biometrici (EEG, ECG e GSR) e al riconoscimento dei movimenti facciali tramite web-camera (EMO). Sono infine stati confrontati i risultati ottenuti grazie all'impiego di *Support Vector Machines* e *Naive Bayes* (NB), addestrati tramite *Leave-One-Out Cross-Validation*, in cui, ad ogni iterazione, una clip è impiegata per il testing e le altre per la fase di training. La percentuale più alta di *F1-score* è raggiunta dal modello NB: 71%.

4.2 Esito degli esperimenti

I dati assumono infine la forma descritta nella tabella 4.1.

Dataset	N	L	F	C
WESAD	952	100 (3 s)	14	4
HAR	960	100 (2 s)	12	6
PAMAP2	930	350 (10 s)	31	4
OPPORTUNITY	480	75 (3 s)	130	4
ASCERTAIN	3360	160 (5 s)	17	4

Tabella 4.1: *Shape* dei dati

N: Numero di sottosequenze, L: Lunghezza delle sottosequenze, F: Numero di *features*, C: Numero di classi.

4.2.1 *Tabelle*

Riportiamo sotto forma di tabelle i risultati degli esperimenti condotti. Sono presenti quattro tabelle per ogni dataset, due riferite al caso *single-layer* e due al caso *multi-layer*. Qualora le prestazioni sul validation-set della versione *multi-layer* risultino peggiori della versione *single-layer*, riportiamo i risultati di quest'ultima anche per la tabella a *layer* multipli.

In ognuna delle pagine successive sono raffigurati due tipi di tabelle, una relativa ai valori degli iperparametri e una riguardante le epoche e il tempo di addestramento. Più nello specifico, nel primo caso andiamo ad indicare i seguenti valori:

- Iperparametri relativi al modello ESN (sez. 2.2.4, sez. 3.5.1):
  - *Input Scaling* (**IS**).
  - *Leaky* (**L**).
  - *Spectral Radius* (**SR**).
- Iperparametri relativi all'ottimizzatore *Adam* (sez. 3.5.1):
  - *Learning Rate* (**LR**).
  - **Beta1**.
  - **Beta2**.

- Regolarizzatori (sez. 3.5.1):
  - *Kernel Regularizer* (**K**).
  - *Bias Regularizer* (**B**).
  - *Activity Regularizer* (**A**).
- Numero di unità (sez. 3.5.1):
  - Numero totale di unità impiegato (**N**).
  - Numero di *Trainable Parameters* esplorati (**TP**).
- Accuratezza raggiunta, basata su cinque iterazioni:
  - **Media**.
  - Deviazione standard (**SD**).

Per quanto riguarda le tabelle sottostanti, relative alle epoche e al tempo di addestramento, specifichiamo i seguenti valori per ogni modello:

- Epoche (sez.3.5.1):
  - Numero massimo di epoche per l'addestramento (**N**).
  - Valore del parametro *Patience*, relativo alla *callback* `EarlyStopping` (**P**).
  - **Media** delle epoche per cinque iterazioni dell'addestramento.
  - Deviazione Standard delle epoche per cinque iterazioni dell'addestramento (**SD**).
- **Tempo** impiegato in media per l'addestramento, espresso in secondi.

Per ogni dataset presentiamo i risultati raggiunti da ciascun modello con le relative configurazioni di iperparametri, mediante l'ausilio di venti tabelle:

- Dataset **WESAD**:
  - Architettura a *layer* singolo:
    - Configurazione degli iperparametri e *accuracy* raggiunta (tab. 4.2).
    - Epoche e tempo di addestramento (tab. 4.3).
  - Architettura a *layer* multipli:
    - Configurazione degli iperparametri e *accuracy* raggiunta (tab. 4.4).
    - Epoche e tempo di addestramento (tab. 4.5).
- Dataset **HAR**:
  - Architettura a *layer* singolo:

    - Configurazione degli iperparametri e *accuracy* raggiunta (tab. 4.6).
    - Epoche e tempo di addestramento (tab. 4.7).
  - Architettura a *layer* multipli:
    - Configurazione degli iperparametri e *accuracy* raggiunta (tab. 4.8).
    - Epoche e tempo di addestramento (tab. 4.9).
- Dataset **PAMAP2**:
  - Architettura a *layer* singolo:
    - Configurazione degli iperparametri e *accuracy* raggiunta (tab. 4.10).
    - Epoche e tempo di addestramento (tab. 4.11).
  - Architettura a *layer* multipli:
    - Configurazione degli iperparametri e *accuracy* raggiunta (tab. 4.12).
    - Epoche e tempo di addestramento (tab. 4.13).
- Dataset **OPPORTUNITY**:
  - Architettura a *layer* singolo:
    - Configurazione degli iperparametri e *accuracy* raggiunta (tab. 4.14).
    - Epoche e tempo di addestramento (tab. 4.15).
  - Architettura a *layer* multipli:
    - Configurazione degli iperparametri e *accuracy* raggiunta (tab. 4.16).
    - Epoche e tempo di addestramento (tab. 4.17).
- Dataset **ASCERTAIN**:
  - Architettura a *layer* singolo:
    - Configurazione degli iperparametri e *accuracy* raggiunta (tab. 4.18).
    - Epoche e tempo di addestramento (tab. 4.19).
  - Architettura a *layer* multipli:
    - Configurazione degli iperparametri e *accuracy* raggiunta (tab. 4.20).
    - Epoche e tempo di addestramento (tab. 4.21).

Tabella 4.2: WESAD: Benchmarking single-layer

	ESN Hyperparams			**Hyperparams**			**Regularizers**			**Units**		**Accuracy**	
Model	IS	L	SR	LR	Beta1	Beta2	K	B	A	N	TP	Media	SD
ESN	1.8	0.4	1.1	0.005	0.9	0.999	1e-6	1e-6	1e-6	1000	4004	92.44%	0.75%
S-RNN	–	–	–	0.005	0.85	0.98	1e-4	1e-6	1e-6	55	4074	83.09%	8%
LSTM	–	–	–	0.008	0.85	0.999	1e-5	1e-5	1e-6	25	4104	94.23%	3.49%
GRU	–	–	–	0.005	0.85	0.999	1e-5	1e-5	1e-6	29	4035	97.73%	1.01%

Tabella 4.3: WESAD single-layer: Epochs e tempo di addestramento

	Epochs				
Model	N	P	Media	SD	**Tempo (sec)**
ESN	200	20	112.3	12.02	115.65
S-RNN	200	20	46	17.2	21.76
LSTM	200	20	38.2	10.7	39.6
GRU	200	20	44	14.7	46.5

IS: Input Scaling, L: Leaky, SR: Spectral Radius, LR: Learning Rate, K: Kernel_Reg., B: Bias_Reg., A: Activity_Reg., TP: Trainable Parameters, P: Patience, SD: Std. Dev.

Tabella 4.4: WESAD: Benchmarking multi-layer

		ESN Hyperparams			Hyperparams			Regularizers			Units		Accuracy	
Model	**Layers**	IS	L	SR	LR	Beta1	Beta2	K	B	A	N	TP	Media	SD
ESN	3	1.8	0.4	1.1	0.005	0.9	0.99	1e-5	1e-6	1e-6	750	4004	94.96%	2.6%
S-RNN	3	–	–	–	0.005	0.8	0.985	1e-5	1e-4	1e-5	26	3930	94.62%	2.84%
LSTM	2	–	–	–	0.01	0.85	0.98	1e-5	1e-4	1e-5	16	4164	95.48%	1.17%
GRU	2	–	–	–	0.008	0.95	0.999	1e-5	1e-5	1e-6	18	3964	98.13%	1.16%

Tabella 4.5: WESAD multi-layer: Epochs e tempo di addestramento

	Epochs				
Model	N	P	Media	SD	**Tempo (sec)**
ESN	100	10	30.6	7.17	87.64
S-RNN	100	10	28.6	8.09	40.6
LSTM	100	10	41	7.63	55.72
GRU	100	10	19.6	2.24	43.95

IS: Input Scaling, L: Leaky, SR: Spectral Radius, LR: Learning Rate, K: Kernel_Reg., B: Bias_Reg., A: Activity_Reg., TP: Trainable Parameters, P: Patience, SD: Std. Dev.

Tabella 4.6: HAR: Benchmarking single-layer

	ESN Hyperparams			Hyperparams			Regularizers			Units		Accuracy	
Model	IS	L	SR	LR	Beta1	Beta2	K	B	A	N	TP	Media	SD
ESN	2.0	0.4	1.1	0.005	0.9	0.985	1e-4	1e-6	1e-5	1000	6006	88.96%	2.24%
S-RNN	–	–	–	0.005	0.85	0.999	1e-4	1e-6	1e-4	70	6236	74.78%	5.04%
LSTM	–	–	–	0.01	0.9	0.98	1e-6	1e-5	1e-4	33	6276	91.5%	4.37%
GRU	–	–	–	0.01	0.95	0.98	1e-4	1e-4	1e-6	38	6162	97.08%	1.21%

Tabella 4.7: HAR single-layer: Epochs e tempo di addestramento

	Epochs				
Model	N	P	Media	SD	**Tempo (sec)**
ESN	100	10	20.2	5.08	34.52
S-RNN	100	10	13.4	7.31	10.57
LSTM	100	10	19.8	10.63	17.25
GRU	100	10	15.8	6.62	14.05

IS: Input Scaling, L: Leaky, SR: Spectral Radius, LR: Learning Rate, K: Kernel_Reg., B: Bias_Reg., A: Activity_Reg., TP: Trainable Parameters, P: Patience, SD: Std. Dev.

Tabella 4.8: HAR: Benchmarking multi-layer

		ESN Hyperparams			Hyperparams			Regularizers			Units		Accuracy	
Model	**Layers**	IS	L	SR	LR	Beta1	Beta2	K	B	A	N	TP	Media	SD
ESN	2	2.0	0.4	1.1	0.005	0.95	0.985	1e-4	1e-6	1e-5	668	6018	89.79%	3.81%
S-RNN	4	–	–	–	0.008	0.85	0.999	1e-6	1e-6	1e-5	28	6110	78.54%	2.04%
LSTM	3	–	–	–	0.01	0.85	0.985	1e-5	1e-5	1e-4	16	6182	92.71%	2.72%
GRU	3	–	–	–	0.01	0.8	0.999	1e-4	1e-6	1e-6	18	5946	98.54%	0.83%

Tabella 4.9: HAR multi-layer: Epochs e tempo di addestramento

	Epochs				
Model	N	P	Media	SD	**Tempo (sec)**
ESN	100	10	8.2	3.37	40.83
S-RNN	100	10	13.9	4.82	31.98
LSTM	100	10	14.6	3.06	42.46
GRU	100	10	11.2	2.04	36.13

IS: Input Scaling, L: Leaky, SR: Spectral Radius, LR: Learning Rate, K: Kernel_Reg., B: Bias_Reg., A: Activity_Reg., TP: Trainable Parameters, P: Patience, SD: Std. Dev.

Tabella 4.10: PAMAP2: Benchmarking single-layer

	ESN Hyperparams			**Hyperparams**			**Regularizers**			**Units**		**Accuracy**	
Model	IS	L	SR	LR	Beta1	Beta2	K	B	A	N	TP	Media	SD
ESN	2.0	0.4	0.9	0.008	0.8	0.995	1e-4	1e-5	1e-5	750	3004	95.64%	0.89%
S-RNN	–	–	–	0.008	0.85	0.985	1e-4	1e-4	1e-5	39	2929	95.27%	1.45%
LSTM	–	–	–	0.01	0.8	0.99	1e-4	1e-4	1e-4	16	3140	96.36%	1.99%
GRU	–	–	–	0.01	0.8	0.99	1e-5	1e-6	1e-4	19	3044	96.36%	0%

Tabella 4.11: PAMAP2 single-layer: Epochs e tempo di addestramento

	Epochs				
Model	N	P	Media	SD	**Tempo (sec)**
ESN	100	10	7.8	2.93	26.75
S-RNN	100	10	9.4	6.05	12.68
LSTM	100	10	10.8	6.65	24.12
GRU	100	10	13.8	7.08	31.19

IS: Input Scaling, L: Leaky, SR: Spectral Radius, LR: Learning Rate, K: Kernel_Reg., B: Bias_Reg., A: Activity_Reg., TP: Trainable Parameters, P: Patience, SD: Std. Dev.

Tabella 4.12: PAMAP2: Benchmarking multi-layer

		ESN Hyperparams			Hyperparams			Regularizers			Units	Accuracy	
Model	**Layers**	IS	L	SR	LR	Beta1	Beta2	K	B	A	N	TP	Media	SD
ESN	3	2.0	0.4	0.9	0.01	0.85	0.99	1e-5	1e-5	1e-6	550	2932	97.5%	2.74%
S-RNN	3	–	–	–	0.008	0.85	0.999	1e-4	1e-4	1e-4	21	3007	96%	3.39%
LSTM	2	–	–	–	0.01	0.95	0.98	1e-4	1e-5	1e-4	11	2952	96.5%	1.22%
GRU	2	–	–	–	0.01	0.8	0.98	1e-6	1e-5	1e-4	14	3294	98.5%	2%

Tabella 4.13: PAMAP2 multi-layer: Epochs e tempo di addestramento

	Epochs				
Model	N	P	Media	SD	**Tempo (sec)**
ESN	100	10	5.2	1.72	57.24
S-RNN	100	10	7.9	3.59	44.22
LSTM	100	10	9.4	2.94	54.29
GRU	100	10	17.4	11.89	77.25

IS: Input Scaling, L: Leaky, SR: Spectral Radius, LR: Learning Rate, K: Kernel_Reg., B: Bias_Reg., A: Activity_Reg., TP: Trainable Parameters, P: Patience, SD: Std. Dev.

Tabella 4.14: OPPORTUNITY: Benchmarking single-layer

	ESN Hyperparams			**Hyperparams**			**Regularizers**			**Units**		**Accuracy**	
Model	IS	L	SR	LR	Beta1	Beta2	K	B	A	N	TP	Media	SD
ESN	1.3	0.6	1.0	0.005	0.85	0.985	1e-5	1e-5	1e-6	1000	4004	87.5%	3.73%
S-RNN	–	–	–	0.01	0.85	0.985	1e-4	1e-5	1e-6	26	4190	87.59%	1.69%
LSTM	–	–	–	0.008	0.9	0.995	1e-6	1e-4	1e-6	7	3896	93.08%	2.88%
GRU	–	–	–	0.005	0.85	0.98	1e-5	1e-4	1e-6	9	3847	92.31%	2.43%

Tabella 4.15: OPPORTUNITY single-layer: Epochs e tempo di addestramento

	Epochs				
Model	N	P	Media	SD	**Tempo (sec)**
ESN	100	10	19.6	8.36	10.3
S-RNN	100	10	7.2	2.14	3.67
LSTM	100	10	12.4	3.5	7.71
GRU	100	10	19.4	5.92	9.18

IS: Input Scaling, L: Leaky, SR: Spectral Radius, LR: Learning Rate, K: Kernel_Reg., B: Bias_Reg., A: Activity_Reg., TP: Trainable Parameters, P: Patience, SD: Std. Dev.

Tabella 4.16: OPPORTUNITY: Benchmarking multi-layer

		ESN Hyperparams			**Hyperparams**			**Regularizers**			**Units**		**Accuracy**	
Model	**Layers**	IS	L	SR	LR	Beta1	Beta2	K	B	A	N	TP	Media	SD
ESN	3	1.3	0.6	1.0	0.001	0.8	0.99	1e-5	1e-6	1e-6	750	4004	94.74%	5.77%
S-RNN	3	–	–	–	0.01	0.8	0.995	1e-5	1e-6	1e-5	18	4090	96.84%	2.58%
LSTM	1	–	–	–	0.008	0.9	0.995	1e-6	1e-4	1e-6	6	4260	93.08%	2.88%
GRU	3	–	–	–	0.01	0.8	0.999	1e-5	1e-5	1e-5	8	4252	96.84%	2.58%

Tabella 4.17: OPPORTUNITY multi-layer: Epochs e tempo di addestramento

	Epochs				
Model	N	P	Media	SD	**Tempo (sec)**
ESN	100	10	43.8	7.55	27.42
S-RNN	100	10	22.6	9.88	10.23
LSTM	100	10	12.4	3.5	7.71
GRU	100	10	32.2	4.53	22.97

IS: Input Scaling, L: Leaky, SR: Spectral Radius, LR: Learning Rate, K: Kernel_Reg., B: Bias_Reg., A: Activity_Reg., TP: Trainable Parameters, P: Patience, SD: Std. Dev.

Tabella 4.18: ASCERTAIN: Benchmarking single-layer

	ESN Hyperparams			**Hyperparams**			**Regularizers**			**Units**		**Accuracy**	
Model	IS	L	SR	LR	Beta1	Beta2	K	B	A	N	TP	Media	SD
ESN	1.8	0.5	0.9	0.008	0.8	0.99	1e-5	1e-6	1e-4	800	3204	91.76%	0%
S-RNN	–	–	–	0.01	0.9	0.999	1e-4	1e-5	1e-4	46	3132	91.83%	0.14%
LSTM	–	–	–	0.005	0.99	0.98	1e-4	1e-5	1e-5	20	3124	91.76%	0%
GRU	–	–	–	0.01	0.9	0.99	1e-6	1e-6	1e-6	24	3196	91.76%	0%

Tabella 4.19: ASCERTAIN single-layer: Epochs e tempo di addestramento

	Epochs				
Model	N	P	Media	SD	**Tempo (sec)**
ESN	50	5	6.4	2.76	56.79
S-RNN	50	5	7.8	2.99	23.12
LSTM	50	5	14.6	6.83	66.68
GRU	50	5	9	4.56	53.5

IS: Input Scaling, L: Leaky, SR: Spectral Radius, LR: Learning Rate, K: Kernel_Reg., B: Bias_Reg., A: Activity_Reg., TP: Trainable Parameters, P: Patience, SD: Std. Dev.

Tabella 4.20: ASCERTAIN: Benchmarking multi-layer

		ESN Hyperparams			Hyperparams			Regularizers			Units		Accuracy	
Model	**Layers**	IS	L	SR	LR	Beta1	Beta2	K	B	A	N	TP	Media	SD
ESN	4	1.8	0.5	0.9	0.008	0.85	0.999	1e-6	1e-6	1e-5	625	3124	96.54%	0.77%
S-RNN	2	–	–	–	0.005	0.85	0.985	1e-5	1e-6	1e-6	30	3394	94.77%	0.78%
LSTM	5	–	–	–	0.001	0.9	0.999	1e-5	1e-6	1e-6	8	3044	94.63%	0%
GRU	2	–	–	–	0.01	0.95	0.999	1e-5	1e-4	1e-6	16	3380	94.63%	0%

Tabella 4.21: ASCERTAIN multi-layer: Epochs e tempo di addestramento

	Epochs				
Model	N	P	Media	SD	**Tempo (sec)**
ESN	50	5	4.8	2.4	64.85
S-RNN	50	5	6.8	0.98	30.66
LSTM	50	5	16	2.55	76.55
GRU	50	5	5.6	2.8	59.39

IS: Input Scaling, L: Leaky, SR: Spectral Radius, LR: Learning Rate, K: Kernel_Reg., B: Bias_Reg., A: Activity_Reg., TP: Trainable Parameters, P: Patience, SD: Std. Dev.

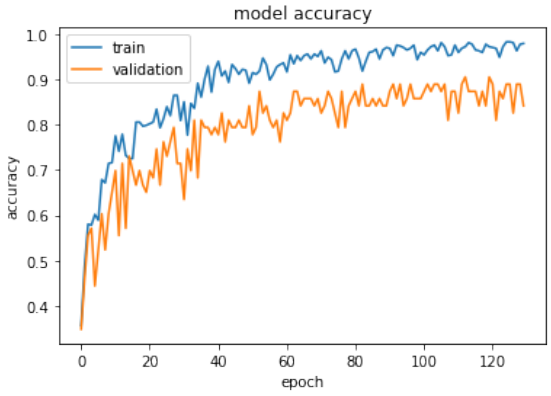
4.2.2 *Grafici*

I seguenti grafici mostrano l'andamento, sui dataset WESAD e HAR, di *accuracy* e *loss* (riportate sull'asse *y*) dei modelli RNN durante le epoche di addestramento (riportate sull'asse *x*). Le curve di *accuracy* e *loss* sono mostrate in blu per il training-set, in arancione per il validation-set. Illustriamo i grafici corrispondenti sia all'architettura *single-layer*, sia all'architettura *multi-layer*.

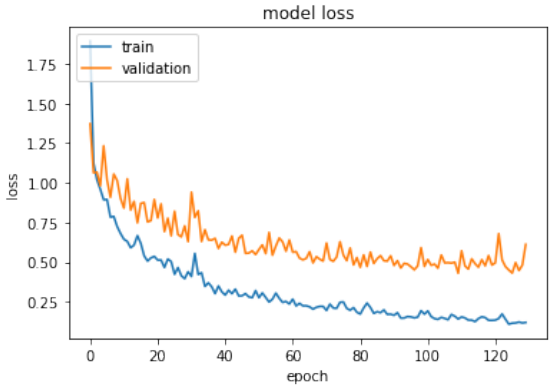
Per ogni dataset presentiamo quattro grafici per ciascun modello, suddivisi nel seguente modo:

- **(a)**: *Accuracy*, architettura *single-layer*.
- **(b)**: *Loss*, architettura *single-layer*.
- **(c)**: *Accuracy*, architettura *multi-layer*.
- **(d)**: *Loss*, architettura *multi-layer*.

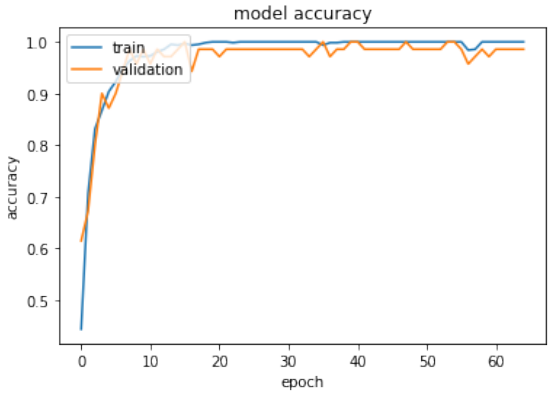
Abbbiamo dunque, per il dataset WESAD, le curve di *accuracy* e *loss* durante le epoche di addestramento, relative ai modelli ESN (fig. 4.5), SimpleRNN (fig. 4.6), LSTM (fig. 4.7) e GRU (fig.4.8). Analogamente illustriamo le curve di *accuracy* e *loss* dei modelli ESN (fig. 4.9), SimpleRNN (4.10), LSTM (4.11) e GRU (4.12) anche per il dataset HAR.



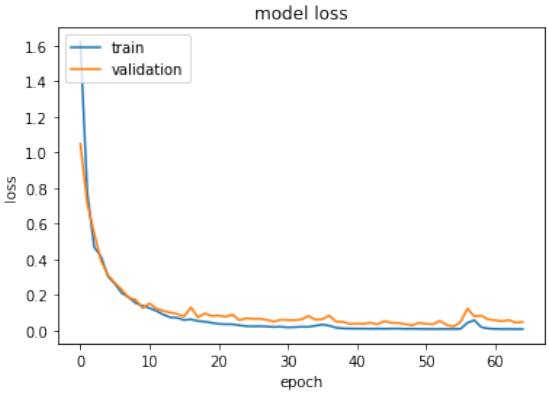
(a) *Accuracy single – layer*



(b) *Loss single – layer*

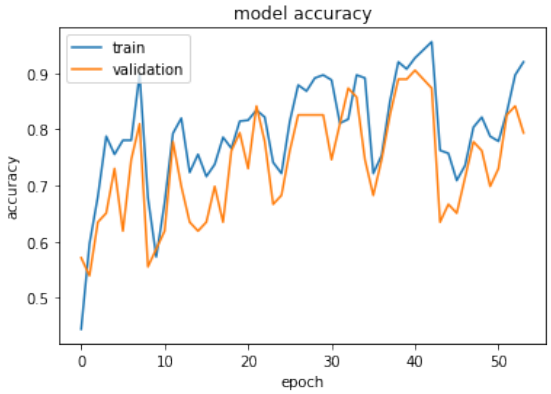


(c) *Accuracy multi – layer*

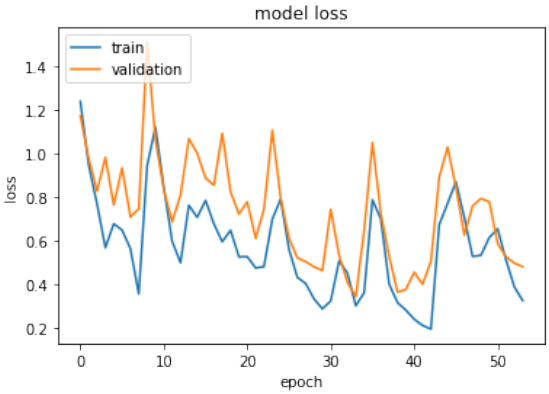


(d) *Loss multi – layer*

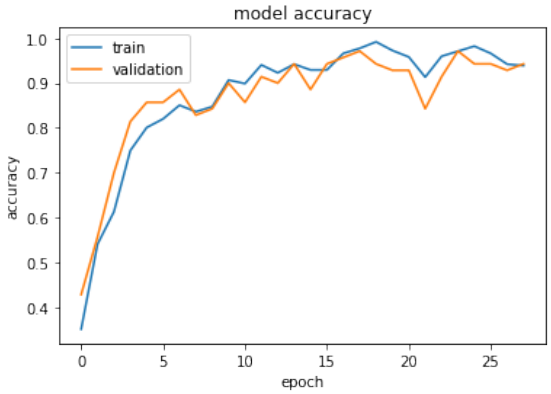
Figura 4.5: WESAD, modello ESN



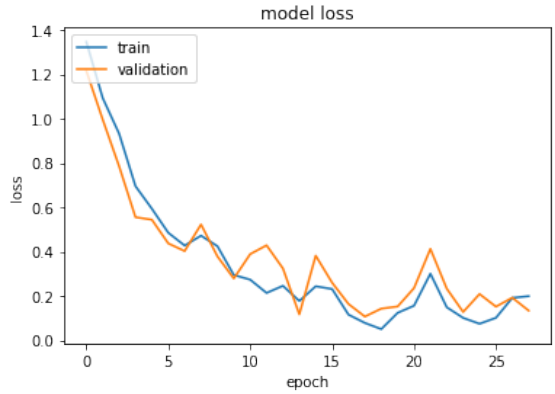
(a) *Accuracy single – layer*



(b) *Loss single – layer*

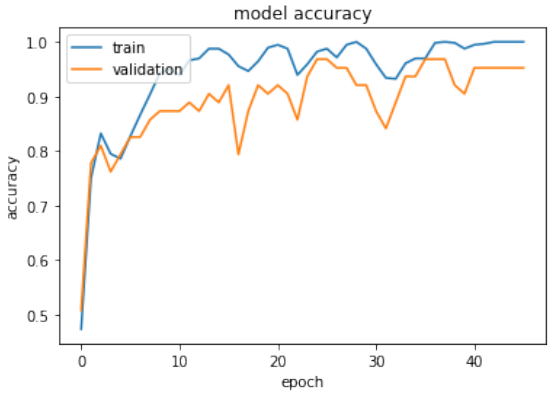


(c) *Accuracy multi – layer*

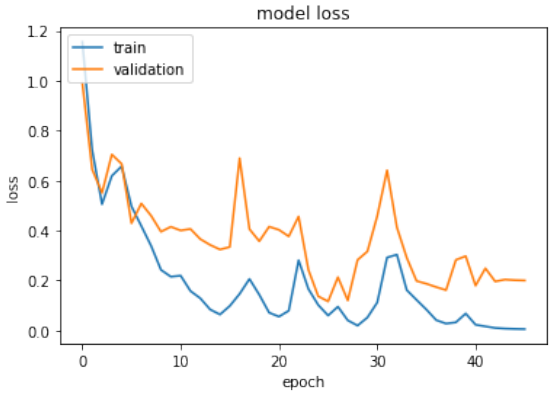


(d) *Loss multi – layer*

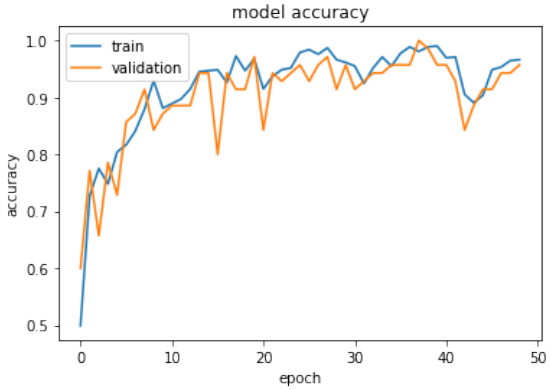
Figura 4.6: WESAD, modello SimpleRNN



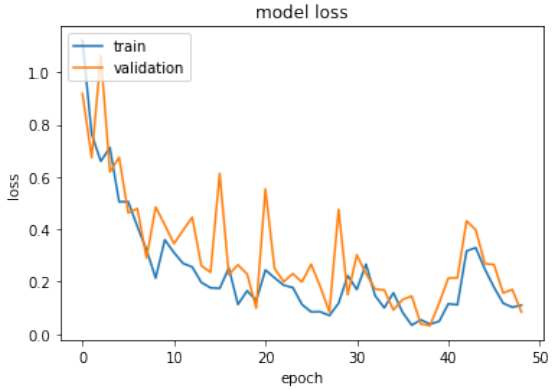
(a) *Accuracy single – layer*



(b) *Loss single – layer*

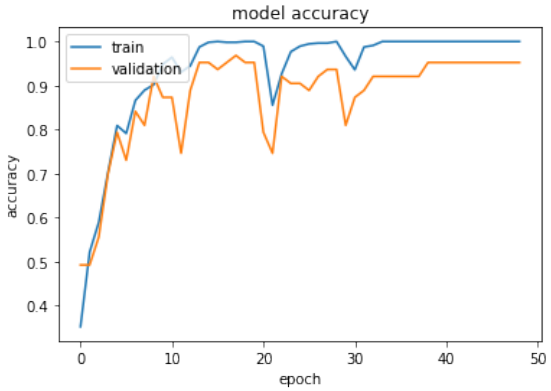


(c) *Accuracy multi – layer*

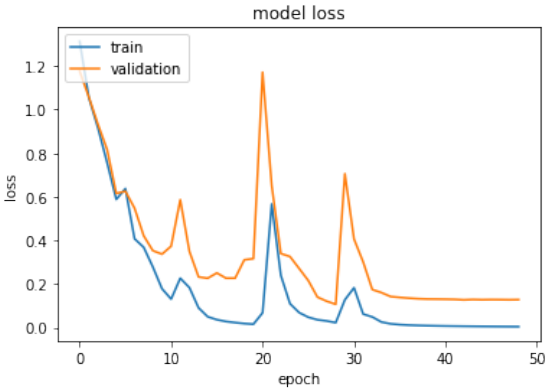


(d) *Loss multi – layer*

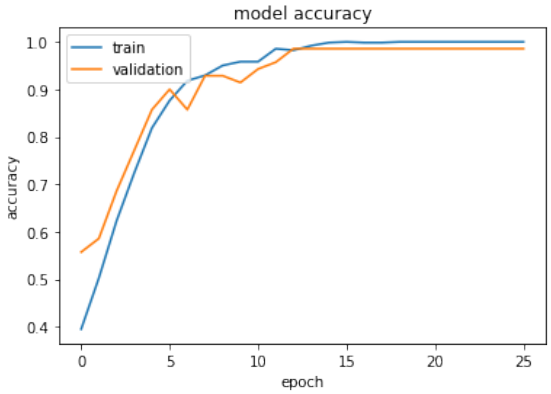
Figura 4.7: WESAD, modello LSTM



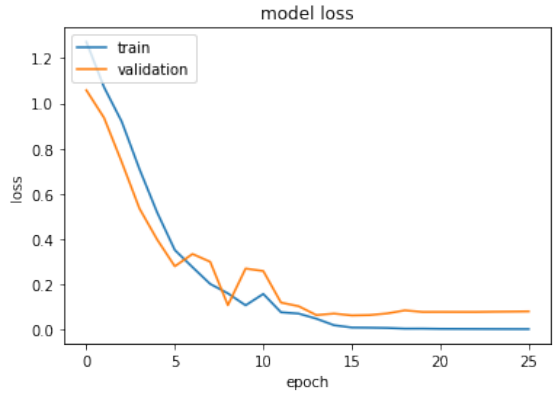
(a) *Accuracy single – layer*



(b) *Loss single – layer*

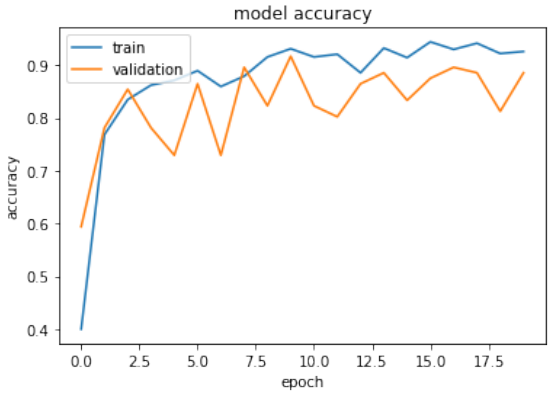


(c) *Accuracy multi – layer*

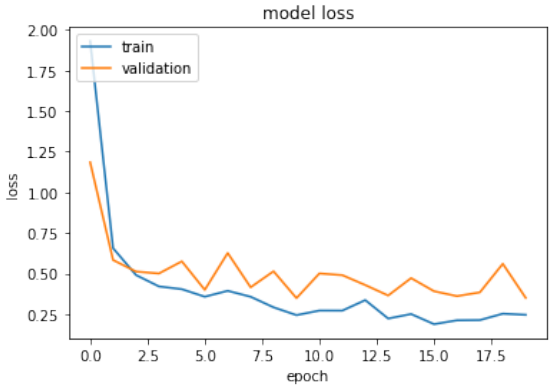


(d) *Loss multi – layer*

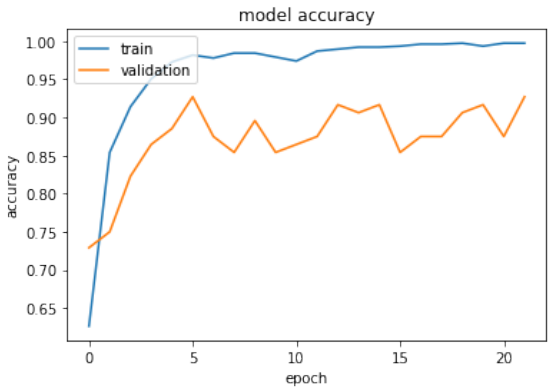
Figura 4.8: WESAD, modello GRU



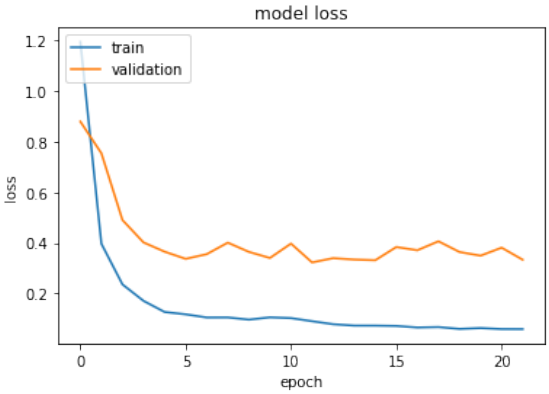
(a) *Accuracy single – layer*



(b) *Loss single – layer*

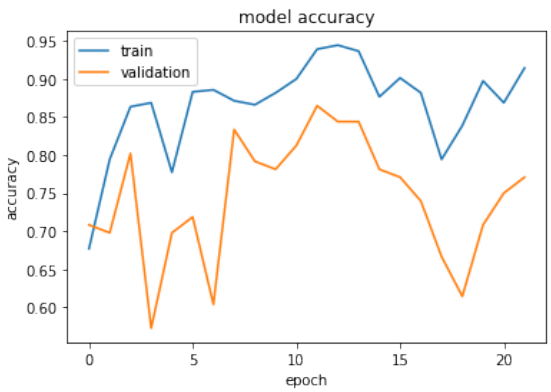


(c) *Accuracy multi – layer*



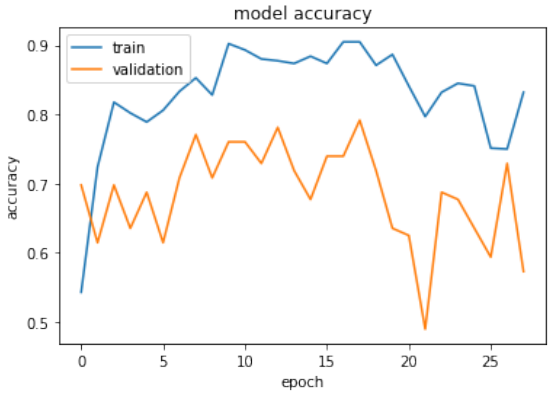
(d) *Loss multi – layer*

Figura 4.9: HAR, modello ESN



(a) *Accuracy single – layer*

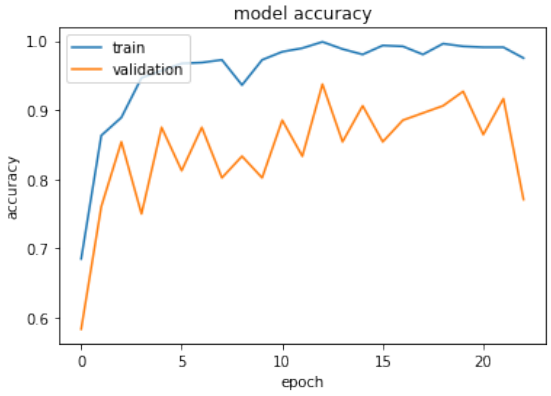
(b) *Loss single – layer*



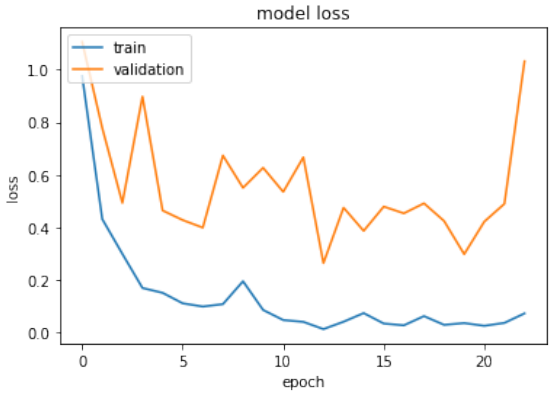
(c) *Accuracy multi – layer*

(d) *Loss multi – layer*

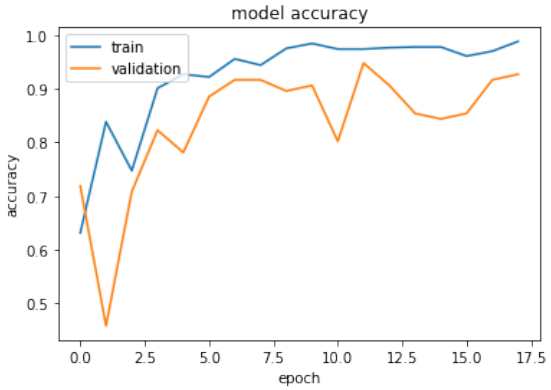
Figura 4.10: HAR, modello SimpleRNN



(a) *Accuracy single – layer*



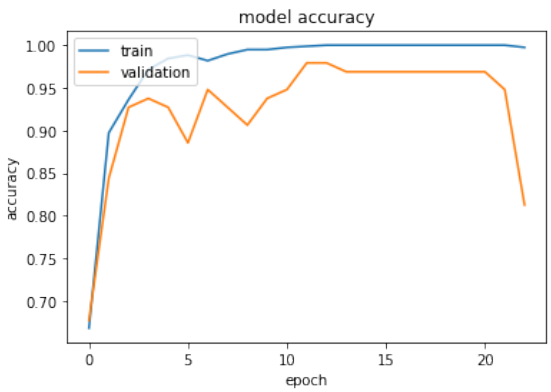
(b) *Loss single – layer*



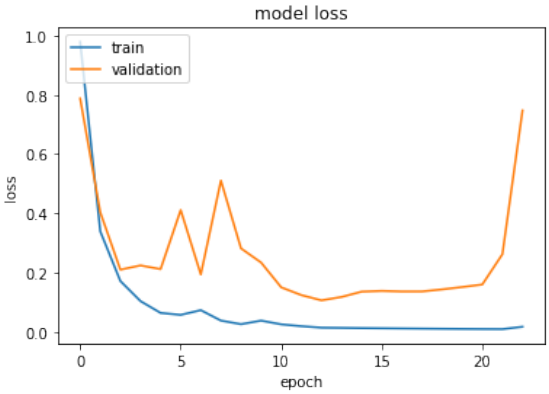
(c) *Accuracy multi – layer*

(d) *Loss multi – layer*

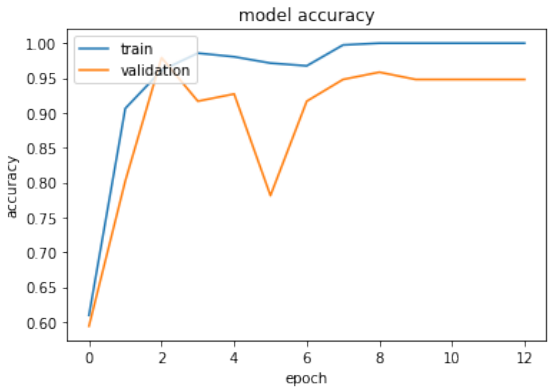
Figura 4.11: HAR, modello LSTM



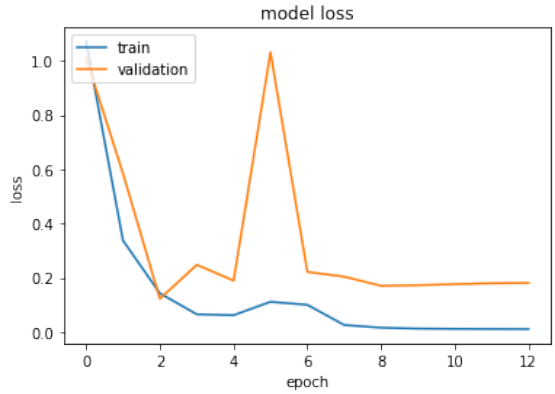
(a) *Accuracy single – layer*



(b) *Loss single – layer*



(c) *Accuracy multi – layer*



(d) *Loss multi – layer*

Figura 4.12: HAR, modello GRU

4.2.3 *Discussione dei risultati*

Complessivamente i valori di *accuracy* raggiunti dalle reti neurali ricorrenti si attestano oltre il 90%, indicando quindi un'ottima affidabilità, sia per la predizione dello stato fisico, sia per la predizione dello stato emotivo.

A differenza di quanto fatto negli studi precedentemente citati [32-41], si è lavorato tenendo conto dei vincoli hardware dei dispositivi indossabili, limitando di conseguenza il numero di parametri addestrabili ed eseguendo un corposo filtraggio dei dati. Bisogna inoltre considerare che la quasi totalità dei modelli non-RNN impiegati (sez. 4.1.6) si è avvalsa dell'addestramento mediante *Cross-Validation*, una tecnica che aumenta l'efficacia delle capacità predittive del modello (anche riducendo il rischio di *overfitting*), ma a scapito di un ragguardevole incremento del tempo di training.

Ciò nonostante, i risultati raggiunti in questo lavoro superano (o pareggiano) quasi sempre le performance degli altri modelli non-RNN impiegati in letteratura [32-41]. In particolare, la superiorità delle RNN è evidente per quanto riguarda entrambi i dataset per la predizione dello stato emotivo (WESAD [32,33] e ASCERTAIN [41]) e OPPORTUNITY [38], con un miglioramento delle prestazioni anche del 15%. Specificatamente per il dataset WESAD, gli studi già citati [32,33] hanno analizzato tutt'al più il caso con tre classi, conseguendo un picco massimo di *accuracy* dell'80%, tramite l'algoritmo *AdaBoost-DT*. Grazie alle reti neurali ricorrenti invece, considerando il totale delle classi (quattro), tutti i modelli *multi-layer* superano il 94%, con picchi fino al 98% nel caso di GRU. Volendo invece confrontare le performance sul dataset PAMAP2 [37], notiamo un sostanziale equilibrio di prestazioni (95-98% per le RNN e 94-99% per gli altri modelli). Per quanto riguarda il dataset HAR [34,35] infine, l'*accuracy* raggiunta da ESN (90%) e LSTM (93%) *multi-layer* concorre con i risultati conseguiti dall'algoritmo k-NN (91%) [34] e dal modello di rete neurale *feed-forward* DNN (93%) [35], mentre GRU (98,5%) si rivela superiore di oltre cinque punti percentuali. Peraltro, durante l'addestramento della DNN viene esplorato un numero di *trainable parameters* quattro volte superiore rispetto a quello relativo ai modelli RNN (24.000 vs. 6000). Le reti neurali ricorrenti si dimostrano quindi molto redditizie in termini di efficienza.

Analizzando e confrontando tra loro i risultati dei quattro tipi di RNN utilizzati, possiamo notare come nella maggior parte dei casi i modelli *multi-layer* raggiungano un'*accuracy* più alta rispetto alla controparte *single-layer*, a parità di *trainable parameters*. In particolare, facendo riferimento ai grafici riguardanti il dataset WESAD (figg. 4.5, 4.6, 4.7, 4.8), osserviamo che il caso a *layer* multipli, oltre a garantire una precisione maggiore, riduce notevolmente le oscillazioni delle funzioni *loss* e *accuracy*, rispetto ai grafici *single-layer*.

In riferimento ai dataset WESAD e HAR, SimpleRNN appare, tra i quattro modelli di RNN, il meno affidabile nel caso *single-layer* (83% WESAD e 75% HAR), difficoltà rilevabile anche dai grafici corrispondenti (figg. 4.6, 4.10). Questo è dovuto alla maggiore semplicità di struttura della RNN standard, i cui principali difetti sono brevemente esposti nella Sezione 2.2.1. Ciò nonostante, aggiungendo *hidden layer*, il modello di RNN *vanilla* assicura comunque il 79% di *accuracy* per HAR e quasi il 95% per WESAD.

In termini di performance di *accuracy*, il modello GRU raggiunge generalmente i risultati migliori, con una percentuale che si aggira oltre il 95% nella maggior parte dei casi. Le prestazioni di GRU sono notevolmente elevate anche con l'architettura *single-layer*, risultando di poco inferiori (circa 2%) rispetto all'architettura *multi-layer* e mantenendosi sempre al di sopra del 90%. In particolare, possiamo notare l'efficienza del modello GRU *single-layer* confrontando l'accuratezza raggiunta sul dataset HAR, che supera di oltre 20 punti percentuali il modello SimpleRNN (*single-layer*).

Relativamente al modello ESN, possiamo osservare come le prestazioni raggiunte, pur addestrando il solo *layer* di output, risultino generalmente in linea con le performance di modelli più complessi quali LSTM e GRU. Grazie al paradigma RC (sez. 2.2.4) [4-7], viene infatti risparmiato il calcolo degli stati ad ogni epoca di addestramento, riducendo la complessità del modello senza però rinunciare all'accuratezza predittiva.

In merito ai tempi di addestramento, vi è un'effettiva omogeneità tra i vari modelli, eccezion fatta per ESN, che possiede tempi più lunghi. Tuttavia, come si può vedere dalle tabelle, una volta implementato il *caching* degli stati (sez. 3.5.1), la durata necessaria per il training del modello ESN si abbassa notevolmente, arrivando molto spesso a competere coi valori relativi a LSTM e GRU. In alcuni casi (e.g. WESAD), la fase di training del modello ESN risulta comunque più lunga rispetto agli altri modelli, a causa della dimensione degli stati caricati inizialmente. Senza implementazione del *caching*, il caricamento

viene inutilmente eseguito ad ogni epoca di addestramento. Una volta caricati e memorizzati gli stati iniziali tramite *caching* (procedimento che richiede gran parte del tempo di addestramento), il training del *readout* durante le epoche viene eseguito in un tempo minore rispetto agli altri modelli.

Infine, i modelli LSTM e GRU, a differenza del modello ESN, godono di importanti ottimizzazioni automatiche di *Keras* in dispositivi con supporto GPU. I risultati derivanti dall'impiego delle *Echo State Networks* si dimostrano dunque incoraggianti e preferibili in contesti di applicazioni distribuite su *device* sprovvisti di GPU, quali, appunto, smartband e smartwatch.

5

CONCLUSIONI

E' stata presentata un'analisi comparativa tra quattro modelli di reti neurali ricorrenti per la predizione dello stato fisico ed emotivo, sulla base di misurazioni biometriche provenienti da cinque dataset: WESAD, HAR, PAMAP2, OPPORTUNITY e ASCERTAIN. I dati ricavati derivano da sensori biometrici multipli (e.g. elettrocardiogramma, accelerometro) posti su vari soggetti partecipanti. Lo studio si è concentrato sulla suddivisione temporale dei dati in sottosequenze, da utilizzare per l'addestramento e la valutazione delle performance delle reti neurali ricorrenti standard, LSTM, GRU ed ESN. L'individuazione di un *trade-off* ottimale tra complessità e prestazioni ha assunto un ruolo primario. E' stata infatti prestata particolare attenzione ad un approccio poco intrusivo, considerando un'applicazione pratica su dispositivi quali smartband e smartwatch.

Attraverso la ricerca degli iperparametri, ciascun modello di RNN è stato implementato *ad hoc* per ogni dataset. Una volta selezionati gli iperparametri migliori è stato eseguito l'addestramento, scegliendo il numero di unità in modo da limitare la complessità, senza tuttavia impattare sulle prestazioni in modo significativo.

I risultati conseguiti attestano una percentuale di *accuracy* sopra il 90% nella maggior parte dei casi, superando (talvolta notevolmente) o pareggiando le performance dei modelli non-RNN impiegati in altri lavori sugli stessi dataset.

L'applicazione pratica delle reti neurali ricorrenti mediante l'implementazione su smartband e smartwatch, rappresenta, soprattutto per quanto riguarda la predizione dello stato emotivo, una possibilità concreta in più per il monitoraggio delle attività e dell'umore quotidiano. In particolare, dal momento che la fase di addestramento delle reti neurali *deep* randomizzate coinvolge esclusivamente il *readout* (giacché il *reservoir* viene solo inizializzato), il loro impiego su dispositivi indossabili si appresta ad offrire un ottimo compromesso tra complessità e prestazioni.

6

BIBLIOGRAFIA

[1] Samir B. Unadkat e altri, *Recurrent Neural Network, Design and Applications*, L.R. Medsker and L.C. Jain, CRC Press, Boca Raton, Florida, FL, USA, 1999.

[2] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho and Yoshua Bengio, "How to Construct Deep Recurrent Neural Networks", *CoRR*, 2014.

[3] Ken-ichi Funahashi and Yuichi Nakamura, "Approximation of dynamical systems by continuous time recurrent neural networks", *Neural Networks*, Vol. 6, N. 6, 1993, pp. 801-806.

[4] Claudio Gallicchio, Alessio Micheli, Luca Pedrelli, "Deep reservoir computing: A critical experimental analysis", *Neurocomputing*, Vol. 268, 2017, pp. 87-99.

[5] Lukoševičius, Mantas, and Herbert Jaeger, "Reservoir computing approaches to recurrent neural network training", *Computer Science Review*, Vol. 3, N. 3, 2009, pp. 127-149.

[6] Benjamin Schrauwen, David Verstraeten and Jan Van Campenhout, "An overview of reservoir computing: theory, applications and implementations", *Proceedings of the 15th european symposium on artificial neural networks*, 2007, pp. 471-482.

[7] David Verstraeten e altri, "An experimental unification of reservoir computing methods", *Neural networks*, Vol. 20, N. 3, 2007, pp. 391-403.

[8] Alex Sherstinsky, "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network", *Elsevier "Physica D: Nonlinear Phenomena" journal*, Vol. 404, 2020.

[9] Christoph Aurnhammer and Stefan L. Frank, "Comparing Gated and Simple Recurrent Neural Network Architectures as Models of Human Sentence Processing", in: *Proceedings of the 41st Annual Conference of the Cognitive Science Society*, 2019, pp. 112-118.

[10] Sanidhya Mangal, Poorva Joshi and Rahul Modak, "LSTM vs. GRU vs. Bidirectional RNN for script generation", *arXiv e-prints*, 2019.

[11] J. J. Hopfield, "Artificial neural networks," in: *IEEE Circuits and Devices Magazine*, Vol. 4, N. 5, 1988, pp. 3-10.

[12] Ajith Abraham, "Artificial Neural Networks", *Handbook of Measuring System Design*, 2005, pp. 901-908.

[13] Rongrong Liu, Florent Nageotte, Philippe Zanne, Michel de Mathelin and Birgitta Dresp-Langley, "Deep Reinforcement Learning for the Control of Robotic Manipulation: A Focussed Mini-Review", *Robotics*, Vol. 10, N. 22, 2021.

[14] `https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/`

[15] P. J. Werbos, "Backpropagation through time: what it does and how to do it," in: *Proceedings of the IEEE*, Vol. 78, N. 10, 1990, pp. 1550-1560.

[16] Robert Hecht-Nielsen, "Theory of the Backpropagation Neural Network", *Neural Networks for Perception*, 1992, pp. 65-93.

[17] Marko Bursac, Danijela Milosevic and Katarina Mitrovic, "Proposed Model For Automatic Learning Style Detecting Based On Artificial Intelligence", in: *Science and Higher Education in Function of Sustainable Development (SED)*, 2019.

[18] https://www.moogsoft.com/blog/aiops/understanding-machine-learning-aiops-part-2

[19] Dana Hughes, Nicholaus Correll, "Distributed Machine Learning in Materials that Couple Sensing, Actuation, Computation and Communication", 2016.

[20] Nikhil Ketkar, "Introduction to Keras", in: *Deep Learning with Python*, 2017, pp. 95-109.

[21] http://dprogrammer.org/rnn-lstm-gru

[22] https://www.oreilly.com/library/view/hands-on-reinforcement-learning/9781788836524/a089ac1d-f4a0-44b5-999f-e70a7f25ab62.xhtml

[23] Paul Werbos, "Backpropagation through time: what it does and how to do it", *Proceedings of the IEEE*, Vol. 78, 1990, pp. 1550-1560.

[24] Sepp Hochreiter, "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions", *International Journal of Uncertainty Fuzziness and Knowledge-Based Systems*, Vol. 6, 1998, pp. 107-116.

[25] Hong Hui Tan and King Hann Lim, "Vanishing Gradient Mitigation with Deep Learning Neural Network Optimization," in: *2019 7th International Conference on Smart Computing & Communications (ICSCC)*, Sarawak, Malaysia, 2019, pp. 1-4.

[26] Razvan Pascanu, Tomas Mikolov and Yoshua Bengio, "On the difficulty of training Recurrent Neural Networks", in: *Proceedings of the 2013 International Conference on Machine Learning (ICML)*, 2013, pp. 1310-1318.

[27] George Philipp, Dawn Song and Jaime G. Carbonell, "The exploding gradient problem demystified - definition, prevalence, impact, origin, tradeoffs, and solutions", *CoRR*, 2017.

[28] Jeff Donahue, Lise Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko and Trevor Darrell, "Long-term recurrent convolutional networks for visual recognition and description", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 39, N. 4, 2017, pp. 677–691.

[29] https://www.developersmaggioli.it/blog/le-reti-neurali-ricorrenti/

[30] Dhireesha Kudithipudi, Qutaiba Saleh, Cory Merkel James Thesing and Bryant Wysocki, "Design and Analysis of a Neuromemristive Reservoir Computing Architecture for Biosignal Processing", *Frontier in Neuroscience*, 2016.

[31] Kim Taehwan and Brian R. King, "Time series prediction using deep echo state networks", *Neural Computing and Applications*, Vol. 32, 2020, pp. 17769–17787.

[32] Kizito Nkurikiyeyezu, Anna Yokokubo and Guillaume Lopez, "The Effect of Person Specific Biometrics in Improving Generic Stress Predictive Models", *Sensors and Materials*, Vol. 32, 2019, pp. 703-722.

[33] Philip Schmidt, Attila Reiss, Robert Duerichen, Claus Marberger and Kristof Van Laerhoven, "Introducing WESAD, a multimodal dataset for Wearable Stress and Affect Detection", in: *2018 International Conference on Multimodal Interaction (ICMI 2018)*, New York, NY, USA, 2018.

[34] Allan Stisen, Henrik Blunck, Sourav Bhattacharya, Thor Siiger Prentow, Mikkel Baun Kjærgaard, Anind Dey, Tobias Sonne and Mads Møller Jensen, "Smart Devices are Different: Assessing and Mitigating Mobile Sensing Heterogeneities for Activity Recognition", in: *Proc. 13th ACM Conference on Embedded Networked Sensor Systems (SenSys 2015)*, Seoul, Korea, 2015.

[35] Konstantin Sozinov, Vladimir Vlassov, Sarunas Girdzijauskas, "Human Activity Recognition Using Federated Learning", in: *IEEE Int. Conf. on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications*, 2019, pp. 1103–1111.

[36] Attila Reiss and Didier Stricker, "Creating and Benchmarking a New Dataset for Physical Activity Monitoring", in: *The 5th Workshop on Affect and Behaviour Related Assistance (ABRA)*, 2012.

[37] Attila Reiss and Didier Stricker, "Introducing a New Benchmarked Dataset for Activity Monitoring", in: *The 16th IEEE International Symposium on Wearable Computers (ISWC)*, 2012.

[38] Hesam Sagha, Sundara Tejaswi Digumarti, José del R. Millán, Ricardo Chavarriaga, Alberto Calatroni, Daniel Roggen and Gerhard Tröster, "Benchmarking classification techniques using the Opportunity human activity dataset", in:*IEEE International Conference on Systems, Man, and Cybernetics*, Anchorage, AK, USA, 2011.

[39] Ricardo Chavarriaga, e altri, "Robust activity recognition for assistive technologies: Benchmarking ML techniques, Workshop on Machine Learning for Assistive Technologies", in:*24th Annual Conference on Neural Information Processing Systems (NIPS)*, 2010.

[40] Ricardo Chavarriaga, Hesam Sagha, Alberto Calatroni, Sundaratejaswi Digumarti, Gerhard Tröster, José del R. Millán and Daniel Roggen, "The Opportunity challenge: A benchmark database for on-body sensor-based activity recognition", *Pattern Recognition Letters*, Vol. 34, N. 15, 2013, pp. 2033-2042.

[41] Ramanathan Subramanian, Julia Wache, Mojtaba Khomami Abadi, Radu L. Vieriu, Stefan Winkler and Nicu Sebe, "ASCERTAIN: Emotion and Personality Recognition using Commercial Sensors", *IEEE Transactions on Affective Computing*, Vol. 9, N. 2, 2018, pp. 147-160.

[42] `https://www.dataprix.com/en/blog-it/palakairon/3-practical-ways-use-decision-tree-your-advantages`

[43] Huan Wu, Liang Wang, Zhiyong Zhao, Chester Shu and Chao Lu, "Support Vector Machine based Differential Pulse-width Pair Brillouin Optical Time Domain Analyzer", *IEEE Photonics Journal*, Vol. 10, N. 4, 2018, pp. 1-11.

[44] Payam Refaeilzadeh, Lei Tang and and Huan Liu, "Cross-Validation", *Encyclopedia of Database Systems*, 2009, pp. 532-538.

[45] Daniel Berrar, "Cross-Validation", *Encyclopedia of Bioinformatics and Computational Biology*, Shoba Ranganathan, Kenta Nakai and Christian Schonbach, Elsevier, 2018.

[46] Jeff Schneider and Andrew W. Moore, "A Locally Weighted Learning Tutorial using Vizier 1.0", 1997, pp. 32,33.

[47] Sebastian Raschka, "Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning", 2018.

[48] Xue Ying, "An Overview of Overfitting and its Solutions", *Journal of Physics: Conference Series 1168*, 2019.

[49] Martin Abadi e altri, "TensorFlow: A system for large-scale machine learning", *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265-283.

[50] Fabian Pedregosa e altri, "Scikit-learn: Machine Learning in Python", *Journal of Machine Learning Research*, Vol. 12, 2011, pp. 2825-2830.

[51] Andrew P. Allen, Paul J. Kennedy, Samantha Dockray, John F. Cryan, Timothy G. Dinan and Gerard Clarkea, "The Trier Social Stress Test: Principles and practice", *Neurobiology of Stress*, Vol. 6, 2017, pp. 113-126.

[52] Christopher J. Pannucci and Edwin G. Wilkins, "Identifying and Avoiding Bias in Research", *Plastic and reconstructive surgery*, Vol. 126, N. 2, 2010, pp. 619-625.

[53] `https://en.wikipedia.org/wiki/Emotion_classification#Circumplex_model`

[54] James Bergstra and Yoshua Bengio, "Random Search for Hyper-Parameter Optimization", *Journal of Machine Learning Research*, Vol. 13, 2012, pp. 281-305.

[55] Diederik P. Kingma and Jimmy Ba, "Adam: A Method for Stochastic Optimization", in: *3rd International Conference for Learning Representations*, San Diego, California, CA, USA, 2015.

[56] Kaibo Duan, S.Sathiya Keerthi, Wei Chu, Shirish Krishnaj Shevade and Aun Neow Poo, "Multi-category Classification by Soft-Max Combination of Binary Classifiers", *Lecture Notes in Computer Science*, Vol. 2709, 2003.

[57] Ozgur Demir-Kavuk, Mayumi Kamada, Tatsuya Akutsu and Ernst-Walter Knapp, "Prediction using step-wise L1, L2 regularization and feature selection for small data sets with large number of features", *BMC Bioinformatics*, Vol. 12, N. 412, 2011.

[58] Tong Zhang and Bin Yu, "Boosting with early stopping: Convergence and consistency", *Annals of Statistics*, Vol. 33, N. 4, 2005, pp. 1538-1579.

7

APPENDICE

Modelli utilizzati in letteratura

- *Decision Tree* (DT): Dato un insieme di dati di training, viene eseguita una ricerca nello spazio dell'albero decisionale, separando i *data points* in due categorie alla volta. Infine viene restituita la classe di appartenenza di ogni esempio. *AdaBoost DT*, *C4.5 DT*, *Boosted C4.5 DT* e *Bagging C4.5 DT* costituiscono ottimizzazioni ed estensioni dell'algoritmo standard per gli alberi decisionali *ID3*.
- *Random Forest* (RF): Espansione di DT, combina una moltitudine di alberi decisionali in un unico modello. Il risultato finale è dato dalla classe restituita dal maggior numero di alberi.
- *Linear Discriminant Analysis* (LDA): Metodo per trovare una combinazione lineare di caratteristiche che separi due o più classi in un problema di classificazione. In pratica "proietta" i dati di input su uno spazio di dimensione inferiore, rendendo le classi più facilmente separabili. La combinazione risultante può essere utilizzata come classificatore lineare.
- *Quadratic Discriminant Analysis* (QDA): Metodo analogo a LDA, in cui la funzione "discriminante" non è lineare, bensì quadratica.
- *k-Nearest Neighbors* (k-NN): Algoritmo che utilizza il set di dati di addestramento per trovare i k *data points* più vicini all'esempio di input. Restituisce la classe più diffusa tra i k esempi più vicini.
- *Nearest Centroid Classifier* (NCC): Classificatore in cui ogni classe è caratterizzata dal proprio centroide (i.e. la media degli esempi di training). Per ogni input, NCC assegna un'etichetta che indica la classe il cui centroide è più vicino all'esempio.
- *Support Vector Machines* (SVM): Divide i dati in diverse classi, mediante un iperpiano di separazione che massimizza la distanza tra di esse (margine). Maggiore è la distanza, maggiore è l'accuratezza del modello.
- *Naive Bayes* (NB): Algoritmo di classificazione. Calcola la probabilità che un dato di input appartenga o meno a una determinata categoria (target), tramite la formula

  $$P(A|B)^1 = \frac{P(B|A) \times P(A)}{P(B)}. \quad (\textit{Teorema di Bayes})$$

[1] Probabilità che l'*input* A appartenga alla categoria B