Università di Pisa

DIPARTIMENTO DI INFORMATICA
Laurea Triennale in Informatica

Metodologie di Apprendimento Automatico Continuo per il Monitoraggio dello Stato Umano

Candidato:
Federico Matteoni

Relatori:
Prof. Vincenzo Lomonaco
Prof. Claudio Gallicchio
Prof. Davide Bacciu

Anno Accademico 2020-2021

A Simone

Indice

Elenco delle figure

Elenco delle tabelle

Glossario

Abbreviazioni e Acronimi

Termine	Significato
BWT	Backward Transfer, Traserimento all'Indietro
EWC	Elastic Weight Consolidation, Consolidamento Elastico dei Pesi
FWT	Forward Transfer, Trasferimento in Avanti
GPU	Graphics Processing Unit, Unità di Processo Grafico
GRU	Gated Recurrent Unit, Unità Ricorrente con Cancello
HAR	Human Activity Recognition, Riconoscimento dell'Attività Umana
HSM	Human State Monitoring, Monitoraggio dello Stato Umano
LSTM	Long Short-Term Memory, Memorie a Lungo Breve-Termine
LWF	Learning Without Forgetting, Apprendere Senza Dimenticare
ML	Machine Learning, Apprendimento Automatico
MNIST	Modified National Institute of Standards and Technology Database
ReLU	Rectified Linear Unit, Unità Lineare Rettificata
RNN	Recurrent Neural Network, Rete Neurale Ricorrente
TPU	Tensor Processing Unit, Unità di Processo Tensoriale

Introduzione

Negli ultimi decenni, l'"Intelligenza Artificiale" se è fatta sempre più spazio nella nostra vita di tutti i giorni. Grazie ai nostri smartphone, usiamo modelli di Machine Learning efficienti che indirizzano le nostre ricerche nella rete, e grazie a tali modelli le aziende possono mostrarci inserzioni sempre più aderenti ai nostri gusti personali. Ma il mondo del Machine Learning ha importanti e utili applicazioni anche nella salvaguardia del benessere dell'individuo, ad esempio in contesti come il riconoscimento delle malattie o l'interpretazione del linguaggio parlato. Questi contesti, che rientrano nell'ambito dello *Human State Monitoring*, stanno diventando sempre più importanti nella vita di tutti i giorni e diventerà sempre più necessario l'avere a disposizione sistemi che sappiano sfruttare efficacemente i dati sull'utente che hanno a disposizione, così da realizzare sistemi responsivi e reattivi in base allo stato attuale dell'utente.

Con Addestramento Continuo si definisce l'approccio all'addestrare un modello di Machine Learning in diverse sessioni di addestramento distribuite nel tempo, con l'obiettivo di migliorare gradualmente la sua performance. Questo approccio può portare ad un graduale deterioramento delle conoscenze acquisite nelle sessioni di addestramento iniziali: la nuova conoscenza può soppiantare la conoscenza già appresa, e questo fenomeno prende il nome di *catastrophic forgetting*. Attualmente, diversi studi concentrano i propri sforzi sul risolvere alcune problematiche intrinseche nell'Addestramento Continuo, come il *catastrophic forgetting*[17],[18],[20],[26] o riguardanti la realizzazione in contesti reali di produzione di infrastrutture di continual learning[8], mentre ulteriori studi hanno analizzato l'applicabilità in altri contesti come la *Human Activity Recognition*[15] o l'acquisizione di informazioni dai social network[23]. Sono rari gli studi sullo *Human State Monitoring* e ancora non è stato realizzato un confronto fra le tecniche di Addestramento Continuo di questa tesi.

L'obiettivo è dunque mettere a confronto alcune metodologie di Addestramento Continuo su dati relativi allo *Human State Monitoring*.

Obiettivi

L'obiettivo principale di questa tesi è studiare, attraverso una serie di esperimenti, le performance raggiunte da alcune metodologie di Apprendimento Automatico Continuo applicate a Reti Neurali Ricorrenti riguardo problemi che rientrano nell'ambito dello *Human State Monitoring*. Per poter realizzare questo obiettivo sono stati individuati alcuni sotto-obiettivi necessari alla realizzazione degli esperimenti e della comparazione.

- **Individuare dataset contenenti dati realitivi allo *Human State Monitoring***
  Per poter valutare la sinergia tra continual learning e *Human State Monitoring*, inanzitutto si rende necessaria la raccolta di dati riguardanti lo HSM. Ne sono stati individuati due, scelti per le caratteristiche che verranno discusse in seguito: WESAD[29] e ASCERTAIN[30].

- **Selezionare le metriche e le informazioni utili alla comparazione delle varie metodologie**
  Comparare le varie metodologie significa avere a disposizione per ognuna di esse delle metriche che vadano a misurare i loro vari aspetti, come il tempo necessario all'addestramento, l'accuratezza raggiunta, la conoscenza trasferita e l'impatto a livello di memoria.

- **Scegliere una *baseline***
  Cioè una metodologia base da cui partire e che sia utile a confrontare tutti gli altri. Come sarà specificato in seguito, la metodologia scelta come baseline sarà quella *offline*: ciò consiste nell'addestramento di una rete neurale usando l'intero training set a disposizione senza usare approcci continual.

- **Definire gli strumenti tecnologici**
  Si rende necessario individuare un ambiente in cui poter eseguire le computazioni necessarie all'addestramento, identificare le API e i linguaggi di programmazione adatti allo scopo e produrre del codice verificabile e degli esperimenti replicabili.

- **Confrontare i risultati e trarre le conclusioni**
  Una volta raccolti tutti i dati necessari, essi verranno confrontati per poter finalmente valutare la sinergia fra continual learning e Human State Monitoring.

Risultati

Gli esperimenti e i risultati prodotti nell'ambito di questa tesi mostreranno quanto le tecniche di Addestramento Continuo siano applicabili a contesti di *Human State Monitoring* con risultati anche comparabili all'apprendimento classico *offline*. In particolare, molte delle tecniche di addestramento continuo adottate riescono a ottenere risultati inferiori ma comunque buoni quando applicate alla medesima struttura di rete neurale utilizzata nell'addestramento offline, con misure di trasferimento della conoscenza spesso anche molto positive.

TEACHING

Questa tesi è stata svolta nell'ambito del progetto TEACHING[3].
TEACHING è un progetto finanziato dall'Unione Europea volto a progettare una piattaforma informatica ed il relativo toolkit software a supporto dello sviluppo e del rilascio di applicazioni autonome, adattive e affidabili, consentendo a tali applicazioni di sfruttare il feedback umano per guidare, ottimizzare e personalizzare i servizi offerti.

Struttura della tesi

Nel capitolo 1 verranno definite nel dettaglio le conoscenze preliminari necessarie per affrontare gli argomenti di questa tesi. Nei due capitoli successivi verranno presentati, rispettivamente, le due raccolte di dati utilizzate negli esperimenti (capitolo 2) e gli approcci di addestramento continuo messe a confronto (capitolo 3). Nel capitolo 4 vengono presentati nel dettaglio i risultati prodotti nell'ambito della tesi. Infine, nel capitolo 5, sono delineate le conclusioni tratte a partire da tali risultati.

CAPITOLO 1

Preliminari

In questo capitolo verranno presentati alcuni degli argomenti preliminari necessari alla realizzazione della tesi.
Nella sezione 1.1 viene delineato il contesto riguardante l'ambito del Machine Learning, presentando gli argomenti di questo campo di studi che riguardano questa tesi. Nella sezione 1.2 si presenta l'attuale stato dell'arte riguardo le Reti Neurali Ricorrenti, gli strumenti a disposizione e alcune future possibili applicazioni degli argomenti trattati in questa tesi.

1.1 Contesto

In un mondo sempre più interconnesso e in rapida evoluzione, gli approcci statici al Machine Learning possono diventare impraticabili quando si tratta di produrre sistemi in grado di adattarsi dinamicamente allo stato fisico e mentale degli utenti che vi interagiscono. Inoltre, con l'ampia diffusione di dispositivi in grado di raccogliere in tempo reale informazioni sulla condizione attuale degli utenti, sono costantemente resi a disposizione nuovi dati con cui perfezionare i modelli di Machine Learning in uso attraverso tecniche di Addestramento Continuo.

1.1.1 Human State Monitoring

Uno di questi contesti è lo *Human State Monitoring*. Con questo termine si indica l'obiettivo di un sistema di raccogliere dati riguardanti l'attuale stato psico-fisico di un individuo e trarre conclusioni sulla sua condizione attuale, ad esempio interpretando un battito cardiaco elevato insieme ad un alto livello di sudorazione come uno stato di stress, o determinati movimenti facciali come una condizione

di sonnolenza. L'avere a disposizione metodologie efficienti nel catalogare queste condizioni umane può portare a sistemi responsivi e adattivi, che modificano le proprie interfacce o i propri comportamenti accomodando lo stato psico-fisico attuale dell'utente.

Un possibile applicazione dello *Human State Monitoring* la si può trovare nel progetto europeo TEACHING[3], che mira alla produzione di un toolkit per costruire applicazioni autonome efficienti che facciano uso di intelligenza simile a quella umana. L'obiettivo di questo toolkit è supportare lo sviluppo e il rilascio di applicazioni che possano sfruttare il feedback umano per ottimizzare, personalizzare e guidare il modo in cui esse forniscono i propri servizi. In altre parole, applicazioni che sfruttino lo *Human State Monitoring* per adattarsi dinamicamente alle condizioni dell'utente in tempo reale.
Per poter realizzare questo adattamento continuo è necessario, oltre al flusso di dati in tempo reale sullo stato corrente dell'utente, anche un sistema che sappia efficientemente e correttamente interpretare questi dati, traendo conclusioni sulle attuali condizioni della persona. Questa situazione presenta le classiche caratteristiche necessarie all'applicazione del Machine Learning: abbiamo dati raccolti da sensori che possono essere difettosi, per cui i dati possono contenere incertezze o essere rumorosi o incompleti, ed essendo in esame lo stato umano è molto difficile, se non impossibile, formalizzare attraverso una descrizione procedurale la classificazione di questi dati in categorie comportamentali, ad esempio affaticamento o sonnolenza.

1.1.2 Addestramento

L'addestramento di un modello di Machine Learning avviene attraverso l'utilizzo di un insieme di dati chiamato *training set*, che contiene i dati di addestramento. Questi dati sono sottoposti al modello, che li userà nelle sessioni di addestramento. Nell'addestramento supervisionato, il training set è composto da una serie di esempi composti da coppie $\langle$input, etichetta$\rangle$ dove le etichette sono fornite da chi ha preparato il training set, detto supervisore. Queste etichette vengono assegnate agli input in base ad una certa funzione f, che è ignota ed è la funzione che il modello di Machine Learning deve apprendere. Il modello elabora questi dati e le loro etichette cercando di apprendere quali etichette sono assegnate a quali dati e in che modo. L'obiettivo del modello è quindi trovare una buona approssimazione di f che faccia corrispondere le etichette corrette ai rispettivi input.
Per verificare l'apprendimento, si sottopone al modello addestrato dei dati non etichettati, denominati *test set*, e si verificano le etichette che il modello assegna a tali dati controllando se ha classificato correttamente o meno. La percentuale di dati correttamente etichettati dal modello è chiamata **accuratezza**.

1.1.3 Loss

Per misurare l'andamento dell'apprendimento di un modello di Machine Learning, vengono usate funzioni che vanno a valutare il modello in base alle risposte che esso fornisce sui dati etichettati, confrontandole con le etichette reali. Queste funzioni prendono il nome di funzioni di *loss*. Ne esistono di diversi tipi, a seconda del problema in esame: ad esempio, nei problemi dove ad ogni input va assegnata una risposta si/no (classificazione binaria) una loss molto usata è la cross-entropia binaria (*binary crossentropy*), delineata nell'equazione 1.1 dove y è l'etichetta reale, che può assumere valori 0/1, falso/vero o altri, e $p(y)$ è la probabilità assegnata dal modello in fase di previsione.

$$H_p(q) = -\frac{1}{N}\sum_{i=1}^{N}(y_i \cdot \log(p(y_i)) + (1 - y_i) \cdot \log(1 - p(y_i))) \tag{1.1}$$

1.1.4 Reti Neurali

In letteratura vengono studiati moltissimi modelli di Machine Learning. Si va dai più semplici come i modelli lineari, nei quali abbiamo un'equazione nella forma $y = w_0x + w_1$ e dobbiamo apprendere i w_0 e w_1 che ai dati x fanno corrispondere le corrette etichette y, alle *Support Vector Machine*, che cercano di trovare dei confini di divisione fra insiemi di dati così da poterli classificare.
Un'ampia area di modelli di Machine Learning è quella delle cosiddette **reti neurali**, studiate sin dagli anni '40. Esse prendono liberamente spunto dalla struttura dei neuroni del cervello umano (Figura 1.1), che sono stati schematizzati in strutture matematiche denominate **perceptron**[27]. In un perceptron (Figura 1.2), ad ogni

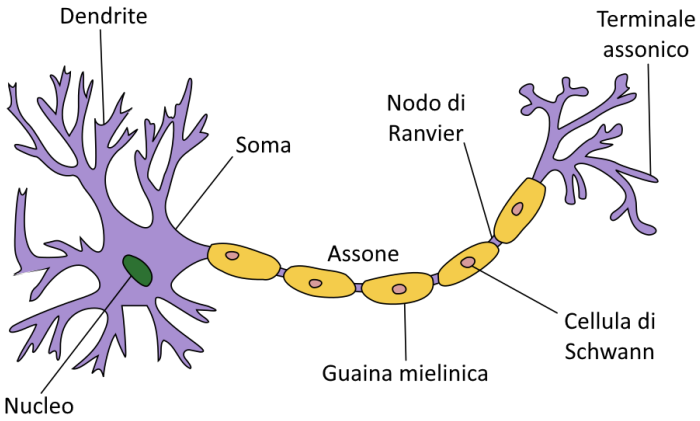

Figura 1.1: *Neurone (credits: Quasar, Wikipedia)*

input x_i è assegnato un peso w_i che lo rende più o meno importante nel contribuire al risultato y, che viene calcolato dal nodo attraverso la **funzione di attivazione**, nella forma 1.2 dove b è un ulteriore peso definito *bias*.

$$f\left(\sum_i w_i \cdot x_i + b\right) \tag{1.2}$$

Una rete neurale (Figura 1.3) è composta da un insieme di perceptron collegati fra

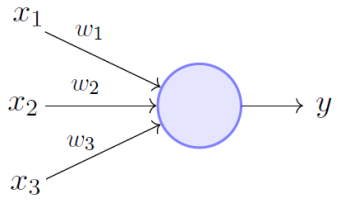

Figura 1.2: *Perceptron*

loro suddivisi su livelli, o *layer*: una rete neurale composta da un elevato numero di layer è detta profonda, o *deep neural network*, e i layer interni sono detti layer nascosti.

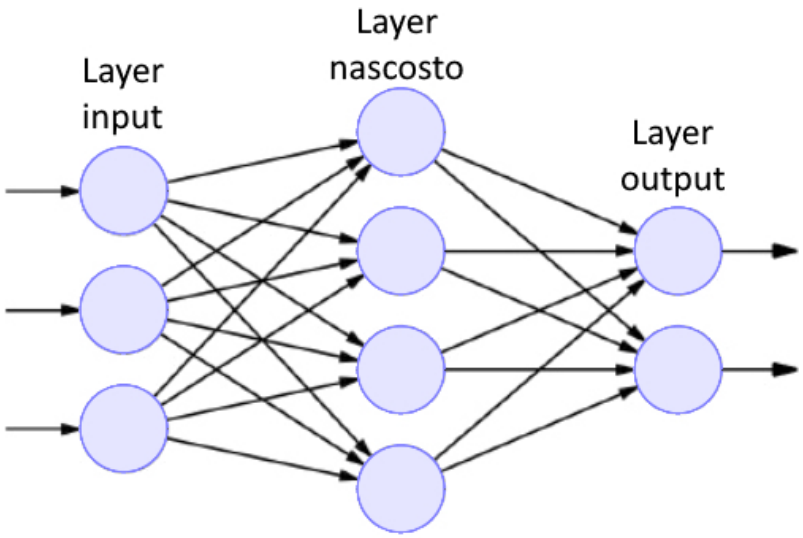

Figura 1.3: *Esempio di rete neurale (credits: databricks.com)*

Le funzioni di attivazione possono essere funzioni di qualsiasi tipo, ma le più usate e studiate sono funzioni specifiche come ReLU e softmax. La funzione ReLU (*Rectified Linear Unit*) è molto semplice e veloce da calcolare: restituisce la parte positiva del proprio input, e consente un migliore apprendimento nelle reti neurali profonde[10].

$$ReLU(x) = max(0, x) \tag{1.3}$$

La funzione softmax, invece, prende in input un vettore di K valori e lo comprime in un vettore sempre di K elementi ma dai valori compresi nell'intervallo $(0,1)$ e la cui somma è 1. Vengono usate nel layer finale delle reti neurali realizzate per compiti di classificazione: ogni valore $\sigma(\mathbf{z})_j$ è interpretabile come la probabilità che l'input appartenga alla classe j fra le K possibili.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \quad \text{per } j = 1, \ldots, K \tag{1.4}$$

Le funzioni di attivazione descrivono quindi il comportamento dei nodi di una rete neurale e la scelta della funzione di attivazione è un elemento importante nella strutturazione di una rete neurale.

Tipologie

La struttura interna di una rete neurale influenza particolarmente i risultati che essa può ottenere. A seconda del problema che si vuole risolvere, si può rendere necessario l'utilizzo di reti neurali con strutture interne complesse, in modo da ottenere risultati soddisfacenti. In particolare, in letteratura attualmente alcune delle strutture più studiate sono (Figura 1.4):

- Reti Feed Forward (FF)[12]
- Reti Neurali Ricorrenti (RNN, Recurrent Neural Networks)[13]
- Auto Encoders (AE)[4]
- Reti Profonde Convolutive (DCN, Deep Convolutional Networks)[2]
- Echo State Networks (ESN)[9]

Alcune di queste strutture ottengono risultati particolarmente buoni su certe tipologie di problemi: ad esempio, le DCN possiedono una struttura ispirata alla corteccia visiva del cervello e sono particolarmente usate nel campo del riconoscimento delle immagini e del linguaggio parlato, mentre le RNN mantengono informazioni riguardo gli input precedenti e consentono di modellare i comportamenti dinamici e ricorrenti delle sequenze temporali.

Backpropagation

La retropropagazione dell'errore, *backward propagation of errors*[28], è l'algoritmo più diffuso per l'addestramento supervisionato delle reti neurali. Questo algoritmo consiste di due fasi:

- *Forward phase*, dove gli input vengono elaborati dalla rete neurale fino a produrre un output

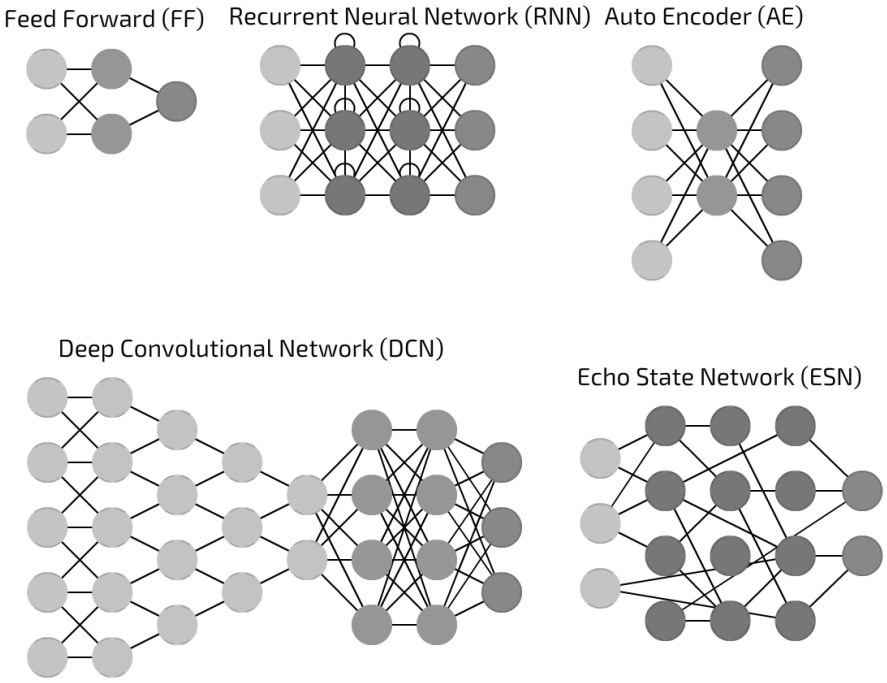

Figura 1.4: *Esempi di reti neurali (credits: Andrew Tch, towardsdatascience.com)*

- *Backward phase*, dove l'output prodotto è confrontato con la vera etichetta assegnata e, attraverso calcoli differenziali, vengono individuati i contributi all'errore di ogni nodo. Con questi contributi, attraverso un algoritmo di ottimizzazione (solitamente, la discesa di gradiente), vengono modificati i pesi delle connessione fra i nodi, partendo dai nodi output e risalendo la rete fino ai nodi input.

Questo algoritmo richiede quindi che le funzioni di attivazione dei nodi siano differenziabili, e può soffrire di un problema che prende il nome di *vanishing/exploding gradient problem*[22], o problema della scomparsa/esplosione del gradiente. Nel dettaglio, attraverso questo algoritmo ogni parametro del modello viene modificato, durante la *backward phase*, in maniera proporzionale alla derivata parziale della funzione *loss* rispetto al parametro stesso. Propagando all'indietro attraverso la regola della catena(1.5) dei gradienti compresi nell'intervallo $[0, 1]$, come nel caso di alcune funzioni di attivazione, il prodotto diventa tanto più piccolo quanto più è profonda la rete neurale o, viceversa, esplodere in caso di gradienti dai valori elevati.

$$D\left[f(g(x))\right] = f'(g(x)) \cdot g'(x) \tag{1.5}$$

Questo problema viene mitigato dall'uso di funzioni di attivazioni diverse, come le già citate ReLU(1.3) e softmax(1.4), o dall'uso di algoritmi di addestramento che fanno uso solo del segno del gradiente e non della sua norma, come l'algoritmo Rprop[24].

Nelle reti neurali ricorrenti, un grande contributo alla mitigazione del problema è avvenuto grazie all'introduzione delle LSTM, *Long Short-Term Memory*[13], che mantengono informazioni sullo stato del modello senza propagarle attraverso funzioni non lineari.

1.1.5 Continual Learning

L'addestramento di un modello di Machine Learning attraverso un *training set* disponibile fin da subito e fornito interamente al modello è denominato "addestramento *offline*", o *offline training*. Esistono contesti, però, dove i dati necessari o utili all'addestramento non sono subito tutti disponibili, ma diventano tali col passare del tempo. Un esempio di questa situazione sono i sistemi di raccomandazione in servizi come Netflix o YouTube: essi imparano continuamente i gusti dell'utente, in modo da fornire suggerimenti su video e film sempre adatti ai gusti del momento. Questa situazione, in cui i dati di addestramento diventano disponibili col passare del tempo e cioè di addestramento continuativo, è detta "Addestramento Continuo", o *continual learning*.
Nel caso dell'Addestramento Continuo, un grosso ostacolo è quello denominato *catastrophic forgetting*[21]: quando una rete neurale già addestrata viene addestrata su nuovi dati, essa tenderà a dimenticare ciò che ha appreso sui dati precedenti. Questa situazione è una manifestazione del cosiddetto dilemma della **plasticità-stabilità**: si ha stabilità quando la nuova conoscenza non interferisce con la vecchia conoscenza, mentre si parla di plasticità quando la nuova conoscenza migliora la conoscenza già appresa e viceversa. Questi due obiettivi sono in contrapposizione fra loro e mantenere un buon compromesso fra i due è uno degli obiettivi chiave degli algoritmi e attuale campo di studio.

In un mondo in continua e rapida evoluzione l'Addestramento Continuo diventa un approccio sempre più necessario, e apporta al Machine Learning un cambiamento di paradigma al pari dell'introduzione della filosofia Agile[31] nel mondo dell'ingegneria del software: mentre in molti dei modelli di Machine Learning correnti l'addestramento è eseguito da zero e il modello viene utilizzato così com'è, con un procedimento comparabile ai primi approcci "a cascata" adottati nello sviluppo dei software, l'Addestramento Continuo fornisce a questo campo la possibilità di produrre modelli che vengono migliorati iterativamente ogni volta che è necessario senza dover addestrare un nuovo modello da zero. "Il parallelismo sta nell'equiparare i dati ai requisiti (arrivano in maniera continuativa e cambiano nel tempo) e l'addestramento alla fase di design e sviluppo che porta al prodotto software (che nel nostro caso è la funzione di predizione realizzata dal modello di Machine Learning)."[19]

1.2 Stato dell'Arte

1.2.1 Addestramento Continuo

L'ambito dell'Addestramento Continuo è in continua evoluzione. Negli ultimi anni, studi come *Learning without Forgetting*[18] o *Overcoming catastrophic forgetting in neural networks*[17] hanno affrontato il problema del *catastrophic forgetting* proponendo soluzioni da adottare durante l'addestramento delle reti neurali. In *Learning without Forgetting* viene proposto l'omonimo approccio, LWF, che non usa dati relativi alla conoscenza già appresa ma solo i nuovi dati relativi alla nuova conoscenza da apprendere, introducendo parametri del modello specifici per i nuovi dati ma che abbiano performance non impattanti sulla conoscenza già presente. Nel secondo studio viene proposto un algoritmo che prende il nome di *Elastic Weight Consolidation*, o EWC, che interpreta i parametri da apprendere della rete neurale come spazi probabilistici, e rende meno probabile la modifica di quei parametri che risultano importanti per la conoscenza già appresa quando si addestra su nuova conoscenza. Ulteriori studi hanno proposto approcci definiti *reharsal* o di replay[20],[26]. Queste tecniche consistono nel mantenere in memoria, all'interno di buffer, degli esempi della conoscenza passata selezionati secondo qualche politica. Durante le fasi di addestramento successive, ai nuovi dati vengono intervallati gli esempi mantenuti in memoria, così da consentire al modello di "ricordare" la conoscenza passata e mantenerla, mitigando così il *catastrophic forgetting*.

Altri studi hanno proposto metriche per misurare le performance dell'Apprendimento Continuo e l'impatto che questo approccio ha sulla vecchia e nuova conoscenza, come in *Gradient Episodic Memory for Continual Learning*[20] dove vengono proposte metriche per misurare il *forward transfer*, FWT, e il *backward transfer*, BWT: con queste due metriche si misura l'impatto che la nuova conoscenza ha sulla conoscenza, rispettivamente, già appresa e su quella che si andrà ad apprendere.

Altri recenti studi nell'ambito del continual learning si sono concentrati, ad esempio, sull'applicabilità pratica e l'ideazione di un'architettura in grado di manutenere modelli di Machine Learning in produzione e gestire la dinamicità e i cambiamenti sui dati[8], oppure in ambiti come la *human activity recognition* (HAR)[15], che riguarda il riconoscimento delle attività quotidiane delle persone attraverso dei sensori, o ancora l'acquisizione di informazioni dai social media per gestire al meglio situazioni di crisi[23].
In tutti questi ambiti l'Addestramento Continuo si rivela fondamentale per adattare dinamicamente i modelli di Machine Learning al mutevole ambiente reale, senza la necessità di addestrare continuamente nuovi modelli da zero con l'ulteriore difficoltà costituita dal memorizzare dati anche molto vecchi portando a dataset di

dimensioni elevate che aumentano il tempo necessario all'addestramento. Grazie all'Addestramento Continuo, invece, è possibile mantenere solo una parte degli esempi della conoscenza precedente, negli approcci cosiddetti di *reharsal* o *replay*, o non mantenerne affatto.

In molti studi, l'approccio all'Addestramento Continuo è stato suddiviso in tre scenari fondamentali[32],[33]. Viene presa in considerazione una rete neurale che deve apprendere in maniera sequenziale una serie di task, dove con task si indica una qualsiasi attività che la rete neurale deve svolgere, ad esempio classificare le razze dei cani che appaiono in un immagine:

- **Task-IL**, o *task incremental*: viene indicato il task a cui appartiene il problema, tra i task già appresi dal modello, e viene chiesto di risolverlo

- **Domain-IL**, o *domain incremental*: viene chiesto di risolvere il problema senza indicare il task a cui appartiene tra quelli appresi

- **Class-IL**, o *class incremental*: viene chiesto di risolvere il problema e anche di identificare a quale task esso appartiene.

Un dataset molto usato nell'ambito del Machine Learning è il MNIST, un database di cifre manoscritte che viene usato per valutare modelli, approcci e algoritmi riguardanti l'addestramento e il comportamento dei modelli di classificazione delle immagini (Figura 1.5).

Figura 1.5: *Esempio dei dati del MNIST (credits: Josef Steppan, Wikipedia)*

Per meglio comprendere i tre scenari proposti, i task relativi al dataset MNIST possono essere individuati nei seguenti:

- Task 1: identificare tra 0 e 1
- Task 2: identificare tra 2 e 3
- Task 3: identificare tra 4 e 5
- Task 4: identificare tra 6 e 7
- Task 5: identificare tra 8 e 9

Con questi task, i tre scenari individuati diventano:

- **Task-IL**: dato il task, è la prima o la seconda classe? Ad esempio, identificare fra 0 e 1
- **Domain-IL**: senza sapere il task, è la prima o la seconda classe? Ad esempio, identificare se è un numero pari (prima classe) o dispari (seconda classe)
- **Class-IL**: senza sapere il task, che cifra è? Nell'esempio, individuare quale cifra è da 0 a 9

1.2.2 Reti Neurali Ricorrenti

Una rete neurale ricorrente, o RNN, è una rete neurale dove i neuroni, o alcuni di essi, sono collegati a sé stessi in un ciclo chiuso. Questo collegamento consente ai layer di una RNN di mantenere informazioni in maniera analoga ad uno stato interno, permettendo di modellare, ad esempio, i comportamenti dinamici di una sequenza temporale fornita come input, dove ogni dato dipende anche dai dati precedenti già ricevuti.
In letteratura esistono diverse architetture di reti neurali ricorrenti, ma molte di esse devono la loro esistenza a due nodi ricorrenti fondamentali: le *Long Short-Term Memory*, LSTM[13], e le *Gated Recurrent Unit*, GRU[6] (Figura 1.6). LSTM e GRU

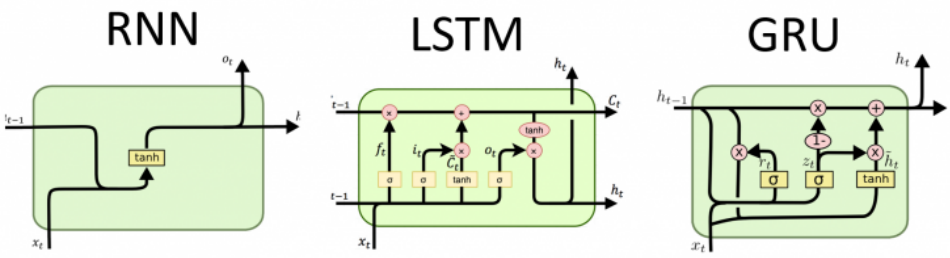

Figura 1.6: *Schemi di nodi ricorrenti (credits: Michael Phi, towardsdatascience.com)*

hanno meccanismi interni, chiamati *gates* o cancelli, che regolano il flusso delle informazioni. Attraverso questi *gate*, le unità apprendono quali dati sono importanti all'interno della sequenza e quali si possono ignorare, lasciando così al resto della rete solamente le informazioni che ritiene essere rilevanti per fare predizioni.

In una LSTM, i dati passano attraverso il *forget gate*, che decide immediatamente quali informazioni è utile mantenere e quali ignorare. Dopodiché i dati passano attraverso l'*input gate* assieme allo stato precedente dell'unità, che decide quali valori dello stato interno conviene aggiornare, e queste informazioni sono poi usate per aggiornare lo stato dell'unità. Infine, attraverso l'*output gate* viene calcolato il nuovo stato interno dell'unità, che servirà all'input successivo.

Nelle GRU abbiamo un comportamento simile, ma la struttura dell'unità è semplificata. I dati passano subito attraverso il *update gate*, che si comporta in modo simile ai *gate forget* e *input* dell'LSTM, decidendo quali informazioni ignorare e quali mantenere. Dopodiché si attraversa il *reset gate*, che decide cosa mantenere delle informazioni precedententemente analizzate. Avendo una struttura più semplificata, la GRU sono solitamente più veloci delle LSTM.

1.2.3 Software e Hardware

Negli ultimi anni sono nate diverse librerie software che consentono di realizzare modelli di Machine Learning performanti in breve tempo dei tipi più disparati, da semplici modelli lineari a reti neurali profonde convolutive e oltre.
Librerie open source come PyTorch, Tensorflow[1] e Keras, ma anche librerie proprietarie come Apple Core ML, IBM Watson e MATLAB Deep Learning Toolbox, hanno aperto le porte del Machine Learning a chiunque sia disposto a studiarle. Alcune librerie, come in particolare TensorFlow 2.0, possiedono API estremamente semplici che consentono di eseguire il preprocessing dei dataset, strutturare una rete neurale, addestrarla e valutarla con poche di righe di codice, e nonostante la semplicità essa consente comunque di mettere mano e andare a personalizzare ogni minimo aspetto di tutte le fasi, dal preprocessing al training. In letteratura sono disponibili anche moltissimi testi che consentono di approcciarsi facilmente al Machine Learning[7],[11].

Anche a livello hardware gli ultimi anni hanno portato enormi sviluppi, grazie soprattutto all'industria videoludica e alle criptovalute che hanno spinto la ricerca e lo sviluppo di processori specializzati in calcolo parallelo e matriciale. Questi dispositivi, denominati *Graphics Processing Unit* o GPU, consentono di ridurre esponenzialmente i tempi di addestramento delle reti neurali, abbassando ancora di più i requisiti per accedere a questo campo di studi e consentendo l'esecuzione

di modelli di Machine Learning anche su un comune computer casalingo.
Inoltre, aziende come Google hanno sviluppato hardware realizzato ad hoc per le computazioni necessarie alle reti neurali, incrementando così ulteriormente la velocità e l'efficienza di questi modelli. Nel caso di Google, questo hardware è chiamato *Tensor Processing Unit* o TPU.

1.2.4 Applicazioni Future

Alla luce degli attuali studi nell'ampio campo che è il Machine Learning, e in particolare riguardanti il continual learning e lo HSM, diventa chiaro come il futuro richiederà sempre più responsività ai sistemi predittivi e un sempre maggior grado di adattamento allo stato psico-fisico del momento dell'utente. Questa necessità di continua evoluzione ben si sposa con le caratteristiche del continual learning, e possibili applicazioni di questa sinergia continual learning – HSM si possono individuare in diversi contesti, tra cui ad esempio:

- **Guida autonoma**.
  La crescente industria dei veicoli a guida autonoma ha portato ad uno sviluppo in moltissimi campi del Machine Learning[14], in particolare nel campo dell'*object recognition*[5], del *behavioral planning*[35] e altri ancora. L'*autonomous driving* negli ultimi anni è, senza dubbio, uno dei motori principali nello sviluppo della disciplina del Machine Learning.
  Un veicolo a guida autonoma che sappia adattarsi allo stato psico-fisico attuale del conducente può portare a grossi benefici: ad esempio, essere in grado di modificare il proprio andamento per accomodare uno stato di malessere mitigando così il rischio di incidenti, o poter proporre tappe durante il percorso a seconda della condizione dei passeggeri portando così a più piacevole utilizzo del sistema da parte degli utenti.

- **Intrattenimento**.
  L'industria dell'intrattenimento si è sempre interessata alle nuove tecnologie, puntando alla novità del momento in modo da raggiungere un pubblico sempre più ampio: ad esempio, la tecnologia degli schermi 3D o della realtà virtuale.
  Un sistema adattivo di riconoscimento dello stato psico-fisico dello spettatore o del videogiocatore può portare a esperienze multimediali più realistici e responsivi, che ad esempio possano adattare la propria narrazione o l'atmosfera proposta all'utente in maniera da sfruttare, o alleviare, eventuali emozioni che nascono nello spettatore.

In tutti questi ambiti, per poter ottenere ottimi risultati, i sistemi devono essere in grado di adattarsi e specializzarsi nel riconoscere lo stato psico-fisico del singolo utente (ad esempio, il conducente proprietario del veicolo o il videogiocatore che

possiede la console) ottenendo così sistemi calibrati e personalizzati in base alle caratteristiche e alle esigenze del singolo utente.

Questa sinergia risulta ancora poco studiata in letteratura, quindi può essere utile mettere sotto esame alcune tecniche di Addestramento Continuo attualmente studiate per giudicare la loro applicabilità o meno in contesti di *Human State Monitoring*, così da concludere se siano necessari ulteriori sforzi in questa direzione per poter realizzare, in futuro, sistemi sempre più integrati con l'uomo e interfacce uomo-macchina sempre più intuitive e adattive. Valutare fin da subito l'applicabilità di questa sinergia è fondamentale in un campo di studi che si muove così in fretta come quello del Machine Learning.

CAPITOLO 2

Dataset

Ai fini della tesi, il primo passo necessario per poter valutare correttamente la sinergia Apprendimento Continuo – Human State Monitoring è ottenere raccolte di dati riguardanti quest'ultimo. I dati dovrebbero provenire da più soggetti, per correttezza statistica, e riguardare diversi aspetti biometrici della persona: battito cardiaco, sudorazione, respirazione, ecc.

I dataset selezionati ai fini di questa comparazione sono due, presentati nel dettaglio di seguito. In particolare, nella sezione 2.1 viene introdotto il dataset WESAD e il preprocessing eseguito nella sezione 2.1.1, mentre nella sezione 2.2 introduciamo il dataset ASCERTAIN, con il preprocessing dettagliato all'interno della sezione 2.2.1. Infine, nella sezione 2.3, vengono presentate le principali differenze fra i due dataset precedentemente descritti.

2.1 WESAD

Il dataset WESAD[29], acronimo di *WEarable Stress and Affect Detection*, contiene dati raccolti da 15 soggetti durante uno studio effettuato in laboratorio riguardante i livelli di stress misurati tramite sensori biometrici e di movimento indossabili. Il dataset classifica questi dati in tre scale: neutralità, stress e divertimento.
I dispositivi usati per la raccolta dei dati sono due: uno indossato sul petto (RespiBAN) e uno indossato sul polso (Empatica E4).

Il RespiBAN, indossato sul petto, raccoglie dati campionati a 700 Hz su:

- Elettrocardiogramma (ECG)
- Attività elettrodermica (EDA)
- Elettromiogramma (EMG)
- Respirazione
- Temperatura corporea
- Accelerazione sui tre assi

L'Empatica E4, indossato sul polso, monitora con frequenza di campionamento eterogenea:

- Pulsazioni (BVP), a 64Hz
- Attività elettrodermica (EDA), a 4 Hz
- Temperatura corporea, a 4 Hz
- Accelerazione sui tre assi, a 32 Hz

La quantità di dati contenuti in questo dataset lo rende un ottimo strumento nell'addestramento di modelli di Machine Learning riguardanti l'ambiente dello HSM. Dati come le pulsazioni, la temperatura e l'elettrocardiogramma sono precisissimi indicatori biometrici dello stato psico-fisico di una persona e pertanto sono estremamente utili alle reti neurali per dedurre lo stato psico-fisico di soggetto.

2.1.1 Preprocessing

Una volta caricato il dataset, i dati sono stati concatenati e ricampionati a 32 Hz, attraverso la libreria `SciPy`.

```
31    X = np.concatenate([
32        scipy.signal.resample(ds[s]['signal']['chest']['ACC'],
    len(ds[s]['signal']['wrist']['ACC'])),
33        scipy.signal.resample(ds[s]['signal']['chest']['EDA'],
    len(ds[s]['signal']['wrist']['ACC'])),
34        scipy.signal.resample(ds[s]['signal']['chest']['EMG'],
    len(ds[s]['signal']['wrist']['ACC'])),
35        scipy.signal.resample(ds[s]['signal']['chest']['ECG'],
    len(ds[s]['signal']['wrist']['ACC'])),
36        scipy.signal.resample(ds[s]['signal']['chest']['Resp'],
     len(ds[s]['signal']['wrist']['ACC'])),
37        scipy.signal.resample(ds[s]['signal']['chest']['Temp'],
     len(ds[s]['signal']['wrist']['ACC'])),
```

```
38          scipy.signal.resample(ds[s]['signal']['wrist']['ACC'],
    len(ds[s]['signal']['wrist']['ACC'])),
39          scipy.signal.resample(ds[s]['signal']['wrist']['BVP'],
    len(ds[s]['signal']['wrist']['ACC'])),
40          scipy.signal.resample(ds[s]['signal']['wrist']['EDA'],
    len(ds[s]['signal']['wrist']['ACC'])),
41          scipy.signal.resample(ds[s]['signal']['wrist']['TEMP'],
     len(ds[s]['signal']['wrist']['ACC']))
42          ], axis = 1)
```

Dopodiché, l'insieme delle features è stato standardizzato ponendo media pari a 0 e deviazione standard uguale a 1

```
46      X = (X - X.mean(axis = 0)) / X.std(axis = 0)
```

Sono state ricampionate le etichette e sono state rimosse l'etichetta 0, corrispondente ad una condizione di neutralità, e le etichette 5, 6 e 7 poichè secondo le indicazioni del dataset sono corrispondenti a dati da ignorare.

```
50      Y = scipy.signal.resample(ds[s]['label'], len(ds[s]['signal
    ']['wrist']['ACC']))
51      Y = np.around(Y)
52      Y = abs(Y.astype(np.int32))

56      X = X[(Y>0) & (Y<5)]
57      Y = Y[(Y>0) & (Y<5)]

86      Y = np.array(SubsequencesY, dtype = np.float32) - 1
87      y_WES = keras.utils.to_categorical(Y, num_classes = 4)
```

I dati sono poi raccolti in sottosequenze di 100 punti relativi alla stessa etichetta, corrispondenti a circa 3 secondi, e per ogni etichetta vengono estratte 100 di queste sottosequenze.

```
62      count = 0
63      prev = Y[0]
64      LenSubsequences = []
65      for elem in Y:
66          if(elem != prev):
67              LenSubsequences.append(count)
68              count = 0
69          count += 1
70          prev = elem
71      SubsequencesX = []
72      SubsequencesY = []
73      i = 0
74      for elem in LenSubsequences:
75          for j in range(0, elem, 100):
76              if(j+100 <= elem):
```

```
77                  SubsequencesX.append(X[i+j:i+j+100])
78                  SubsequencesY.append(Y[i+j+50])
79          i += elem
80
81      assert len(SubsequencesX) == len(SubsequencesY)
82      X_WES = (np.array(SubsequencesX, dtype = np.float64)).
    reshape(-1, 100, 14)
```

Dai dati viene poi estratto un test set. Questo ci lascia con un test set di 1500 elementi e un training set da 4500 elementi suddivisi come specificato nella tabella 2.1.

2.2 ASCERTAIN

ASCERTAIN[34], acronimo di *multimodal databASe for impliCit pERsonaliTy and Affect recognitIoN*, è un dataset contenente dati provenienti da 58 soggetti raccolti con sensori fisiologici commerciali e classifica le informazioni su diverse scale: incitamento, valenza, investimento, apprezzamento e familiarità. Il dataset contiene anche dati relativi all'attività facciale dei soggetti registrati con sensori comuni, oltre ai dati sull'elettroencefalogramma (EEG), elettrocardiogramma (ECG) e sulle risposte galvaniche della pelle (GSR).

ASCERTAIN è un dataset contenente molti più dati rispetto a WESAD, raccolti con dispositivi commerciali comuni, il che rende questa raccolta di dati molto utile per poter valutare l'applicazione di metodologie di Apprendimento Continuo. Usando dispositivi commerciali si può simulare un contesto più vicino all'ambiente di produzione che ad un esperimento controllato in laboratorio.

2.2.1 Preprocessing

Dal dataset sono stati esclusi due soggetti poiché i loro dati risultavano imprecisi o incompleti. Ogni soggetto rimanente è stato caricato e per ognuno sono state create le etichette seguendo la seguente suddivisione basata sui valori di incitamento e valenza specificati, che corrispondono ai quadranti del grafico cartesiano creato dai due valori:

- Label 0, corrispondente a valori di incitamento > 3 e valenza > 0
- Label 1, corrispondente a valori di incitamento > 3 e valenza ≤ 0
- Label 2, corrispondente a valori di incitamento ≤ 3 e valenza > 0
- Label 3, corrispondente a valori di incitamento ≤ 3 e valenza ≤ 0

```
40          Y = []
41          for j in range(len(arousal)):
42              if(arousal[j] > 3):
43                  if(valence[j]> 0):
44                      Y.append(0)
45                  elif(valence[j] <= 0):
46                      Y.append(1)
47              elif(arousal[j] <= 3):
48                  if(valence[j] > 0):
49                      Y.append(2)
50                  elif(valence[j] <= 0):
51                      Y.append(3)
```

I dati sono poi ripuliti dai valori mancanti, ricampionati a 32 Hz e concatenati fra loro.

```
55          for clip in clips_ECG.keys():
56              clips_ECG[clip] = clips_ECG[clip][~np.isnan(
    clips_ECG[clip]).any(axis = 1)]
57              clips_EEG[clip] = clips_EEG[clip][~np.isnan(
    clips_EEG[clip]).any(axis = 1)]
58              clips_GSR[clip] = clips_GSR[clip][~np.isnan(
    clips_GSR[clip]).any(axis = 1)]

62          for clip in clips_ECG.keys():
63              clips_ECG[clip] = scipy.signal.resample(clips_ECG[
    clip], len(clips_EEG[clip]))
64              clips_GSR[clip]= scipy.signal.resample(clips_GSR[
    clip], len(clips_EEG[clip]))

68          clips = {}
69          j = 0
70          for clip in clips_ECG.keys():
71              clips['Clip' + str(j)] = np.concatenate([
72              clips_ECG[clip],
73              clips_EEG[clip],
74              clips_GSR[clip],
75              ], axis = 1)
76              j += 1
```

Vengono poi costruite le sottosequenze, da 160 elementi ciascuna (5 secondi), mantenendo un bilanciamento fra le diverse classi.

```
80          SubsequencesX, SubsequencesY = [], []

82          count_0, count_1, count_2, count_3 = 0, 0, 0, 0
83          j = 0
84          for clip in clips.keys():
```

```
85          if(((Y[j] == 0) & (count_0 < 84)) or ((Y[j] == 1) &
    (count_1 < 84)) or ((Y[j] == 2) & (count_2 < 84)) or ((Y[j]
    == 3) & (count_3 < 84))):
86              length = len(clips[clip])
87              for k in range(0, 10):
88                  SubsequencesX.append(clips[clip][length
    -(160*(k+1)):length-(160*k),:])
89                  if (Y[j] == 0):
90                      count_0 += 1
91                      SubsequencesY.append(0)
92                  elif (Y[j] == 1):
93                      count_1 += 1
94                      SubsequencesY.append(1)
95                  elif (Y[j] == 2):
96                      count_2 += 1
97                      SubsequencesY.append(2)
98                  elif (Y[j] == 3):
99                      count_3 += 1
100                     SubsequencesY.append(3)
101             j += 1
102
103     Xs = (np.array(SubsequencesX, dtype = np.float64)).
    reshape(-1, 160, 17)
104     ys = tf.keras.utils.to_categorical(np.array(
    SubsequencesY, dtype = np.float32), num_classes = 4)
```

Analogamente a WESAD, anche da ASCERTAIN viene estratto un test set. A preprocessing ultimato, il dataset è suddiviso in un test set da 720 elementi e un training set da 2160 elementi suddivisi come indicato nella Tabella 2.2

2.3 Differenze fra i due dataset

Uno degli elementi da considerare quando si addestra una rete neurale è il bilanciamento dei dati presi in considerazione per l'addestramento.
Andando ad analizzare nel dettaglio la struttura dei due dataset considerati, diventa evidente la decisa sovrarappresentanza della classe 0 nel dataset ASCERTAIN (Figura 2.2), mentre le classi nel dataset WESAD risultano molto più bilanciate (Figura 2.1). Questo è anche dovuto all'aver scelto delle classi fittizie, per quanto riguarda ASCERTAIN, che ha evidenziato come non tutti i soggetti abbiano prodotto dati bilanciati rispetto alle nuove classi selezionate. Anche eseguendo un bilanciamento successivo, ovvero eliminando gli esempi in eccesso sulla prima classe, si raggiungono misurazioni analoghe a quelle presentate.

Bilanciando il dataset ASCERTAIN attraverso la produzione di soggetti fittizi, ognu-

Soggetto	Dati
S2	287
S3	287
S4	298
S5	298
S6	297
S7	303
S8	309
S9	292
S10	313
S11	292
S13	308
S14	297
S15	305
S16	313
S17	301
Totale	4500

Tabella 2.1: *Datataset WESAD*

Soggetto	Dati
S0	95
S1	75
S2	148
S3	154
S4	153
S5	208
S6	131
S7	115
S8	119
S9	206
S10	119
S11	78
S12	74
S13	109
S14	72
S15	84
S16	220
Totale	2160

Tabella 2.2: *Datataset ASCERTAIN*

no contenente dati presi da tutti i soggetti in maniera bilanciata rispetto alle nuove classi, otteniamo immediatamente risultati sensibilmente migliori, con accuratezze a volte anche doppie rispetto allo stesso metodologia sul dataset ASCERTAIN originale. Questo rende evidente l'importanza dell'avere dati di addestramento bilanciati, quindi esempi in quantità simile per ognuna delle etichette possibili.

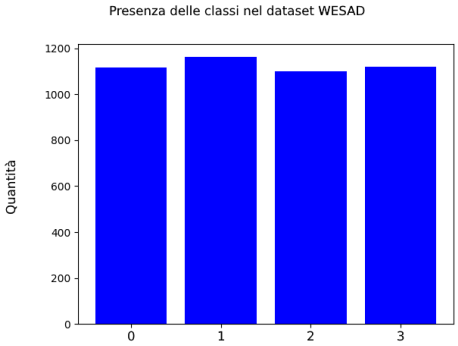

Figura 2.1: *Presenza delle classi su dataset WESAD*

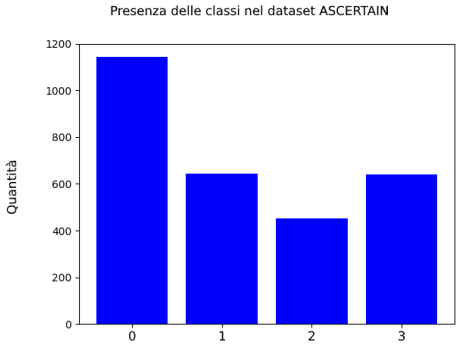

Figura 2.2: *Presenza delle classi su dataset ASCERTAIN*

CAPITOLO 3

Metodologie

In questo capitolo verranno delineate nel dettaglio le varie metodologie scelte per la realizzazione della comparazione fra alcune strategie di Addestramento Continuo triviali applicate ai dataset di Human State Monitoring presentati nel precedente capitolo.

In particolare, nella sezione 3.1 vengono delineate nel dettaglio le metriche prese in considerazione per comparare le metodologie scelte, nella sezione 3.2 viene presentata la metodologia di addestramento utilizzata come base di confronto e infine, nella sezione 3.3, vengono descritte nel dettaglio le metodologie di Addestramento Continuo selezionate nell'ambito di questa tesi.

3.1 Metriche

Per poter confrontare fra loro le diverse strategie, fin da subito è apparsa chiara la necessità di metriche che misurassero con precisione alcune caratteristiche fondamentali delle metodologie di Addestramento Continuo, in particolare: l'accuratezza sul test set, l'accuratezza media sulle classi e i *forward* e *backward transfer* per poter misurare l'impatto della nuova conoscenza sulla conoscenza precedentemente appresa e su quella ancora da acquisire.

3.1.1 Accuratezza

Con "accuratezza" si intende la misura della percentuale di previsioni corrette che il modello di Machine Learning ottiene su un test set, cioè un insieme di dati a cui il modello non è stato esposto durante le sessioni di addestramento.

Per poter misurare l'accuratezza su un insieme di dati, l'API di TensorFlow mette a disposizione un comando che valuta un modello di Machine Learning applicato ai dati passati come parametro:

```
model.evaluate(values, labels)
```

Questa chiamata ritorna il valore assunto dalla funzione di loss al termine della valutazione e le metriche che vengono specificate durante la compilazione del modello. Compilando il modello con il seguente comando

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

otterremo un modello la cui funzione di loss è la cross-entropia categorica, usata per modelli che devono classificare dati appartenenti ad una sola classe fra diverse disponibili, e come metrica aggiuntiva richiediamo l'accuratezza. Così facendo, con

```
model.evaluate(values, labels)[1]
```

otteniamo la misura dell'accuratezza che il modello `model` ottiene sul test set composto dai dati `values` e dalle etichette `labels`.

La misura di accuratezza indicata nei risultati è l'accuratezza media ottenuta dal modello su tutte le sessioni di addestramento.

3.1.2 ACC, FWT e BWT

Queste tre metriche sono state tratte e adattate dalla pubblicazione *Gradient Episodic Memory for Continual Learning*[20], e si basano sulla costruzione di una matrice $R \in \mathbb{R}^{E \times T}$. Con T si indica il numero di classi del problema in esame, e ogni volta che il modello finisce una sessione di addestramento tra le E che vengono eseguite si va a valutare la sua *test performance* su tutte le T classi. Così facendo, si calcolano le $R_{i,j} \in R$ valutazioni dell'accuratezza del test di classificazione del modello sulla classe t_j dopo aver osservato gli esempi della i-esima sessione di addestramento. Inoltre, una volta creato il modello con una inizializzazione dei pesi casuale si calcola $\overline{b}$, valutazione delle accuratezza nei test sui vari task.
Possiamo ora definire le tre metriche:

- **ACC**, *Accuracy* o accuratezza

$$ACC = \frac{1}{T} \cdot \sum_{i=1}^{T} R_{E,i} \tag{3.1}$$

  Va a misurare l'accuratezza del modello su tutti le T classi a fine addestramento

- **BWT**, *Backward Transfer* o trasferimento all'indietro

$$BWT = \frac{1}{T-1} \cdot \sum_{i=1}^{T} \left(\frac{1}{E} \sum_{j=1}^{E} R_{E,i} - R_{j,i} \right) \tag{3.2}$$

  Valuta l'influenza che ha una sessione di apprendimento sulle precedenti, valutando l'impatto che ha avuto sulle valutazioni dell'accuratezza in fase di test. Si ha un BWT positivo quando una sessione di addestramento migliora l'accuratezza delle precedenti già valutate, mentre un valore negativo indica che la sessione di addestramento ha deteriorato l'accuratezza che si otteneva grazie alla precedente conoscenza. Non è significativo parlare di BWT durante la prima sessione di addestramento.
  Un BWT fortemente negativo evidenzia il verificarsi del fenomeno del *catastrophic forgetting*.

- **FWT**, *Forward Transfer* o trasferimento in avanti

$$FWT = \frac{1}{T-1} \cdot \sum_{i=1}^{T} \left(\frac{1}{E} \sum_{j=1}^{E} R_{j,i} - \overline{b}_i \right) \tag{3.3}$$

  Serve per misurare l'influenza che la sessione di apprendimento ha sulla conoscenza ancora da apprendere. In particolare, un FWT positivo è possibile quando il modello è in grado di realizzare lo *zero-shot* learning, cioè di apprendere informazioni sui task futuri senza essere esposto ad esempi da apprendere, per esempio sfruttando la struttura interna della conoscenza in caso di task simili.
  Non è significativo parlare di FWT durante l'ultima sessione di addestramento.

Tra queste tre, quella senza dubbio più interessante ai fini dell'esperimento è il BWT. Diventa particolarmente importante evitare gli scenari che portano al *catastrophic forgetting*: nel nostro caso, uno scenario simile porterebbe a dimenticare esempi di HSM passati e a far sì che il modello si basi solamente sull'esperienza recente, rischiando di dimenticare importanti esempi passati di stati d'animo e di conseguenza a non saper più riconoscerli correttamente.

Più grandi sono queste metriche, migliore è il comportamento del modello in esame. A parità di ACC, si preferisce il modello con maggior BWT e FWT che denoterebbe un miglior trasferimento della conoscenza attraverso i task e le sessioni di training.

3.1.3 Altre metriche

Le altre metriche raccolte durante gli esperimenti sono le seguenti:

- **Numero di epoche** medio, e deviazione standard
  Questa misurazione fornisce una indicazione sul tempo di convergenza delle reti neurali selezionate sui dati, così da poter stimare le loro prestazioni su hardware diversi da quello di test.

- **Tempo di addestramento** medio, e deviazione standard
  Gli esperimenti sono eseguiti su un personal computer comune, sfruttando le ottimizzazioni messe a disposizione dall'utilizzo di una GPU per i calcoli. Nello specifico, le computazioni sono state eseguite su una macchina che montava una GPU nVidia GeForce GTX 1070 con 6 Gb di memoria dedicata, una CPU Intel i5 3570 e 12 Gb di RAM.
  Il tempo di addestramento è calcolato come la media dei tempi impiegati delle varie sessioni di addestramento.

- **Quantità di memoria** media
  Metrica utile per valutare l'impatto sulla memoria del sistema che andrà poi a utilizzare il modello addestrato. Un elevato consumo di memoria può portare alla esclusione di dispositivi portatili o con specifiche particolari.

3.2 Baseline: l'addestramento offline

L'approccio classico all'addestramento di un modello di Machine Learning avviene raccogliendo i dati di addestramento e sottoponendoli al modello in una singola sessione di addestramento. Questa metodologia è statica e produce un modello che sui futuri dati si comporterà tanto meglio quanto essi saranno simili ai dati di addestramento.
La scelta di usare questa metodologia classica come baseline è stata guidata dal principio che, avendo tutti i dati a disposizione in una sola volta, la rete neurale risultante dovrebbe ottenere le migliori prestazioni possibili sul test set una volta che questo viene sottoposto, e quindi fornisce un "limite superiore" all'accuratezza delle metodologie continue. Una strategia di Addestramento Continuo è tanto migliore quanto più si avvicina alle performance che la stessa rete neurale otterrebbe se venisse addestrata su tutti i dati in maniera offline o, se possibile, quando la supera.

L'addestramento offline è quindi realizzato seguendo il seguente pseudo-algoritmo, applicabile sia a WESAD che ad ASCERTAIN:

1. **Creare e compilare la rete neurale** che verrà poi addestrata.
   La creazione della rete neurale, attraverso la API Sequential di TensorFlow,

è analoga per entrambi i dataset a meno della struttura interna. L'esempio seguente è tratto dall'addestramento offline per WESAD, e costruisce una rete neurale con due layer da 18 unità GRU e un layer output da 4 unità, corrispondenti alle 4 classi del dataset.

```
model = tf.keras.models.Sequential()
model.add(tf.keras.layers.GRU(18, return_sequences=True,
    input_shape=(100, 14)))
model.add(tf.keras.layers.GRU(18))
model.add(tf.keras.layers.Dense(4, activation = 'softmax',
                  kernel_regularizer = tf.keras.
    regularizers.l1_l2(l1 = 1e-04, l2 = 1e-04),
                  bias_regularizer = tf.keras.regularizers
    .l1_l2(l1 = 1e-04, l2 = 1e-04),
                  activity_regularizer = tf.keras.
    regularizers.l1_l2(l1 = 1e-04, l2 = 1e-04)))
opt = tf.keras.optimizers.Adam(learning_rate = 0.005,
    beta_1 = 0.99, beta_2 = 0.99)
model.compile(loss = 'categorical_crossentropy', optimizer
     = opt, metrics = ['accuracy'])
```

2. **Caricare il dataset**, composto da training set e test set.
   Come precedentemente spiegato, i dataset sono divisi in training set e test set, e il training set è ulteriormente diviso fra i soggetti dei due studi. In entrambi i casi quindi, per quanto riguarda la metodologia offline, viene caricato il test set così come viene fornito, mentre il training set è realizzato concantenando fra loro i dati di tutti i soggetti. Esempio, sempre tratto da WESAD:

```
Xtr, ytr = None, None
for S in ["S2", "S3", "S4", "S5", "S6", "S7", "S8", "S9",
    "S10", "S11", "S13", "S14", "S15", "S16", "S17"]:
     Xs = pickle.load(open("datasets/WESAD/splitted/X" + S
    + ".pkl", 'rb'), encoding='latin1')
     ys = pickle.load(open("datasets/WESAD/splitted/y" + S
    + ".pkl", 'rb'), encoding='latin1')

     if (Xtr is None):
         Xtr = copy.deepcopy(Xs)
         ytr = copy.deepcopy(ys)
     else:
         Xtr = np.concatenate([Xtr, Xs], axis = 0)
         ytr = np.concatenate([ytr, ys], axis = 0)
     del Xs
     del ys

Xts = pickle.load(open("datasets/WESAD/splitted/Xts.pkl",
    'rb'), encoding='latin1')
```

```
52 yts = pickle.load(open("datasets/WESAD/splitted/yts.pkl",
       'rb'), encoding='latin1')
```

Il training set è ulteriormente diviso in training set e validation set, utilizzato per monitorare l'andamento dell'addestramento e implementare la fermata anticipata dell'addestramento:

```
56 Xtr, Xvl, ytr, yvl = train_test_split(Xtr, ytr, test_size
       = 0.25, train_size = 0.75, random_state=42)
```

3. **Addestramento** e risultati.
   Una volta preparato il dataset, si può dare il via all'addestramento. Prima di esso si preparano, sempre grazie all'API di TensorFlow, le due callback usate durante gli esperimenti: la creazione di grafici attraverso TensorBoard e la fermata anticipata in base all'andamento del valore della funzione di loss sul validation set:

```
59 logdir = get_run_logdir("all")
60 es = tf.keras.callbacks.EarlyStopping(monitor = 'val_loss'
       , mode = 'min', patience = 10, verbose = 1,
       restore_best_weights = True)
61 tb = tf.keras.callbacks.TensorBoard(log_dir=logdir,
       write_graph=True)
62 start = time.time()
63 graph = model.fit(Xtr, ytr, epochs = 100, validation_data
       = (Xvl, yvl), callbacks = [es, tb], batch_size=256,
       use_multiprocessing=True, workers=8)
64 end = time.time()
65 modeltime += (end - start)
66 history = model.history.history['val_loss']
67 epochs.append(np.argmin(history) + 1)
68 scores.append(model.evaluate(Xts, yts)[1])
```

I risultati sono poi proiettati in output sul terminale:

```
70 print('Media numero epoche:', np.around(np.mean(epochs),
       2))
71 print('Media tempo di addestramento:', np.around(modeltime
       , 2), 's')
72 print('Media accuracy:', np.around(np.mean(scores), 4)
       *100, '%')
73 print('Deviazione standard accuracy:', np.around(np.std(
       scores), 4)*100, '%')
```

La metodologia offline, come visto, è immediata e grazie alle API di TensorFlow anche semplice da implementare. Essa è stata utilizzata anche per poter selezionare la rete neurale usata per il dataset relativo, una per WESAD e una per ASCERTAIN

e che è poi stata usata su ciascun esperimento, in base all'accuratezza finale raggiunta.

3.3 Metodologie di Addestramento Continuo

Al centro di questo progetto ci sono le metodologie di Addestramento Continuo. Ne sono state selezionate alcune: 3 tecniche di replay e 2 tecniche che non sfruttano esempi già visti.

Per realizzare il flusso di dati necessario a implementare una strategia di Addestramento Continuo, le quali richiedono che i dati di addestramento non siano subito tutti disponibili ma che lo diventino nel tempo, i dati di entrambi i dataset sono stati mantenuti suddivisi per soggetto e raggruppati a coppie, per poi essere sottoposti all'addestramento della rete neurale una coppia per volta. Le tabelle 3.1 e 3.2 mostrano la suddivisione a coppie rispettivamente per il dataset WESAD e il dataset ASCERTAIN.

Sessione	Soggetti
1	S2, S3
2	S4, S5
3	S6, S7
4	S8, S9
5	S10, S11
6	S13, S14
7	S15, S16

Tabella 3.1: *Suddivisione a coppie del dataset WESAD*

Sessione	Soggetti
1	S0, S1
2	S2, S3
3	S4, S5
4	S6, S7
5	S8, S9
6	S10, S11
7	S12, S13
8	S14, S15

Tabella 3.2: *Suddivisione a coppie del dataset ASCERTAIN*

Per poter misurare le metriche ACC, BWT e FWT delle metodologie di continual learning, in ogni esperimento viene prima preparato il test set suddividendolo nelle varie classi:

```
21 Xts = pickle.load(open("datasets/WESAD/splitted/Xts.pkl", 'rb')
       , encoding='latin1')
22 yts = pickle.load(open("datasets/WESAD/splitted/yts.pkl", 'rb')
       , encoding='latin1')

24 Xts_1, Xts_2, Xts_3, Xts_4, yts_1, yts_2, yts_3, yts_4 = [],
       [], [], [], [], [], [], []
25 idx_1 = [i for i, x in enumerate(yts) if x[0] == 1]
26 for i in idx_1:
27     Xts_1.append(Xts[i])
28     yts_1.append([1., 0., 0., 0.])
```

```
Xts_1, yts_1 = np.array(Xts_1), np.array(yts_1)
idx_2 = [i for i, x in enumerate(yts) if x[1] == 1]
for i in idx_2:
    Xts_2.append(Xts[i])
    yts_2.append([0., 1., 0., 0.])
Xts_2, yts_2 = np.array(Xts_2), np.array(yts_2)
idx_3 = [i for i, x in enumerate(yts) if x[2] == 1]
for i in idx_3:
    Xts_3.append(Xts[i])
    yts_3.append([0., 0., 1., 0.])
Xts_3, yts_3 = np.array(Xts_3), np.array(yts_3)
idx_4 = [i for i, x in enumerate(yts) if x[3] == 1]
for i in idx_4:
    Xts_4.append(Xts[i])
    yts_4.append([0., 0., 0., 1.])
Xts_4, yts_4 = np.array(Xts_4), np.array(yts_4)
```

Subito dopo, appena viene creato il modello, vengono inizializzate le variabili necessarie e il vettore $\overline{b}$:

```
R = []
T = 4
E = 7
b = [model.evaluate(Xts_1, yts_1, verbose = 0)[1], model.
    evaluate(Xts_2, yts_2, verbose = 0)[1], model.evaluate(Xts_3
    , yts_3, verbose = 0)[1], model.evaluate(Xts_4, yts_4,
    verbose = 0)[1]]
ACC, BWT, FWT = 0, 0, 0
modeltime = 0
epochs = []
scores = []
```

Al termine di ogni sessione di addestramento, cioè appena concluso l'addestramento su una coppia, vengono calcolati gli $R_{i,j}$:

```
    R.append([model.evaluate(Xts_1, yts_1, verbose = 0)[1],
    model.evaluate(Xts_2, yts_2, verbose = 0)[1], model.evaluate
    (Xts_3, yts_3, verbose = 0)[1], model.evaluate(Xts_4, yts_4,
     verbose = 0)[1]])
```

Infine, al termine dell'ultima sessione di addestramento, vengono calcolate le varie metriche:

```
t = 0
for i in range(T):  # ACC
        t += R[E-1][i]
ACC = t/T

t = 0
```

```
105 for i in range(T):  # BWT
106         m = 0
107         for j in range(E):
108             m += (R[E-1][i] - R[j][i])
109         t += m/E
110 BWT = t/(T)
111
112 t = 0
113 for i in range(T):  # FWT
114         m = 0
115         for j in range(E):
116             m += (R[j][i] - b[i])
117         t += m/E
118 FWT = t/(T)
```

3.3.1 Naive

La prima metodologia continua sperimentata è chiamata *naive*. Questa metodologia consiste nel fornire al modello una coppia alla volta come training set, senza ulteriori elaborazioni: la semplicità di questa metodologia la rende molto facile da implementare, ma anche esposta alle problematiche intrinseche dell'Apprendimento Continuo esposte precedentemente. Questa metodologia è stata descritta in alcuni studi pionieristici nel campo dell'Apprendimento Continuo, come [25].

Nel dettaglio, inizialmente vengono stabilite le coppie:

```
66 for S in [("S2", "S3"), ("S4", "S5"), ("S6", "S7"), ("S8", "S9"
    ), ("S10", "S11"), ("S13", "S14"), ("S15", "S16")]:
```

Ogni coppia è caricata e concatenata, per formare il training set della sessione che viene poi diviso in training set e validation set:

```
68     Xa = pickle.load(open("datasets/WESAD/splitted/X" + S[0] +
    ".pkl", 'rb'), encoding='latin1')
69     ya = pickle.load(open("datasets/WESAD/splitted/y" + S[0] +
    ".pkl", 'rb'), encoding='latin1')
70     Xb = pickle.load(open("datasets/WESAD/splitted/X" + S[1] +
    ".pkl", 'rb'), encoding='latin1')
71     yb = pickle.load(open("datasets/WESAD/splitted/y" + S[1] +
    ".pkl", 'rb'), encoding='latin1')

73     X = np.concatenate([Xa, Xb], axis = 0)
74     y = np.concatenate([ya, yb], axis = 0)

83     Xtr, Xvl, ytr, yvl = train_test_split(X, y, test_size =
    0.1, train_size = 0.9, random_state=42)
```

Infine il modello è addestrato sul training set e validato sul validation set appena prodotti, sono calcolate le metriche ed esposte in output.

3.3.2 Replay

Questa è la prima tecnica di replay provata, ed anche la più semplice. Questa metodologia consiste nel mantenere una percentuale del training set utilizzato per addestrare la rete neurale durante l'attuale sessione di addestramento e usarla insieme al training set, cioè alla coppia di soggetti, della sessione di addestramento successiva. Anche questa metodologia è di facile implementazione e la sua filosofia è trattata in studi come [26].

Subito prima di avviare l'addestramento, vengono inizializzate le due variabili che conterranno il training set:

```
65 X, y = None, None
```

Appena caricati i soggetti, viene costruito il training set mantenendo ciò che è stato trattenuto dalla sessione precedente e memorizzato in `X`, se presente:

```
74     if (X is None):
75         X = np.concatenate([Xa, Xb], axis = 0)
76         y = np.concatenate([ya, yb], axis = 0)
77     else:
78         X = np.concatenate([X, Xa, Xb], axis = 0)
79         y = np.concatenate([y, ya, yb], axis = 0)
```

A fine sessione di addestramento, si trattiene la percentuale di replay (nell'esempio, il 25%):

```
100     X, _, y, _ = train_test_split(Xtr, ytr, test_size = 0.75,
    train_size = 0.25, random_state=42)  # replay random di
    elementi precedenti
```

3.3.3 Cumulative

Con questa metodologia, si intende mantenere di volta in volta l'intero training set accumulando le coppie di soggetti che diventano disponibili. Così facendo, all'arrivo dell'ultima coppia di soggetti si avrà un training set pari al training set usato nella metodologia offline, contenendo tutti i dati di tutti i precedenti soggetti, con la differenza che si addestra una rete neurale che è stata già addestrata su tutte le coppie presenti nel training set ad eccezione dell'ultima arrivata. Questa metodologia può essere considerata una versione particolare della metodologia di replay precedente: abbiamo un replay del 100% dei dati, così che tra una sessione di addestramento e l'altra si mantiene l'intero training set utilizzato.

Viene realizzato nel seguente modo: la prima sessione di addestramento è realizzata sulla prima coppia di soggetti partendo dalla rete neurale inizializzata con i pesi casuali, mentre mano a mano che arrivano le coppie successive esse vengono concatenate al training set della sessione precedente e la rete è addestrata senza reinizializzare i pesi, cioè mantenendo l'addestramento precedente. Una problematica da tenere in considerazione è la memoria necessaria richiesta per mantenere i training set nella loro totalità: durante l'ultima sessione di addestramento avremo un training set corrispondente al training set usato nella metodologia offline, di conseguenza mantenere in memoria tutti i dati necessari potrebbe non essere praticabile in contesti reali.

3.3.4 Episodic

Nelle precedenti tecniche di replay si mantenevano tutti i dati o un sottoinsieme scelto casualmente del training set. Con questa tecnica, invece, viene mantenuto di sessione in sessione un numero fisso m di esempi per ogni etichetta, così da avere un replay bilanciato rispetto alle classi. L'idea dietro questa tecnica di replay è descritta nello studio "Gradient Episodic Memory for Continuum Learning"[20].

Subito prima di avviare l'addestramento, vengono inizializzate le "memorie episodiche" e il parametro m, che indica quanti esempi per classe mantenere (nell'esempio, $m = 70$):

```
68 X_1, X_2, X_3, X_4, y_1, y_2, y_3, y_4 = [], [], [], [], [],
    [], [], []  # memorie task
69 m = 70
```

Al termine di una sessione di addestramento, il training set appena usato è suddiviso per classi:

```
105     idx_1 = [i for i, x in enumerate(ytr) if x[0] == 1]
106     for i in idx_1:
107         X_1.append(Xtr[i])
108         y_1.append([1., 0., 0., 0.])

112     idx_2 = [i for i, x in enumerate(ytr) if x[1] == 1]
113     for i in idx_2:
114         X_2.append(Xtr[i])
115         y_2.append([0., 1., 0., 0.])

119     idx_3 = [i for i, x in enumerate(ytr) if x[2] == 1]
120     for i in idx_3:
121         X_3.append(Xtr[i])
122         y_3.append([0., 0., 1., 0.])

126     idx_4 = [i for i, x in enumerate(ytr) if x[3] == 1]
```

```
127     for i in idx_4:
128         X_4.append(Xtr[i])
129         y_4.append([0., 0., 0., 1.])
```

Ogni sottoinsieme è mescolato, per evitare di mantenere sempre i soliti esempi, e da ognuno ne vengono presi m:

```
109     random.shuffle(X_1)
110     X_1 = X_1[0:m]
111     y_1 = y_1[0:m]

116     random.shuffle(X_2)
117     X_2 = X_2[0:m]
118     y_2 = y_2[0:m]

123     random.shuffle(X_3)
124     X_3 = X_3[0:m]
125     y_3 = y_3[0:m]

130     random.shuffle(X_4)
131     X_4 = X_4[0:m]
132     y_4 = y_4[0:m]
```

Infine, viene formato il training set che andrà a concatenarsi alla coppia di soggetti successiva:

```
134     X = np.concatenate([X_1, X_2, X_3, X_4], axis = 0)
135     y = np.concatenate([y_1, y_2, y_3, y_4], axis = 0)
```

3.3.5 Elastic Weight Consolidation

Questa metodologia di continual learning, introdotta nella pubblicazione *Overcoming catastrophic forgetting in neural networks*[17], si basa sul principio che non tutti i parametri di una rete neurale sono importanti ai fini dell'apprendimento di un determinato task. Dato un modello con parametri Θ e un dataset D, si può formalizzare il problema dell'apprendimento come la ricerca della parametrizzazione Θ dato il dataset D che massimizza $p(\Theta \mid D)$. Seguendo il Teorema di Bayes

$$p(\Theta \mid D) = \frac{p(D \mid \Theta) \cdot p(\Theta)}{p(D)} \tag{3.4}$$

che può essere trasformata in

$$\log p(\Theta \mid D) = \log p(D \mid \Theta) + \log p(\Theta) - \log p(D) \tag{3.5}$$

Assumendo che il dataset sia diviso in due parti, D_A e D_B, poiché l'obiettivo è apprendere $p(\Theta \mid D)$ con la suddivisione diventa apprendere prima $p(\Theta \mid D_A)$ e

successivamente $p(\Theta \mid D_B)$. Algebricamente, l'obiettivo è

$$p(\Theta \mid D) = p(p(\Theta \mid D_A) \mid D_b) = \frac{p(D_B \mid p(\Theta \mid D_A)) \cdot p(\Theta \mid D_A)}{p(D_B)} \tag{3.6}$$

che, analogamente a 3.4, può essere trasformata in

$$\log p(\Theta \mid D) = \log p(D_B \mid \Theta) + \log p(\Theta \mid D_A) - \log p(D_B) \tag{3.7}$$

Si può dedurre come tutta la conoscenza relativa al task A viene assorbita dal termine $p(\Theta \mid D_A)$, cioè esso indica quali parametri sono importanti per il task A. Con $p(\Theta \mid D_A)$ si può quindi calcolare una matrice F_i, chiamata Matrice dell'Informazione di Fisher, per ogni parametro $\Theta_i \in \Theta$ rispetto a D_A: questa matrice stima l'importanza del parametro Θ_i per quanto riguarda il task descritto dai dati D_A. Questa informazione può essere usata come "penalità elastica", e nella pubblicazione viene fatta l'analogia ad un elastico che trattiene il valore di Θ_i dal distanziarsi troppo dalla precedente soluzione, in modo proporzionale a F_i. La penalità elastica, cioè il termine di regolarizzazione, è

$$\Omega(\Theta) = \sum_i \frac{1}{2} F_i \left(\Theta_i - \Theta^*_{A,i}\right)^2 \tag{3.8}$$

che rappresenta l'elasticità con cui ogni parametro $\Theta_i \in \Theta$ viene mantenuto vicino a $\Theta^*_{A,i} \in \Theta^*_A$. Si aggiunge anche un ulteriore parametro, λ che descrive l'importanza del termine di regolarizzazione.

Con queste informazioni, la loss che viene minimizzata nella metodologia EWC è

$$loss(\Theta) = loss_B(\Theta) + \sum_i \frac{\lambda}{2} F_i \left(\Theta_i - \Theta^*_{A,i}\right)^2 \tag{3.9}$$

dove $loss_B(\Theta)$ è la loss calcolata solamente sul task B, λ è un parametro che indica l'importanza dei vecchi task rispetto ai nuovi e i etichetta ogni parametro. La matrice F è approssimata come la media dei gradienti delle log-verosomiglianze di N esempi presi da D_A al quadrato:

$$F = \frac{1}{N} \sum_{i=1}^{N} \nabla_\Theta \log p(x_{A,i} \mid \Theta^*_A) \, \nabla_\Theta \log p(x_{A_i} \mid \Theta^*_A)^T \tag{3.10}$$

3.3.6 Learning Without Forgetting

La metodologia Learning Without Forgetting, delineata nell'omonima pubblicazione[18], sfrutta esclusivamente i nuovi dati per addestrare il modello mantenendo la cono-

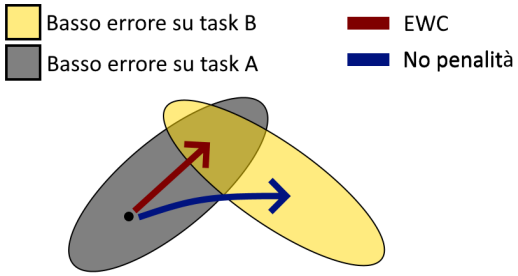

Figura 3.1: *Schematizzazione del comportamento di EWC*

scenza già appresa.
Questo metodo parte da una rete che possiede parametri Θ, di cui i parametri Θ_s sono condivisi fra i task e i parametri Θ_A sono specifici al task A. L'obiettivo è aggiungere parametri Θ_B per il nuovo task usando solo i nuovi dati senza deteriorare la conoscenza precedentemente appresa.

LWF inizia applicando il modello addestrato ai dati nuovi appena arrivati, registrando i risultati che esso produce. Dopodiché si addestra il modello, usando come loss la somma di due loss così definite:

$$loss_{new}(y_n, \overline{y}_n) = -y_n \cdot \log \overline{y}_n \tag{3.11}$$

$$loss_{old}(y_o, \overline{y}_o) = -\sum_{i=1}^{l} y_o^{'(i)} \log y_o^{'(i)} \tag{3.12}$$

dove

$$y_o^{'(i)} = \frac{\left(y_o^{(i)}\right)^{1/T}}{\sum_j \left(y_o^{(j)}\right)^{1/T}}, \quad \overline{y}_o^{'(i)} = \frac{\left(\overline{y}_o^{(i)}\right)^{1/T}}{\sum_j \left(\overline{y}_o^{(j)}\right)^{1/T}} \tag{3.13}$$

T è un parametro della *Knowledge Distillation loss* settato a $T = 2$, l è il numero di etichette, y_n le etichette dei nuovi dati, $\overline{y}_n$ le predizioni dei nuovi dati, y_o le precedenti previsioni del precedente modello e $\overline{y}_o$ le nuove previsioni del precedente modello.
La nuova loss è quindi

$$\lambda_o \cdot loss_{old}(Y_o, \overline{Y}_o) + loss_{new}(Y_n, \overline{Y}_n) + R(\Theta_s, \Theta_o, \Theta_n) \tag{3.14}$$

Dove λ_o è un peso che è tanto più grande quanto più sono importanti i vecchi dati rispetto ai nuovi, e R è una funzione di regolarizzazione dei parametri.

Questa metodologia, che tiene di conto della performance che il modello aveva fino all'arrivo dei nuovi dati, tende quindi a mantenere la rete neurale simile a quella già addestrata, evitando così di variare considerevolmente i pesi distanziandoli da

quelli appresi sui dati precedenti.

3.4 Conclusione

La scelta delle metodologie adottate, elemento centrale di questa tesi, si è rivelata particolarmente laboriosa. Anzitutto si è reso necessario inquadrare precisamente lo scopo del lavoro, cioè la comparazione fra diverse tecniche di Addestramento Continuo applicate a dati relativi allo human state monitoring. Questo ha richiesto la selezione di dataset appropriati e in seguito la scelta di semplici metodologie di Addestramento Continuo che potessero essere comparate. Il confronto ha quindi richiesto la ricerca e la scelta di metriche che ne misurassero i diversi aspetti di trasferimento della conoscenza e dell'accuratezza media e finale, oltre che i tempi di addestramento e la memoria utilizzata.
Il tutto è stato preceduto dalla ricerca e dallo studio personale dell'argomento, oltre che dalla scelta del linguaggio, delle librerie e dell'ambiente di sviluppo. Questo ha portato ad una lunga serie di esperimenti esplorativi, per poter studiare e testare i vari aspetti del Machine Learning e delle reti neurali prima, e dell'Addestramento Continuo successivamente. Questi esperimenti hanno permesso di affrontare le diverse problematiche comuni ad un problema di Machine Learning: il preprocessing dei dati, l'individuazione degli iperparametri adatti ad un modello, la corretta progettazione di una rete neurale, i tempi di addestramento e altri ancora.

Tutto questo lavoro è confluito in una serie di esperimenti finali che hanno prodotto i risultati presentati nel prossimo capitolo.

CAPITOLO 4

Valutazione Empirica

In questo capitolo verranno discussi nel dettaglio gli esperimenti realizzati nell'ambito di questa tesi. Verranno esposte le scelte adoperate nella selezione e realizzazione degli esperimenti e nella raccolta dei risultati.
Infine, verranno presentati e discussi i risultati raccolti.

Nella sezione 4.1 sono descritti gli esperimenti realizzati e vengono giustificate le scelte riguardanti gli iperparametri selezionati. Nella sezione 4.2 sono infine presentati in dettaglio i risultati finali ottenuti nell'ambito di questa tesi.

4.1 Esperimenti

4.1.1 Panoramica

Gli esperimenti sono stati eseguiti su un personal computer con sistema operativo Windows 10 in versione developer build 21390.2025 e all'interno della Windows Subsystem for Linux, con supporto alla GPU. Il computer monta una GPU nVidia GeForce GTX 1070 con 6 Gb di memoria dedicata, una CPU Intel i5 3570 e 12 Gb di RAM.

Le *callback* utilizzate durante le sessioni di addestramento sono state le seguenti:

- *TensorBoard*, per poter visualizzare in tempo reale l'andamento dell'addestramento delle varie reti.

```
61 tb = tf.keras.callbacks.TensorBoard(log_dir=logdir,
       write_graph=True)
```

- *EarlyStopping*, per poter fermare anticipatamente l'addestramento in caso di diverse epoche svolte senza ottenere risultati migliori. La fermata anticipata è stata impostata sulla loss calcolata sul validation set, con pazienza di 10 epoche e ripristino dei migliori pesi trovati della rete neurale.

```
60 es = tf.keras.callbacks.EarlyStopping(monitor = 'val_loss'
    , mode = 'min', patience = 10, verbose = 1,
    restore_best_weights = True)
```

Tutti gli esperimenti sono stati eseguiti con batch di 256 elementi e per una durata massima 100 epoche, a meno di fermata anticipata. Inoltre il training set è sempre stato diviso in 75% training set e 25% validation set.

```
63 graph = model.fit(Xtr, ytr, epochs = 100, validation_data = (
    Xvl, yvl), callbacks = [es, tb], batch_size=256,
    use_multiprocessing=True, workers=8)
```

4.1.2 Model Selection

Il modello adoperato per ogni dataset è stato selezionato in base all'accuratezza che esso ha raggiunto durante l'addestramento offline. L'approccio alla model selection è stato il medesimo per entrambi i dataset, delineato di seguito:

1. Selezione dei parametri dell'ottimizzatore
   L'algoritmo di ottimizzazione selezionato è Adam, poiché consigliato per problemi con molti dati rumorosi, efficiente in termini di tempo e spazio in memoria e facilità di fine-tuning degli iperparametri[16].
   Lo spazio dei parametri preso in considerazione è stato:
   - *Learning rate* $\in \{0.001, 0.005, 0.008, 0.01\}$
   - $\beta_1 \in \{0.8, 0.85, 0.9, 0.95, 0.99\}$
   - $\beta_2 \in \{0.98, 0.985, 0.99, 0.995, 0.999\}$
2. Selezione dei parametri di regolarizzazione
   Il regolarizzatore L1L2 è stato applicato sul kernel, sui bias e sui valori d'uscita del layer di output.
   I parametri presi in considerazione sono:
   - Kernel $\in \{0.0001, 0.00001, 0.000001\}$
   - Bias $\in \{0.0001, 0.00001, 0.000001\}$
   - Output $\in \{0.0001, 0.00001, 0.000001\}$
3. Selezione della struttura della rete neurale
   La rete neurale è stata scelta in base al miglior risultato ottenuto dopo 5 epoche di addestranmento.
   Le configurazioni prese in considerazione sono:

- Layer $\in \{1, 2, 3, 4\}$
- Unità $\in [10, 40]$

Questo procedimento ha prodotto i seguenti modelli, uno per dataset:

- Dataset **WESAD**
  Due layer nascosti composti da 18 unità GRU con i seguenti parametri:
  - *Learning rate* $= 0.005$
  - $\beta_1 = 0.99$
  - $\beta_2 = 0.99$
  - Kernel, bias, output $= 0.0001$

  Il modello è capace di raggiungere in modo consistente il 99% di accuratezza sul dataset durante l'addestramento offline. Questo lo rende un ottimo candidato per giudicare l'applicabilità dello stesso modello su diverse metodologie di continual learning.

- Dataset **ASCERTAIN**
  Due layer nascosti composti da 24 unità GRU con i seguenti parametri:
  - *Learning rate* $= 0.01$
  - $\beta_1 = 0.9$
  - $\beta_2 = 0.99$
  - Kernel, bias, output $= 0.00001$

  Anche questo modello è stato selezionato in base all'accuratezza sul dataset, che si attesta intorno al 45%.

La scelta dei modelli si è basata sui risultati ottenuti nell'addestramento offline poiché è interessante comparare la differenza di risultati ottenuti negli scenari dove i dati di addestramento diventano disponibili nel tempo, cioè le metodologie di Addestramento Continuo, allo scenario in cui i dati sono disponibili sin da subito. Questo consente di verificare il cambiamento di prestazioni del solito modello nei diversi casi.

4.1.3 Percentuale di replay

La percentuale di dati da ripetere nella sessione di addestramento successiva, secondo lo scenario di replay, è stata scelta provando diversi valori percentuali in base all'accuratezza e alle altre metriche misurate nell'Addestramento Continuo naive sul dataset WESAD, con i risultati presentati nella tabella 4.1.
La scelta è ricaduta sul 25% di replay poiché le metriche misurate sono mediamente più alte delle altre percentuali, in particolare nel trasferimento di conoscenza.

Percentuale	Accuratezza	ACC	BWT	FWT
10%	74,55%	0,7589	0,007	0,5539
15%	75,94%	0,7593	-0,005	0,4693
20%	74,50%	0,7568	0,0054	0,3342
25%	**74,83%**	**0,7707**	**0,0158**	**0,5275**
33%	74,71%	0,7823	0,0105	0,5948

Tabella 4.1: *Comparazione fra diverse percentuali di replay*

Inoltre, dalle prove effettuate, l'incremento dell'accuratezza derivato dall'aumentare la percentuale di replay è risultato irrisorio. La scelta è stata ulteriormente influenzata dall'utilizzo di memoria, e un valore maggiore porta a modelli migliori pur usando più memoria durante l'addestramento.

4.1.4 Numero di esempi nelle memorie episodiche

Per quanto riguarda lo scenario episodico, l'iperparametro m indica il numero di esempi per ogni classe che vengono mantenuti fra una sessione di addestramento e l'altra. Tale iperparametro è stato scelto provando diversi valori sulla base dell'accuratezza e delle altre metriche misurate nell'addestranento episodico su dataset WESAD, ottenendo i risultati elencati nella tabella 4.2

$m =$	Accuratezza	ACC	BWT	FWT
40	79,07%	0,8450	0,0516	0,3752
50	78,08%	0,8082	0,0253	0,5651
60	79,50%	0,8979	0,0998	0,6109
70	**81,11%**	**0,8589**	**0,0447**	**0,6351**
80	80,83%	0,9064	0,0957	0,4223
90	80,92%	0,9031	0,0892	0,4793

Tabella 4.2: *Comparazione fra vari valori di m nell'addestramento episodico*

Viene scelto $m = 70$ poiché raggiunge un'accuratezza media superiore e un'accettabile trasferimento di conoscenza. Anche qua la scelta è dettata dall'utilizzo di memoria, e come dimostrato dai dati un valore maggiore di m può risultare in generale preferibile.

4.2 Risultati finali

WESAD Usando come modello una rete neurale con due layer nascosti composti da 18 unità GRU e un layer di output con 4 unità aventi funzione di attivazione *softmax*, su WESAD sono stati ottenuti i seguenti risultati:

Scenario	Epoche	Tempo	Acc.	ACC	BWT	FWT	Memoria
Offline	28	94,69s	99,07	-	-	-	2061,40 Mb
Continual	49,14±33,67	947,24s	73,13±4,02	0,7721	0,0343	0,5397	2173 Mb
Cumulative	39,71±19,91	2786s	81,97±8,67	0,961	0,1383	0,4674	2291,45 Mb
Replay	41,14±21,19	1063,51s	78,29±3,32	0,7849	-0,002	0,4582	2184,77 Mb
Episodic	35±26,26	1088,29s	82,13±6,60	0,9095	0,0841	0,4226	2097,47 Mb
EWC	29,71±17,38	1342,81s	70,74±4,74	0,7251	0,0113	0,4698	2187,40 Mb
LWF	44,29±18,30	3282,51s	69,09±5,82	0,7419	0,0451	0,3248	2121,31 Mb

Tabella 4.3: *Risultati WESAD*

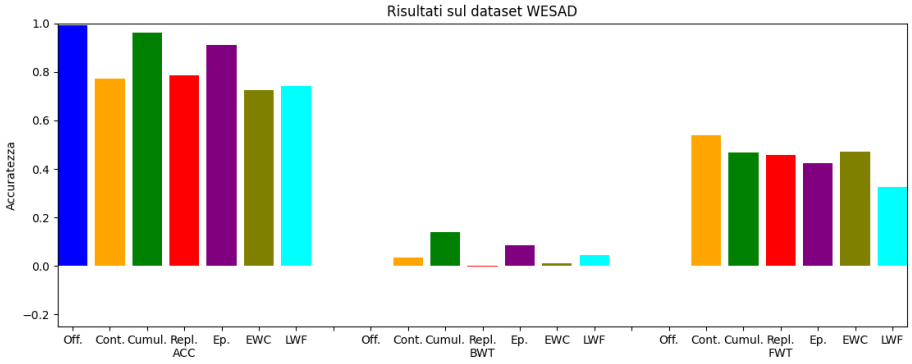

Figura 4.1: *Risultati su WESAD*

L'accuratezza del 99% raggiunta nel training offline dimostra come il dataset WESAD sia particolarmente adatto all'inferenza da parte di reti neurali ricorrenti dalla bassa complessità. Con poco più di 25 epoche di addestramento, abbiamo ottenuto un modello capace di classificare con un alto grado di accuratezza lo stato psico-fisico di un utente.
Ciò che è interessante notare è che l'accuratezza è mantenuta sopra il 70% anche nella maggior parte delle metodologie continue. La performance peggiore media la si ha nel *Learning Without Forgetting*, che però ottiene comunque un'accuratezza finale (ACC) del 74% e un FWT comunque ottimo. La miglior performance tra le metodologie continue è ottenuta dall'apprendimento cumulativo, con un'accuratezza finale del 96% perfettamente paragonabile alla metodologia offline, e in linea con la teoria. In tutti gli approcci, si ha un FWT elevato e molto superiore al BWT, segnale che ogni soggetto migliora molto l'apprendimento sui soggetti successivi, mentre non vi è particolare influenza sui precedenti soggetti o, se vi è, è positiva.

L'esperimento quindi dimostra come l'Apprendimento Continuo su dati sensoriali come respirazione, temperatura, elettrocardiogramma e simili possa portare a modelli con performance paragonabili all'apprendimento offline. Con un po' di *fine-tuning* e *model selection* con le metodologie continue si può probabilmente trovare un modello con performance ancora superiori a quelle sperimentate.

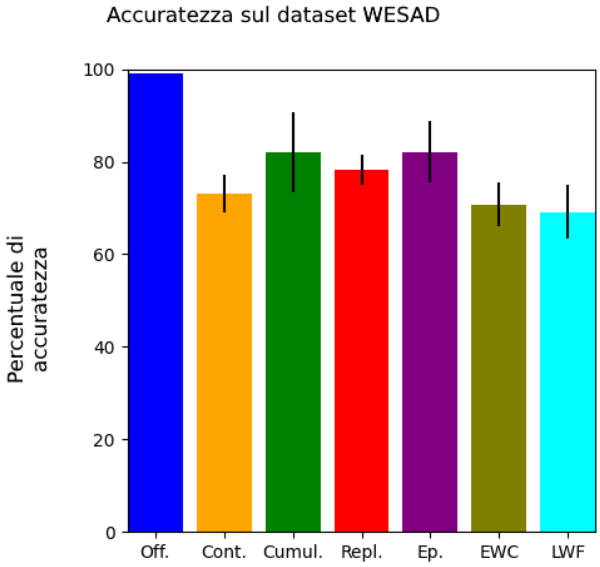

Figura 4.2: *Accuratezza su WESAD*

ASCERTAIN Usando come modello una rete neurale con due layer nascosti composti da 24 unità GRU e un layer di output con 4 unità aventi funzione di attivazione *softmax*, su ASCERTAIN sono stati raggiunti i seguenti risultati:

Scenario	Epoche	Tempo	Acc.	ACC	BWT	FWT	Memoria
Offline	3	29,14s	42,78	-	-	-	1817,28 Mb
Continual	10,88±5,01	249,28s	37,45±5,01	0,25	-0,0168	0,0213	2154,36 Mb
Cumulative	9,25±6,81	932,56s	39,06±4,26	0,2697	0,0064	0,0278	2297,17 Mb
Replay	13,25±10,03	402,96s	39,48±4,37	0,2603	-0,014	0,0485	2173,95 Mb
Episodic	12,62±7,94	498,24s	38,80±4,04	0,2742	0,0048	0,0156	2231,42 Mb
EWC	24,14±16,94	810,96s	36,08±5,66	0,2497	-0,0183	-0,0003	2171,62 Mb
LWF	21,62±10,20	879,68s	35,69±5,43	0,2448	0.0327	0,0156	2103 Mb

Tabella 4.4: *Risultati ASCERTAIN*

Dalla metodologia offline è subito chiaro come il dataset ASCERTAIN presenti una difficoltà maggiore rispetto a WESAD. In particolare, il dataset ASCERTAIN presenta dati particolarmente sbilanciati sulla classe 0 e informazioni riguardo le espressioni facciali dei soggetti durante l'esperimento, dati di difficile classificazione.
Le metodologie continue sono comunque in linea con i risultati sul precedente

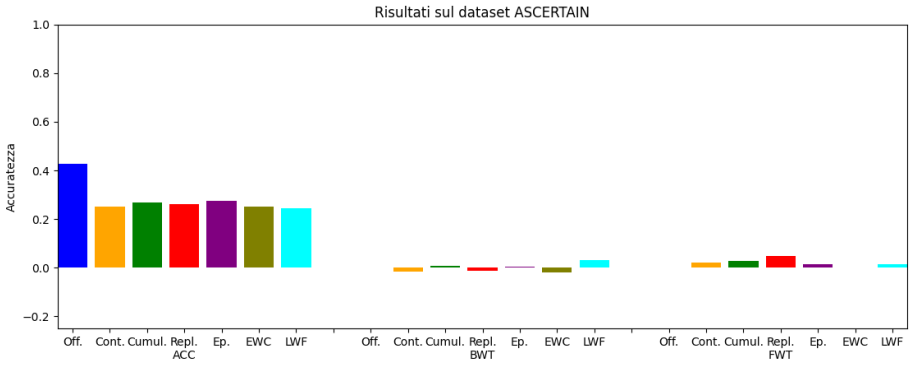

Figura 4.3: *Risultati sul dataset ASCERTAIN*

dataset, con la performance peggiore ottenuta dal *Learning Without Forgetting* mentre la migliore è ottenuta dalla metodologia cumulativa anche grazie ad un BWT superiore alla metodologia replay. In molti casi si nota un valore negativo di trasferimento di conoscenza all'indietro, sintomo di un lieve caso di *catastrophic forgetting*.

Per poter dimostrare l'impatto dovuto allo sbilanciamento delle classi, il dataset ASCERTAIN è stato riorganizzato in soggetti "fittizi" come esposto di seguito.

Custom ASCERTAIN Dopo aver eseguito il preprocessing specificato precedentemente, tutti i dati di addestramento sono stati uniti in un unico insieme. Da questo insieme è stato estratto il test set, e il rimanente training set è stato suddiviso in 17 soggetti bilanciati e composti da 108 sequenze di 160 punti ciascuna.
Usando come modello una rete neurale con due layer nascosti composti da 24 unità GRU e un layer di output con 4 unità aventi funzione di attivazione *softmax*, su ASCERTAIN con i soggetti fittizi sono stati registrati i seguenti risultati:

Scenario	Epoche	Tempo	Acc.	ACC	BWT	FWT	Memoria
Offline	29	66,17s	87,77	-	-	-	1945,91 Mb
Continual	7,75±5,33	226,24s	87,01±0,64	0,2481	-0,0241	0,0052	2143,71 Mb
Cumulative	7,25±8,80	872,8s	87,66±0,16	0,2773	0,0032	0,0801	2273,07 Mb
Replay	12,125±12,94	299,52s	87,13±0,75	0,2711	-0,0012	0,306	2146,35 Mb
Episodic	5,5±3,94	319,76s	82,19±3,95	0,339	0,0346	0,1043	2204,84 Mb
EWC	14,38±7,58	541,68s	87,34±0,42	0,2467	-0,0191	0,0845	2146,36 Mb
LWF	20,75±8,42	771,36s	86,91±0,89	0,2495	-0,0333	-0,146	2095,93 Mb

Tabella 4.5: *Risultati su ASCERTAIN con soggetti fittizi bilanciati*

Dalla tabella 4.5 si evince subito come il bilanciamento abbia prodotto accuratezze medie molto superiori al dataset originale: la metodologia offline raggiunge l'87,77% di accuratezza, paragonabile ai risultati ottenuti sul dataset WESAD, e le metodologie continue, in linea coi precedenti risultati, superano comunque l'80%

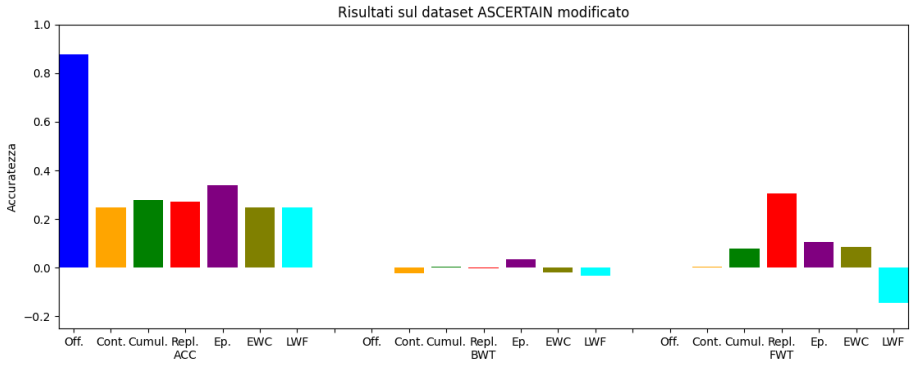

Figura 4.4: *Risultati su ASCERTAIN con soggetti fittizi*

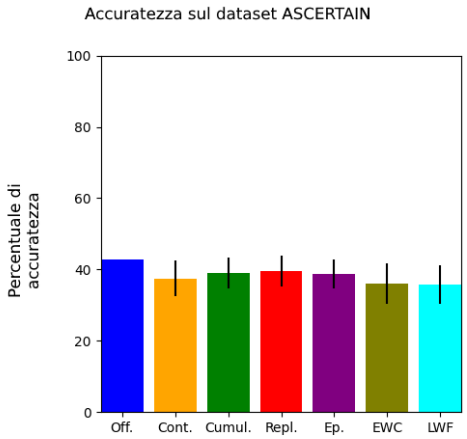

Figura 4.5: *Accuratezza sul dataset ASCERTAIN originale*

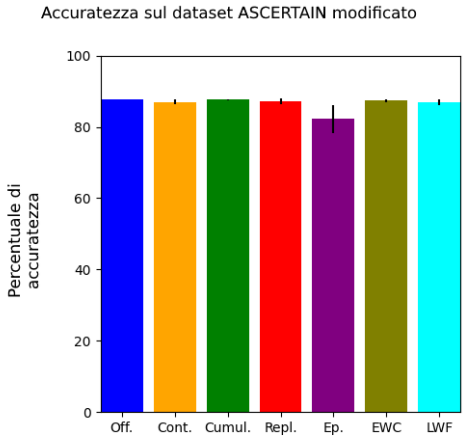

Figura 4.6: *Accuratezza su ASCERTAIN con soggetti fittizi*

di accuratezza media in tutti i casi.
Dalle rimanenti metriche si nota comunque la difficoltà intrinseca del dataset, che raggiunge un'accuratezza finale del solo 33% con la metodologia di replay episodico e BWT spesso negativo o comunque molto basso. Questa variazione del dataset ASCERTAIN dimostra come il bilanciamento dei dati di training sia fondamentale nel corretto addestramento di una rete neurale. L'avere dei soggetti di partenza bilanciati a livello di classi contenute, e di conseguenza esempi di addestramento non fortemente sbilanciati su una o più classi come nel caso del dataset ASCERTAIN originale, porta ad avere un'addestramento della rete neurale di miglior qualità e risultati finali sensibilmente migliori.

4.3 Metodologie Continue a Confronto

Dai risultati ottenuti e elencati nelle tabelle 4.3, 4.4 e 4.5 possiamo mettere a confronto fra loro le diverse metodologie di continual learning prese in considerazione.

Dataset	Epoche	Tempo	Acc.	ACC	BWT	FWT	Memoria
WESAD	49,14±33,67	947,24s	73,13±4,02	0,7721	0,0343	0,5397	2173 Mb
ASCERTAIN	10,88±5,01	249,28s	37,45±5,01	0,25	-0,0168	0,0213	2154,36 Mb
Cust. ASC.	7,75±5,33	226,24s	87,01±0,64	0,2481	-0,0241	0,0052	2143,71 Mb
Media	22,59	474,25s	65,86	0,4234	-0,0022	0,1887	2157,02 Mb

Tabella 4.6: *Risultati delle metodologia naive*

La metodologia continua più semplice ottiene sui dataset risultati sempre comparabili alla metodologia offline utilizzata come baseline. Come elencato in tabella 4.6, la metodologia continual su dataset WESAD ottiene un'accuratezza media del 73.13%, da confrontare con la metodologia offline con un'accuratezza del 99.07%. Questo primo risultato porta ad una considerazione: su dataset come WESAD, contente dati ben bilanciati nelle classi, già solo la metodologia continua più triviale ottiene risultati non ottimi ma comunque buoni, anche dimostrato dalla metrica BWT di 0.0343 e dall'FWT di 0.5397 che indicano un buon trasferimento della conoscenza.
ASCERTAIN d'altro canto ottiene solo un risultato del 42.78% offline, ma anche qua la metodologia continual ottiene un paragonabile 37.45% di accuratezza media, nonostante l'accuratezza finale solo del 25% (che su quattro classi da inferire è equivalente allo scegliere una classe in modo casuale). ASCERTAIN con i soggetti fittizi, invece, ottiene un 87.01% di accuratezza sulla metodologia naive a fronte dell'87.77% ottenuto dalla metodologia offline, dimostrando l'importanza delle classi bilanciate, ma ottenendo solo il 24.81% di accuratezza finale.

Dataset	Epoche	Tempo	Acc.	ACC	BWT	FWT	Memoria
WESAD	39,71±19,91	2786s	81,97±8,67	0,961	0,1383	0,4674	2291,45 Mb
ASCERTAIN	9,25±6,81	932,56s	39,06±4,26	0,2697	0,0064	0,0278	2297,17 Mb
Cust. ASC.	7,25±8,80	872,8s	87,66±0,16	0,2773	0,0032	0,0801	2273,07 Mb
Media	18,74	1530,45s	69,56	0,5027	0,0493	0,1918	2287,23 Mb

Tabella 4.7: *Risultati della metodologia cumulative*

La metodologia cumulativa, i cui risultati sono elencati nella tabella 4.7, evidenzia fin da subito l'effetto che il replay degli esempi ha sull'addestramento continuativo. Sul dataset WESAD, la metodologia cumulativa raggiunge un'accuratezza media dell'81.97% e finale del 96.10%, cioè quasi a livello con la metodologia offline che ottiene un'accuratezza del 99.07%. Anche BWT e FWT evidenziano un buon trasferimento della conoscenza, attestandosi rispettivamente su 0.1383 e 0.4674.
Il dataset ASCERTAIN evidenzia ancora una volta la propria difficoltà, con un'accuratezza media della metodologia cumulativa pari a 39.06% e finale del 26.97%,

comunque molto inferiore rispetto al 42.78% dell'addestramento offline. Con i soggetti bilanciati, otteniamo un'accuratezza media dell'87.66%, da comparare al quasi identico 87.77% di accuratezza della metodologia offline, ma un'accuratezza finale del 27.73%. In entrambi i casi, il BWT indica un trasferimento all'indietro della conoscenza quasi nullo, rispettivamente dello 0.0064 e dello 0.0032, e analogamente l'FWT, dello 0.0278 e dello 0.0801 rispettivamente.

Dataset	Epoche	Tempo	Acc.	ACC	BWT	FWT	Memoria
WESAD	41,14±21,19	1063,51s	78,29±3,32	0,7849	-0,002	0,4582	2184,77 Mb
ASCERTAIN	13,25±10,03	402,96s	39,48±4,37	0,2603	-0,014	0,0485	2173,95 Mb
Cust. ASC.	12,125±12,94	299,52s	87,13±0,75	0,2711	-0,0012	0,306	2146,35 Mb
Media	22,17	588,66s	68,30	0,4388	-0,0057	0,2709	2168,36 Mb

Tabella 4.8: *Risultati della metodologia con il 25% di replay*

Comparandolo al 99.07% della metodologia offline, sul dataset WESAD avere il 25% di replay fa ottenere risultati ovviamente inferiori all'81.97% della metodologia continuativo ma consente comunque di superare il 73.13% della metodologia naive, ottenendo un 78.29% di accurateza media e un 78.49% di accuratezza finale. Nonostante il BWT di -0.002, quasi nullo, si ha un buon trasferimento in avanti della conoscenza come provato dall'FWT di 0.4582.
ASCERTAIN ha un comportamento simile: la metodologia che usa il il 25% di replay ottiene il 39.48% di accuratezza media e il 26.03% di accuratezza finale che supera la metodologia continual e anche quella cumulativa. Lo stesso non si può dire di ASCERTAIN con i soggetti fittizi, che come per WESAD ottiene un'accuratezza sul 25% di replay che si attesta a metà tra la metodologia continual e quella cumulativa. I risultati sono elencati nella tabella 4.8.

Dataset	Epoche	Tempo	Acc.	ACC	BWT	FWT	Memoria
WESAD	35±26,26	1088,29s	82,13±6,60	0,9095	0,0841	0,4226	2097,47 Mb
ASCERTAIN	12,62±7,94	498,24s	38,80±4,04	0,2742	0,0048	0,0156	2231,42 Mb
Cust. ASC.	5,5±3,94	319,76s	82,19±3,95	0,339	0,0346	0,1043	2204,84 Mb
Media	17,71	635,43s	67,70	0,5076	0,0412	0,1808	2177,91 Mb

Tabella 4.9: *Risultati della metodologia episodica con* $m = 70$

Utilizzando un replay statico e bilanciato nelle classi, vale a dire la metodologia episodica i cui risultati sono elencati in tabella 4.9, su WESAD otteniamo risultati comparabili alla metodologia cumulativa pur usando quasi il 10% in meno di memoria: la metodologia episodica raggiunge l'82.13% di accuratezza media mentre quella cumulativa l'81.97%, ottenendo però un'accuratezza finale del 96.10% rispetto al 90.95% della metodologia episodica.
Sul dataset ASCERTAIN abbiamo una situazione simile: la metodologia cumulativa ottiene il 38.80% di accuratezza mentre con quella episodica si raggiunge il 39.06% usando il 3% in meno di memoria. Lo stesso discorso non si può dire per il dataset ASCERTAIN con i soggetti fittizi, che ottiene un'accuratezza media dell'82.19% inferiore all'87.66% della metodologia cumulativa.

Dataset	Epoche	Tempo	Acc.	ACC	BWT	FWT	Memoria
WESAD	29,71±17,38	1342,81s	70,74±4,74	0,7251	0,0113	0,4698	2187,40 Mb
ASCERTAIN	24,14±16,94	810,96s	36,08±5,66	0,2497	-0,0183	-0,0003	2171,62 Mb
Cust. ASC.	14,38±7,58	541,68s	87,34±0,42	0,2467	-0,0191	0,0845	2146,36 Mb
Media	22,74	898,48s	64,72	0,4075	-0,0087	0,1847	2168,46 Mb

Tabella 4.10: *Risultati della metodologia EWC*

Il primo dei due metodi provati che non si basano sui replay, l'Elastic Weight Consolidation nei risultati in tabella 4.10, presenta su WESAD un'accuratezza inferiore anche alla metodologia continua naive: quest'ultimo otteneva un'accuratezza media del 73.13% con un'accuratezza finale del 77.21%, mentre EWC raggiunge l'accuratezza media del 70.74% con un'accuratezza finale del 72.51%. Nonostante questo, il trasferimento di conoscenza è paragonabile, seppur inferiore, all'addestramento naive. Su ASCERTAIN il risultato presenta le medesime caratteristiche: con un'accuratezza media del 36.08% e un'accuratezza finale del 24.97% risulta di poco inferiore alla metodologia naive. Discorso diverso per ASCERTAIN con i soggetti fittizi, che riesce a ottenere un'accuratezza finale leggermente superiore all'addestramento naive con 87.34% rispetto al 87.01% e anche metriche BWT e FWT superiori.

Dataset	Epoche	Tempo	Acc.	ACC	BWT	FWT	Memoria
WESAD	44,29±18,30	3282,51s	69,09±5,82	0,7419	0,0451	0,3248	2121,31 Mb
ASCERTAIN	21,62±10,20	879,68s	35,69±5,43	0,2448	0.0327	0,0156	2103 Mb
Cust. ASC.	20,75±8,42	771,36s	86,91±0,89	0,2495	-0,0333	-0,146	2095,93 Mb
Media	28,89	1644,52s	63,90	0,4121	0,0148	0,0648	2106,75 Mb

Tabella 4.11: *Risultati della metodologia LWF*

Learning Without Forgetting, nei risultati in tabella 4.11, ha performance comparabili ad EWC. Su WESAD risulta inferiore alla metodologia naive, con 69.09% di accuratezza media contro il 73.13% raggiunto dalla metodologia naive e il 70.75% dell'EWC. Stesso discorso per ASCERTAIN, dove raggiunge il 35.69% contro il 37.45% della metodologia naive e il 42.78% raggiunto dalla metodologia offline, e per ASCERTAIN con soggetti fittizi. Quest'ultimo raggiunge un'accuratezza media inferiore dello 0.1% rispetto all'Addestramento Continuo naive.

Metodologia	Epoche	Tempo	Acc.	ACC	BWT	FWT	Memoria
Naive	22,59	474,25s	65,86	0,4234	-0,0022	0,1887	2157,02 Mb
Cumulative	18,74	1530,45s	69,56	0,5027	0,0493	0,1918	2287,23 Mb
Replay	22,17	588,66s	68,30	0,4388	-0,0057	0,2709	2168,36 Mb
Episodic	17,71	635,43s	67,70	0,5076	0,0412	0,1808	2177,91 Mb
EWC	22,74	898,48s	64,72	0,4075	-0,0087	0,1847	2168,46 Mb
LWF	28,89	1644,52s	63,90	0,4121	0,0148	0,0648	2106,75 Mb

Tabella 4.12: *Risultati medi delle varie metodologie*

CAPITOLO 5

Conclusioni

Questa tesi si poneva come obiettivo primario la realizzazione di uno studio riguardante le performance raggiunte da alcune metodologie di Apprendimento Automatico Continuo applicate a Reti Neurali Ricorrenti, in merito a problemi riguardanti l'ambito dello *Human State Monitoring*. Dai risultati raccolti durante gli esperimenti si può concludere che le metodologie di Apprendimento Automatico Continuo ottengono in generale risultati soddisfacenti su questa tipologia di dati, e sono pertanto metodologie di apprendimento applicabili in problemi di questo tipo.

Dai risultati raccolti e presentati nel capitolo 4 è evidente la differenza di accuratezza raggiunta sui due dataset. In particolare, nonostante si riesca facilmente ad ottenere un modello con risultati buoni o ottimi sul dataset WESAD, ciò non è stato ottenuto sul dataset ASCERTAIN originale mentre si hanno risultati sensibilmente migliori sullo stesso dataset ASCERTAIN modificato rendendo ogni soggetto più bilanciato riguardo le classi contenute.

In tutti i dataset, le metodologie continue hanno mostrato accuratezza media inferiore rispetto all'addestramento offline: sul dataset WESAD si ha una riduzione dell'accuratezza media del 23.17% in media sui vari approcci, mentre sul dataset ASCERTAIN si attesta sul 5.02%. Su ASCERTAIN modificato rendendo ogni soggetto più bilanciato, invece, la riduzione media è dell'1.39%, con la maggior parte delle metodologie che ottengono un'accuratezza inferiore di meno dell'1%.
Questa riduzione dell'accuratezza è sicuramente mitigabile attraverso la selezione di un modello con performance migliori delle metodologie continue, invece di eseguire la *model selection* sulla base dei risultati ottenuti nella metodologia offli-

ne. Con questi dati però si può già affermare che nella maggior parte dei casi le metodologie continue sono paragonabili all'addestramento classico, posto di avere un dataset con determinate caratteristiche: le classi all'interno del dataset di addestramento devono essere bilanciate fra loro, o si avrà un aumento di difficoltà nell'addestramento della rete neurale.

Si può quindi concludere che su certe tipologie di dati, il continual learning è applicabile all'ambito dello *human state monitoring* con un elevato grado di accuratezza e ottenendo modelli di Machine Learning in grado di ottenere risultati comparabili o simili alla metodologia offline. Il risultato più vicino all'addestramento offline è stato ottenuto con la metodologia cumulativa su dataset WESAD, con il replay al 25% sul dataset ASCERTAIN e di nuovo con l'addestramento cumulativo sul dataset ASCERTAIN con i soggetti bilanciati artificialmente. Questo dimostra che mantenere esempi precedenti porta ad un addestramento di maggior qualità e un modello più capace di inferire correttamente lo stato psico-fisico di una persona, posto di avere sufficiente memoria per lo stoccaggio dei dati di replay. Una possibile soluzione al problema della memorizzazione potrebbe essere addestrare un modello generativo, ovvero una rete neurale che fornita una classe del dataset genera delle feature che corrispondono a quella classe. Così facendo si può realizzare un replay generativo, artificiale, che non richiede la memorizzazione di istanze ma in compenso rende necessario l'addestramento di una seconda rete neurale totalmente diversa e le cui performance hanno diretto impatto sui risultati del modello predittivo.

Bibliografia

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng, *Tensorflow: A system for large-scale machine learning*, 2016.

[2] L. L. Ankile, M. F. Heggland, and K. Krange, *Deep convolutional neural networks: A survey of the foundations, selected improvements, and some current applications*, CoRR, abs/2011.12960 (2020).

[3] D. Bacciu, S. Akarmazyan, E. Armengaud, M. Bacco, G. Bravos, C. Calandra, E. Carlini, A. Carta, P. Cassarà, M. Coppola, et al., *Teaching-trustworthy autonomous cyber-physical applications through human-centred intelligence*, in 2021 IEEE International Conference on Omni-Layer Intelligent Systems (COINS), IEEE, 2021, pp. 1–6.

[4] D. Bank, N. Koenigstein, and R. Giryes, *Autoencoders*, CoRR, abs/2003.05991 (2020).

[5] M. Carranza-García, P. Lara-Benítez, J. García-Gutiérrez, and J. C. Riquelme, *Enhancing object detection for autonomous driving by optimizing anchor generation and addressing class imbalance*, Neurocomputing, 449 (2021), p. 229–244.

[6] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, *Learning phrase representations using RNN encoder-decoder for statistical machine translation*, CoRR, abs/1406.1078 (2014).

[7] F. Chollet, *Deep Learning with Python*, Manning Publications, 2017.

[8] T. Diethe, T. Borchert, E. Thereska, B. Balle, and N. Lawrence, *Continual learning in practice*, arXiv preprint arXiv:1903.05202, (2019).

[9] C. Gallicchio, A. Micheli, and L. Pedrelli, *Design of deep echo state networks*, Neural Networks, 108 (2018), pp. 33–47.

[10] X. Glorot, A. Bordes, and Y. Bengio, *Deep sparse rectifier neural networks*, in Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, G. Gordon, D. Dunson, and M. Dudík, eds., vol. 15 of Proceedings of Machine Learning Research, Fort Lauderdale, FL, USA, 11–13 Apr 2011, PMLR, pp. 315–323.

[11] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition*, O'Reilly Media, Inc., 2019.

[12] S. Haykin, *Feedforward neural networks : An introduction*, 2004.

[13] S. Hochreiter and J. Schmidhuber, *Long Short-Term Memory*, Neural Computation, 9 (1997), pp. 1735–1780.

[14] Y. Huang and Y. Chen, *Autonomous driving with deep learning: A survey of state-of-art technologies*, arXiv preprint arXiv:2006.06091, (2020).

[15] S. Jha, M. Schiemer, F. Zambonelli, and J. Ye, *Continual learning in sensor-based human activity recognition: An empirical benchmark analysis*, Information Sciences, 575 (2021), p. 1–21.

[16] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017.

[17] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, D. Hassabis, C. Clopath, D. Kumaran, and R. Hadsell, *Overcoming catastrophic forgetting in neural networks*, 2017.

[18] Z. Li and D. Hoiem, *Learning without forgetting*, IEEE transactions on pattern analysis and machine intelligence, 40 (2017), pp. 2935–2947.

[19] V. Lomonaco, *Continual learning for production systems*, Aug 2019.

[20] D. Lopez-Paz and M. Ranzato, *Gradient episodic memory for continuum learning*, CoRR, abs/1706.08840 (2017).

[21] M. McCloskey and N. J. Cohen, *Catastrophic interference in connectionist networks: The sequential learning problem*, vol. 24 of Psychology of Learning and Motivation, Academic Press, 1989, pp. 109–165.

[22] R. Pascanu, T. Mikolov, and Y. Bengio, *Understanding the exploding gradient problem*, CoRR, abs/1211.5063 (2012).

[23] A. Priyanshu, M. Sinha, and S. Mehta, *Continual distributed learning for crisis management*, 2021.

[24] M. Riedmiller and H. Braun, *Rprop - a fast adaptive learning algorithm*, tech. rep., Proc. of ISCIS VII), Universitat, 1992.

[25] M. B. Ring, *Child: A first step towards continual learning*, Machine Learning, 28 (2004), pp. 77–104.

[26] D. Rolnick, A. Ahuja, J. Schwarz, T. P. Lillicrap, and G. Wayne, *Experience replay for continual learning*, 2019.

[27] F. Rosenblatt, *The perceptron: A probabilistic model for information storage and organization in the brain.*, Psychological Review, 65 (1958), pp. 386–408.

[28] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, *Learning representations by back-propagating errors*, Nature, 323 (1986), pp. 533–536.

[29] P. Schmidt, A. Reiss, R. Duerichen, C. Marberger, and K. Van Laerhoven, *Introducing wesad, a multimodal dataset for wearable stress and affect detection*, in Proceedings of the 20th ACM International Conference on Multimodal Interaction, ICMI '18, New York, NY, USA, 2018, Association for Computing Machinery, p. 400–408.

[30] R. Subramanian, J. Wache, M. K. Abadi, R. L. Vieriu, S. Winkler, and N. Sebe, *Ascertain: Emotion and personality recognition using commercial sensors*, IEEE Transactions on Affective Computing, 9 (2018), pp. 147–160.

[31] K. B. J. G. R. C. M. M. B. J. H. S. M. A. van Bennekum; Andrew Hunt; Ken Schwaber; Alistair Cockburn; Ron Jeffries; Jeff Sutherland; Ward Cunningham; Jon Kern; Dave Thomas; Martin Fowler; Brian Marick, *Manifesto for agile software development*.

[32] G. M. van de Ven and A. S. Tolias, *Generative replay with feedback connections as a general strategy for continual learning*, 2019.

[33] ——, *Three scenarios for continual learning*, 2019.

[34] J. Wache, R. Subramanian, M. K. Abadi, R.-L. Vieriu, N. Sebe, and S. Winkler, *Implicit user-centric personality recognition based on physiological responses to emotional videos*, in Proceedings of the 2015 ACM on International Conference on Multimodal Interaction, ICMI '15, New York, NY, USA, 2015, Association for Computing Machinery, p. 239–246.

[35] J. Wei, J. M. Snider, T. Gu, J. M. Dolan, and B. Litkouhi, *A behavioral planning framework for autonomous driving*, in 2014 IEEE Intelligent Vehicles Symposium Proceedings, 2014, pp. 458–464.

Ringraziamenti

Questa tesi è la testimonianza di un percorso formativo che ho potuto intraprendere solamente grazie alla presenza e al supporto di alcune persone, senza le quali non sarei qua oggi.
Il primo ringraziamento va al prof. Lomonaco e ai prof. Gallicchio e Bacciu, per la disponibilità, la preparazione e la gentilezza dimostrata durante tutto il lavoro di sperimentazione e stesura di questa tesi. Vorrei anche ringraziare il prof. Russo, per avermi dimostrato cosa significa avere la passione nell'imparare e nell'insegnare. Ma questi ringraziamenti si estendono a tutte le professoresse e i professori che ho avuto il piacere di incontrare durante la mia carriera scolastica e universitaria, poiché ognuno di essi mi ha influenzato e di ognuno ne conservo i preziosi insegnamenti.
Ringrazio dal profondo del mio cuore i miei genitori, che con la loro presenza e il loro supporto mi hanno reso la persona che sono oggi, e i miei amici, che come colombe mi hanno salvato più volte nella vita.
Ringrazio mio fratello Simone, grazie al quale ho sviluppato alcune delle mie passioni più intense tra cui quella per l'informatica.
Ringrazio Jacopo Belli per il template con il quale ho steso questa tesi, e tutti gli studenti del corso di laurea di informatica per la solidarietà e disponibilità dimostrata negli anni, tra appunti condivisi e spiegazioni ad ogni ora del giorno.