Corso di Laurea in Informatica

TESI DI LAUREA

Analisi multivariata di ECG per la sindrome di Brugada con modelli ESN

Relatore:
Prof. Alessio Micheli

Correlatore:
Dr. Claudio Gallicchio

Correlatore:
Dr. Luca Pedrelli

Candidato:
Filippo Biondi

ANNO ACCADEMICO 2021/2022

Abstract

La sindrome di Brugada è una patologia cardiaca diagnosticabile mediante l'analisi dell'elettrocardiogramma (ECG). Sino ad ora, all'interno del progetto BrAID, sono stati utilizzati modelli di machine learning che sfruttano solo 3 derivazioni dell'ECG su un totale di 12 per effettuare la diagnosi, perché ritenute fra le più importanti dall'esperienza clinica. I dati provenienti dagli ECG sono rappresentati come serie temporali multivariate (cioè composte da più segnali, in questo caso le varie derivazioni), per questo motivo all'interno di questa tesi verranno analizzati e confrontati vari tipi di reti neurali ricorrenti, nell'ambito del reservoir computing, e verranno utilizzate tutte e 12 le derivazioni. L'uso di tutte le derivazioni rappresenta sia un vantaggio, in quanto si dispone di più informazioni per effettuare la diagnosi, sia una sfida poiché molte (probabilmente la maggior parte) di queste informazioni non sono rilevanti nel caso della sindrome di Brugada. Sarà quindi posta attenzione anche alla capacità dei modelli di discriminare fra derivazioni rilevanti e irrilevanti, oltre a valutare se questi modelli siano in grado di migliorare le performance ottenute con solo 3 derivazioni. I modelli utilizzati in questa tesi si focalizzeranno sul separare le derivazioni dell'ECG processandole separatamente, anziché unirle all'interno del modello come accade negli approcci tradizionali di reservoir computing come, ad esempio, nei modelli Echo State Network. Saranno quindi valutate le performance dei modelli introdotti relativamente ad una baseline (un modello Echo State Network) per valutare se separare le derivazioni possa essere utile ai fini della diagnosi della Sindrome di Brugada.

Indice

1 Introduzione **1**

1.1 L'elettrocardiogramma 1

1.2 Sindrome di Brugada 3

1.3 Motivazione della tesi 3

1.4 Obbiettivi della tesi 5

2 Machine learning **6**

2.1 Supervised learning 7

2.2 Parametri e iperparametri di un modello 7

2.3 Validazione e grid seach 8

2.4 Overfitting e underfitting 9

2.5 Regolarizzazione 10

2.6 K-fold cross validation 11

2.7 Test e model assessment 12

2.8 Il modello lineare 13

2.9 Reti neurali artificiali 16

2.10 Discesa del gradiente e overfitting 18

2.11 Reti neurali ricorrenti 19

3 Reservoir Computing **20**

3.1 Echo State Network 20

3.2 Input Routed Echo State Network 23

3.3 Interconnessioni fra sub-reservoir 25

3.4 Training degli iperparametri 26

4 Metodo utilizzato negli esperimenti **28**

5 Risultati **31**

5.1 ESN 31

5.2 Comitato ESN 32

5.3 IRESN 34

5.4 IRESN con training degli iperparametri 38

5.5 IRESN con interconnessioni 40

5.6 IRESN con sub-reservoir di dimensioni variabili 44

6 Analisi dei risultati **46**

Conclusioni **50**

Elenco delle figure

1.1 Posizionamento degli elettrodi 2
1.2 Onde e segmenti di un tracciato di un ECG 3
1.3 Confronto fra ECG di un paziente sano e ECG dei tre tipi di anomalie sull'ECG 4

2.1 rappresentazione grafica di una grid search nel caso di 2 iperparametri . . 9
2.2 Rappresentazione semplificata di situazioni di underfitting e overfitting in un problema di classificazione 10
2.3 Variazione dell'errore sui dati di training e su nuovi dati all'aumentare della complessità del modello 11
2.4 Esempio di k-fold cross validation nel caso k=5 12
2.5 Diagramma della struttura di un singolo neurone artificiale 17
2.6 Valore della funzione di loss sul training set e sul validation set all'aumentare del numero di epoche 19

3.1 Rappresentazione schematica del modello ESN 21
3.2 Rappresentazione schematica del modello IRESN 24

5.1 Curva della loss sul training set e validation set sul modello IRESN con 3 derivazioni 40
5.2 Curva della loss sul training set e validation set sul modello IRESN con 12 derivazioni 42
5.3 Curva della loss sul training set e validation set sul modello IRESN con 3 derivazioni e interconnessioni fra sub-reservoir 43
5.4 Curva della loss sul training set e validation set sul modello IRESN con 12 derivazioni e interconnessioni fra sub-reservoir 43

1. Introduzione

La Sindrome di Brugada è una patologia genetica che aumenta il rischio di aritmie causando la morte improvvisa in soggetti apparentemente sani [16]. La diagnosi della malattia viene effettuata mediante l'osservazione dell'elettrocardiogramma (ECG) nel quale spesso si presentano delle anomalie [22]. A volte in soggetti non sani l'ECG può presentarsi privo di anomalie, perciò, nei casi in cui si sospetta la presenza della sindrome, è possibile effettuare l'ECG dopo la somministrazione di alcuni farmaci antiaritmici che tendono a rendere più visibili le anomalie sull'ECG [15]. Al momento non esiste una cura farmacologica efficace e pertanto per i pazienti più a richio l'unica soluzione è l'utilizzo di un defibrillatore cardiaco impiantabile (ICD) [18], il che rende necessaria una rapida diagnosi dei soggetti malati.

Lo scopo del progetto BrAID è quello di affiancare ai medici uno strumento che riduca i tempi e semplifichi la procedura di diagnosi della malattia. La natura variabile degli elettrocardiogrammi dei pazienti rende particolarmente indicato l'utilizzo di tecniche di machine learning che, attraverso un insieme di ECG classificati da medici come positivi o negativi, permettono di inferire automaticamente le caratteristiche tipiche degli ECG dei pazienti positivi e quindi classificare correttamente (ovviamente con una certa probabilità di errore) gli ECG dei nuovi pazienti. Fra i vari modelli di machine learning un particolare tipo di reti neurali ricorrenti [6], le Echo State Network [7], sono già state testate, all'interno del progetto BrAID, per la diagnosi della Sindrome di Brugada mostrando buone potenzialità [3]. Verrà quindi utilizzato questo modello come riferimento e verranno sperimentati altri modelli derivati dalle Echo State Network.

1.1 L'elettrocardiogramma

L'elettrocardiogramma è un esame diagnostico eseguito con uno strumento chiamato elettrocardiografo che permette di registrare la differenza di potenziale che si sviluppa nel cuore dovuta alle depolarizzazioni e ripolarizzazioni che avvengono durante il ciclo car-

diaco (ossia durante ogni battito). Per misurare questa differenza di potenziale si posizionano 10 elettrodi sulla pelle attraverso i quali si ricavano 12 derivazioni (o lead in inglese). Quattro elettrodi vengono posizionati sugli arti (uno per ogni arto) mentre altri 6 vengono posizionati sulla cassa toracica attorno al cuore [11] come mostrato nella Figura 1.1. Tre derivazioni vengono ricavate utilizzando a coppie gli elettrodi RA, LA e LL (queste derivazioni sono dette bipolari), le altre derivazioni (dette unipolari) sono ottenute utilizzando un elettrodo e un riferimento chiamato terminale centrale di Wilson come polo negativo, che si ottiene calcolando la media dei potenziali sui tre arti [5]. Si ottengono così altre 9 derivazioni (3 dagli elettrodi sugli arti e 6 dagli elettrodi sul torace) che unite alle 3 derivazioni bipolari formano le 12 derivazioni di un comune elettrocardiogramma.

Sul tracciato di queste derivazioni è possibile individuare delle onde indentificate dalle lettere P, Q, R, S, T e U che sono cruciali per la diagnosi delle patologie, inoltre queste onde delineano dei segmenti altrettanto importanti (le onde e i segmenti sono rappresentate nella Figura 1.2) [5].

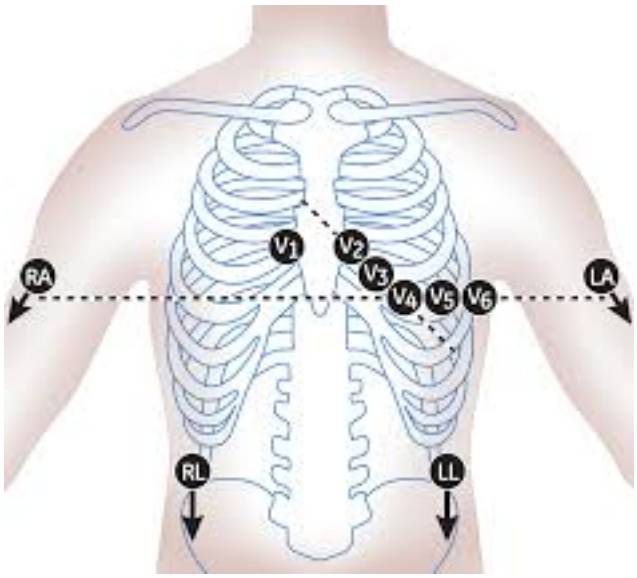

Figura 1.1: Posizionamento degli elettrodi

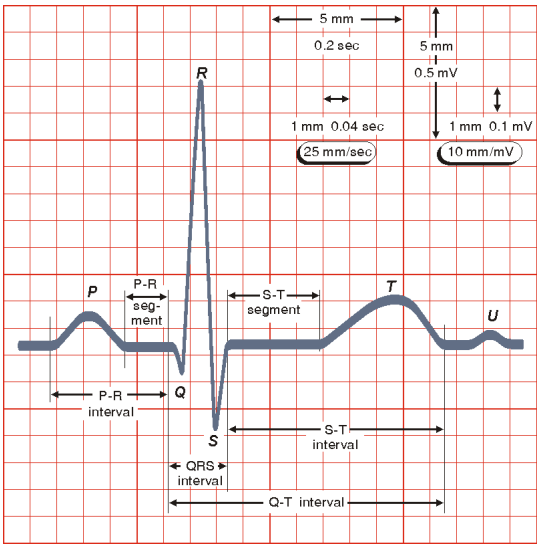

Figura 1.2: Onde e segmenti di un tracciato di un ECG

1.2 Sindrome di Brugada

Essendo la sindrome di Brugada una patologia cardiaca, il principale metodo di diagnosi è l'elettrocardiogramma. Nello specifico si presentano delle anomalie nel tratto ST delle derivazioni V1, V2 e V3 [22]. Sono stati poi distinti 3 modi differenti in cui la patologia può manifestarsi sull'ECG (Figura 1.3), ed è richiesto che si presenti il tipo 1 per confermare la diagnosi, mentre negli altri casi è indicato eseguire ulteriori indagini (eventualmente utilizzando farmaci antiaritmici) per accertare la presenza effettiva della patologia [16]. La diagnosi della sindrome di Brugada dall'ECG è quindi un compito arduo e di competenza dei medici che, osservando l'ECG, identificano la presenza o meno della sindrome. L'utilizzo del machine learning rappresenta un'opportunità per sviluppare uno strumento in grado di aiutare i medici nel processo di diagnosi della sindrome in modo da renderlo disponibile a quante più persone possibili.

1.3 Motivazione della tesi

Poiché i dati provenienti dall'ECG sono delle serie temporali, in cui quindi l'ordine dei dati è molto importante, l'approccio più comune è l'utilizzo di reti neurali ricorrenti, la cui

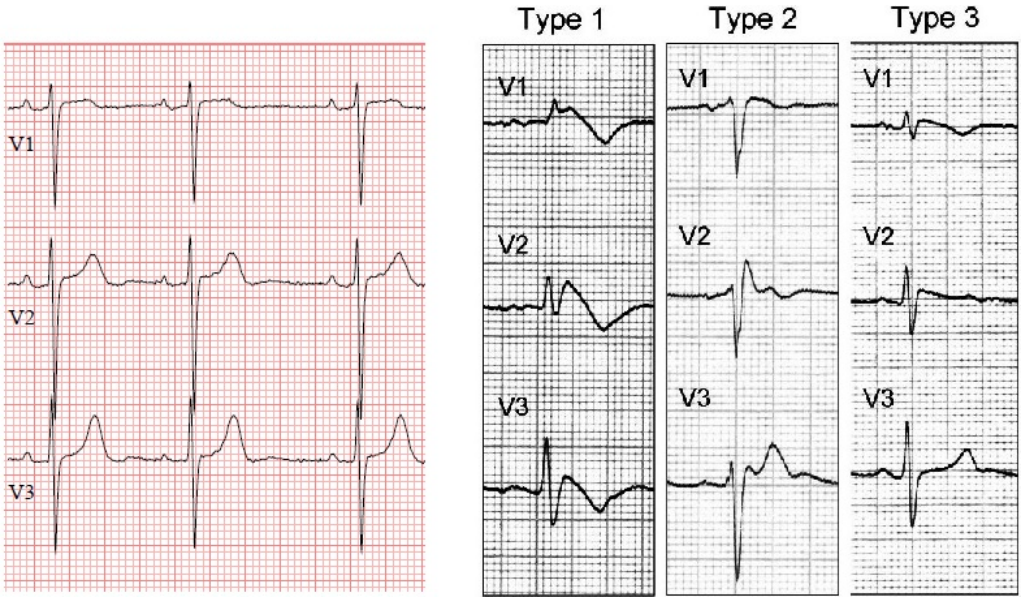

Figura 1.3: Confronto fra ECG di un paziente sano e ECG dei tre tipi di anomalie sull'ECG

struttura si adatta maggiormente a compiti in cui gli input sono costituiti da delle sequenze ordinate. All'interno di questa tesi sono utilizzate reti neurali ricorrenti che sfruttano i principi del reservoir computing, come ad esempio le Echo State Network (ESN) o le Input Routed Echo State Network (IRESN). Questi modelli sono utilizzati per via della loro semplicità (sia nella struttura del modello, sia nel metodo utilizzato per allenare il modello) e della loro efficienza. Fino ad ora nell'ambito del progetto BrAID sono stati utilizzati modelli che analizzavano le tre derivazioni ritenute più rilevanti per la diagnosi della sindrome di Brugada (V1, V2 e V3), ignorando le altre [3]. Benché questo approccio semplifichi il problema eliminando a priori la necessità di un modello di machine learning in grado di individuare le derivazioni più rilevanti (utilizzando invece la conoscenza presente in letteratura medica per stabilire quali esse siano), il rischio è quello di perdere delle informazioni rilevanti presenti nelle derivazioni scartate che invece potrebbero essere utili per classificare correttamente gli ECG dei pazienti. Affinchè il modello riesca a distinguere a quali derivazioni dare più importanza è utile che il modello tratti in maniera separata le varie derivazioni. Verranno quindi analizzate diverse strategie e modelli per elaborare indipendentemente le dimensioni dell'input, come ad esempio modelli a comitato o più varianti del modello IRESN (che a differenza del modello ESN possiede una sottorete

ricorrente per ogni dimensione dell'input e, solo una volta processate tutte dimensioni dell'input, ne unisce i risultati). Questa analisi verrà anche effettuata nel caso delle tre derivazioni classiche (V1, V2 e V3) per valutare se la separazione delle derivazioni possa giovare anche quando non è necessario individuare le derivazioni più importanti. Verrà anche affrontato il problema dell'aumento esponenziale del numero delle combinazioni dei possibili valori degli iperparametri, che sorge nei modelli IRESN quando si aumenta il numero di dimensioni dell'input. Molti iperparametri del modello verranno trasformati in parametri in modo che il loro valore sia determinato automaticamente nella fase di training del modello.

1.4 Obbiettivi della tesi

L'obbiettivo della tesi è quindi confrontare vari modelli di reservoir computing per l'analisi di sequenze temporali multivariate, cioè sequenze relative a dati che variano nel tempo con più dimensioni (in questo caso particolare ogni derivazione corrisponde ad una dimensione), valutando le loro performance rispetto ad una baseline che utilizza solo 3 derivazioni. Inoltre è interessante osservare quali fra le 12 derivazioni saranno indicate dai vari modelli come più rilevanti e confrontare questi risultati con quelli forniti dalla conoscenza medica al riguardo. Misurare le prestazioni dei modelli utilizzando tutte e 12 le derivazioni, nonostante sia già noto quali siano le più importanti, fornisce inoltre una misura della qualità dei modelli testati nell'apprendere automaticamente l'importanza delle dimensioni dell'input. Infatti per problemi in cui non è noto quali siano le dimensioni più rilevanti, non potendole selezionare a priori, l'impiego di tali modelli potrebbe migliorare i risultati ottenuti.

2. Machine learning

Col termine Machine Learning si indica un insieme di discipline e tecniche con l'obbiettivo di sviluppare un programma in grado di apprendere dall'esperienza [12]. Tipicamente si distinguono tre tipi di machine learning: supervised learning, unsupervised learning e reinforcement learning. Nel primo caso si ha a disposizione un insieme di esempi costituiti da coppie di input e output desiderato, e l'obbiettivo è, mediante algoritmi di apprendimento, ottenere un modello che produca output corretti sia per gli input presenti negli esempi che con input che non ha mai visto (la capacità del modello di predire correttamente l'output anche per input che non ha mai visto è chiamata generalizzazione) [19]. I problemi di supervised learning si suddividono a loro volta in due categorie: i problemi di regressione, nei quali gli output assumono valori nel continuo e i problemi di classificazione nei quali gli output possono assumere solo un numero finito di valori (ogni valore viene detto classe). Nei problemi di unsupervised learning invece non si hanno a disposizione degli esempi e pertanto lo scopo in questo tipo di problemi è diverso rispetto ai problemi di supervised learning (ad esempio situazioni tipiche in cui viene utilizzato l'unsupervised learning sono i problemi di clusterizzazione dove l'insieme di dati forniti deve essere suddiviso in più cluster, ossia dei gruppi i cui elementi che li compongono condividono caratteristiche simili). Infine nel reinforcement learning, tipicamente utilizzato in situazioni in cui va presa una decisione o una sequenza di decisioni, al modello viene dato un rinforzo che può essere una ricompensa se le decisioni prese hanno avuto un esito positivo o una punizione se l'esito è negativo e con queste il modello apprende (senza che gli venga fornita la soluzione corretta come nel caso del supervised learning). Il problema affrontato in questa tesi, ossia la diagnosi della sindrome di Brugada dall'ECG del paziente, è un problema di supervised learning, nello specifico di classificazione (le classi sono solo due: positivo o negativo), e pertanto nelle prossime sezioni verranno introdotti i principali concetti solo del supervised learning relativo ai problemi di classificazione.

2.1 Supervised learning

Da un punto di vista matematico nei problemi di supervised learning abbiamo a disposizione un insieme D, chiamato dataset, composto da coppie (x_i, y_i) dove y_i è l'output corrispondente all'input x_i. L'obbiettivo è trovare una funzione h chiamata ipotesi che approssimi nel miglior modo possibile una funzione ignota di cui si conoscono solo un numero finito di punti, ossia i valori contenuti nel dataset [19]. Per trovare questa funzione si effettua una ricerca all'interno di uno spazio, chiamato spazio delle ipotesi (spesso indicato con H), che consiste di tutte le possibili funzioni che il modello che viene utilizzato è in grado di esprimere. Tanto più è complesso il modello tanto più ampio sarà lo spazio delle ipotesi nel quale effettuare la ricerca.

Per misurare la bontà di un'ipotesi rispetto al dataset che abbiamo a disposizione è necessario utilizzare una metrica per misurare l'errore, solitamente indicata come $E(h)$ o $Loss(h)$, la più comune è l'errore quadratico medio indicato con MSE (dall'inglese Mean Squared Error) definito come:

$$MSE(h) = \frac{\sum_{i=1}^{p}(y_i - h(x_i))^2}{p}$$

dove p è il numero di esempi presenti nel dataset.

La ricerca all'interno dello spazio delle ipotesi sarà quindi guidata da questa metrica cercando di rendere il suo valore quanto più piccolo possibile (essendo una somma di quadrati il minimo di questa funzione è zero). Proprio perchè l'errore deve essere minimizzato la presenza di costanti nella formula non è rilevante perciò spesso come metrica si utilizza:

$$E(h) = \frac{1}{2}\sum_{i=1}^{p}(y_i - h(x_i))^2$$

in quanto (come si vedrà più avanti) risulta essere più comoda [13].

2.2 Parametri e iperparametri di un modello

L'allenamento di un modello consiste quindi nel modificare, opportunamente guidati dalla metrica scelta, dei valori che costituiscono il modello: questi valori sono chiamati pa-

rametri o pesi del modello. Ci sono inoltre dei valori che rimangono costanti durante l'allenamento del modello, e vengono chiamati iperparametri (spesso fondamentali per determinare le caratteristiche di un modello). I parametri di un modello spesso vengono inizializzati in modo randomico e poi modificati durante la fase di training, mentre gli iperparametri devono essere fissati fin dall'inizio. Per scegliere un buon assegnamento degli iperparametri (visto che determinare a priori quale siano i valori migliori è quasi sempre impraticabile) è necessario eseguire quella che viene chiamata validazione.

2.3 Validazione e grid seach

Per eseguire la validazione di un modello il dataset viene suddiviso in due parti, comunemente chiamate training set e validation set. Vengono istanziati più modelli, ognuno con diversi iperparametri, allenati utilizzando il training set e in seguito viene scelto il migliore misurandone le prestazioni (sempre calcolando l'errore con una metrica) sul validation set. Per selezionare i valori degli iperparametri da testare una fra le tecniche più utilizzate è la grid search. Nella grid search vengono stabiliti dei valori da testare per ogni iperparametro e si testano tutte le possibili combinazioni di tali valori. Nella figura 2.1 è possibile vedere rappresentato lo spazio degli iperparametri e la griglia utilizzata per coprirlo (che da il nome alla strategia di ricerca) in un caso in cui gli iperparametri sono 2. La rappresentazione grafica è semplice in questo esempio bidimensionale e con un po' di immaginazione può essere estesa al caso tridimensionale, ma in un generico caso con n iperparametri è molto difficile visualizzare la griglia ed è quindi più utile immaginare la ricerca come il test di tutte le possibili combinazioni dei valori.

Il test per stabilire le performance di un particolare assegnamento di valori viene eseguito sul validation set che, non essendo stato coinvolto nella fase di training, è in grado di verificare le performance del modello su dati mai visti. Il fatto che i dati non siano mai stati visti è molto importante in quanto misurare le performance solo su dati sui quali il modello si è allenato può portare ad una situazione nota col nome di overfitting.

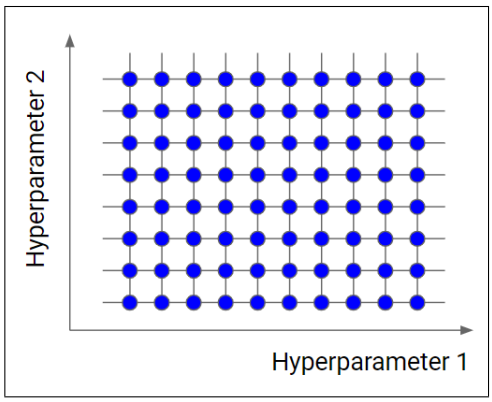

Figura 2.1: rappresentazione grafica di una grid search nel caso di 2 iperparametri

2.4 Overfitting e underfitting

L'overfitting in machine learning è una situazione che si verifica quando il modello performa bene sui dati di training ma poi su nuovi input mai visti le performance calano drasticamente. Più formalmente si dice che un ipotesi $h \in H$ è in overfitting sul training set se esiste un'altra ipotesi $h' \in H$ tale che h ha un minor errore di h' sugli esempi di training ma h' ha un errore minore di h su tutti i possibili input [14]. Questo accade perché nello spazio di ricerca (specialmente se molto grande) sono spesso presenti funzioni troppo complesse che si adattano troppo ai dati di training. Nei dati di training però è spesso presente del rumore, degli errori o delle incongruenze (che nei problemi in cui viene applicato il machine learning sono comuni) e quindi sui nuovi dati queste funzioni hanno delle performance basse.

Una prima soluzione potrebbe essere quella di limitare la complessità delle funzioni, utilizzando quindi un modello in grado di esprimere meno funzioni, ma il rischio è quello di riscontrare un altro problema (opposto all'overfitting) ossia l'underfitting. L'underfitting si verifica quando il modello utilizzato non è in grado di esprimere funzioni sufficientemente complesse per approssimare la funzione incognita e ciò si traduce in performance basse sia sui dati di training che sui nuovi dati. Nella Figura 2.2 è sono messe a confronto tre possibili soluzioni ad un problema di classificazione (con due dimensioni) dove l'obbiettivo è separare i punti delle due diverse classi utilizzando una funzione. Nel caso dell'underfitting la funzione utilizzata è una retta, una funzione molto rigida, che non

é in grado di separare sufficientemente bene i punti. Nel caso dell'overfitting la funzione è troppo flessibile, così flessibile che classifica correttamente tutti i punti del dataset, ma è evidente come questa funzione su nuovi dati non sia adatta, mentre una buona soluzione, sufficientemente flessibile ma non troppo, é quella centrale. Per ottenere una buona soluzione si sceglie un modello in grado di esprimere soluzioni sufficientemente complesse e si va a controllare la complessità della funzione scelta introducendo un ulteriore iperparametro per effettuare quella che è chiamata regolarizzazione del modello.

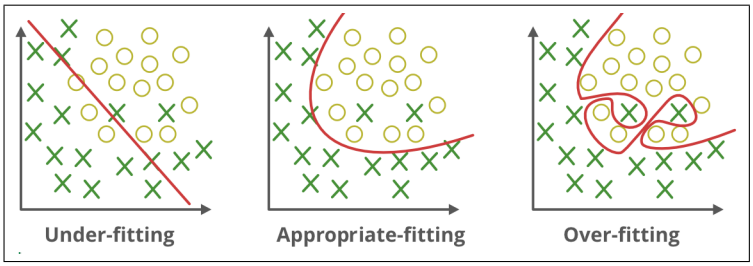

Figura 2.2: Rappresentazione semplificata di situazioni di underfitting e overfitting in un problema di classificazione

2.5 Regolarizzazione

Regolarizzare un modello significa limitarne la complessità, spesso mediante uno o più iperparametri, condizionando la fase di training del modello in modo da penalizzare funzioni troppo complesse. Ovviamente a seconda di quanto si penalizza la complessità di una funzione si otterrà un differente compromesso fra semplicità della funzione ottenuta alla fine del training e prestazioni sul training set, ed è quindi fondamentale selezionare correttamente il valore degli iperparametri responsabili della regolarizzazione tramite la validazione, durante la quale, utilizzando una porzione di dati differente rispetto a quella usata in fase di training, è possibile stimare se il modello è in overfitting, in underfitting o ha raggiunto un giusto compromesso. La Figura 2.3 mostra chiaramente come l'errore sui nuovi dati all'aumentare della complessità del modello forma una parabola, nel cui minimo si trova il miglior compromesso fra underfitting e overfitting.

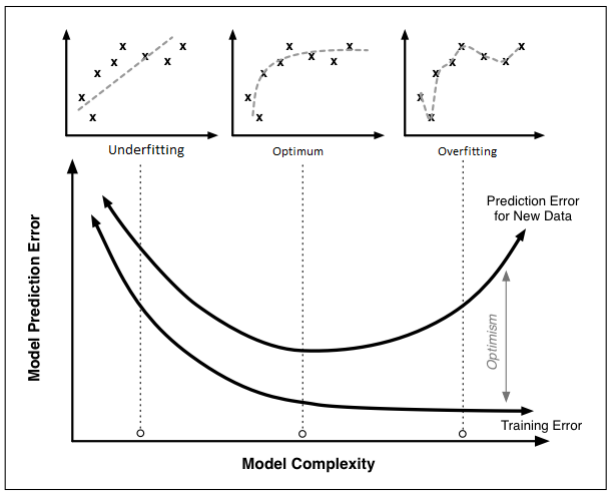

Figura 2.3: Variazione dell'errore sui dati di training e su nuovi dati all'aumentare della complessità del modello

2.6 K-fold cross validation

Quando si vuole validare un modello è necessario suddividere il dataset in 2 parti disgiunte: training set e validation set. Questo nel caso di dataset non molto grandi può essere un problema in quanto le performance (sia in training che in validation) saranno molto più suscettibili al modo col quale viene suddiviso il dataset (che essendo randomico per dataset grandi non rappresenta un problema) e quindi i risultati ottenuti durante la validazione saranno statisticamente meno rilevanti. Una tecnica che può essere utilizzata per ovviare a questo problema è la K-fold cross validation, che consiste nel suddividere il dataset in k porzioni (in inglese chiamate fold) e per ogni assegnamento degli iperparametri che si vuole testare si considerano k istanze del modello, ognuna allenata su k-1 porzioni e si misurano le prestazioni nella porzione restante (Figura 2.4). Ognuna delle k istanze sarà però allenata su un diverso sottoinsieme di k-1 elementi delle k porzioni e valutato sulla restante ed infine verrà effettuata la media fra i risultati delle k istanze per ottenere il risultato finale della validation. In questo modo si ottiene una stima migliore da utilizzare per la validazione in quanto tutti i dati del dataset vengono utilizzati e quindi il risultato è meno suscettibile al modo in cui vengono divisi training set e validation set. Infine una volta terminata la fase di validazione è possibile riallenare il modello su tutto il dataset

utilizzando come iperparametri quelli trovati durante la cross validation per sfruttare al massimo tutti i dati presenti nel dataset.

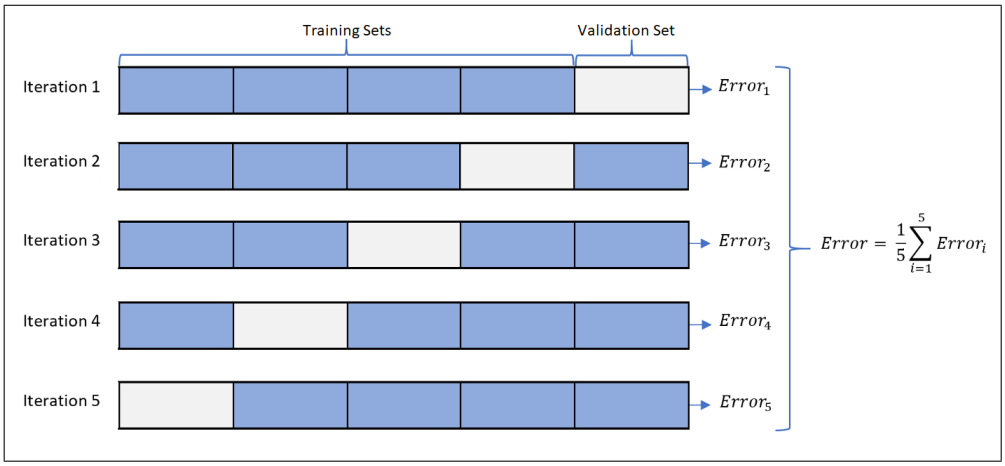

Figura 2.4: Esempio di k-fold cross validation nel caso k=5

2.7 Test e model assessment

Se si desidera avere una stima delle performance del modello ottenuto dopo una fase di validazione non è opportuno ritenere i risultati di quest'ultima fase come rappresentativi (questo è ovvio nel caso in cui si rialleni il modello dopo la validazione su tutto il dataset), infatti bisogna tenere conto del fatto che il modello scelto al termine della validazione è quello che ha ottenuti i risultati migliori cioè quello che si adattava meglio ai dati presenti nel dataset. Per ottenere invece una corretta stima delle performance del modello è necessario testarlo su una porzione di dati che non sono stati utilizzati per il training o validation del modello. Questi dati tipicamente prendono il nome di test set e questa pratica prende il nome di model assessment. È fondamentale che i dati contenuti nel test set non siano in alcun modo utilizzati durante le fasi precedenti al test (che è l'ultima fase) neanche in modo indiretto (ad esempio ripetendo il training sul training set o cambiando gli iperparametri se non si è soddisfatti dei risultati di test) altrimenti si otterranno dei valori non realistici e quando il modello sarà utilizzato in contesti reali le sue performance rischiano di essere inferiori rispetto a quelle stimate con la model assessment.

È possibile utilizzare la k-fold cross validation anche per ottenere una stima migliore dei risultati di test. In questo caso si otterranno k modelli diversi delle cui performance sarà fatta la media. Inoltre è possibile combinare la k-fold cross validation sia per la validazione che per il test facendo quella che è chiamata double cross validation (o nested cross validation) facendo una cross validation esterna per il test e, per ogni iterazione, effettuare un'altra cross validation per la validazione (il numero di fold esterne e interne non deve necessariamente essere lo stesso).

2.8 Il modello lineare

Un primo esempio di un modello molto semplice è il modello lineare, in cui sono presenti n+1 parametri $w_0, w_1, ..., w_n$ quando l'input è un vettore lungo n $\boldsymbol{x}^T = [x_1, .., x_n]$

Lo spazio delle ipotesi del modello è costituito da tutte le funzioni della forma:

$$h(x) = w_0 + \sum_{i=1}^{n} w_i x_i = w_o + \boldsymbol{w}^T \boldsymbol{x}$$

L'utilizzo più immediato di questo modello è nei problemi di regressione in cui si suppone che la funzione incognita sia lineare (dato che questo modello è in grado di esprimere solo funzioni lineari), ma è possibile utilizzarlo anche nei problemi di classificazione utilizzando l'ipotesi individuata per suddividere lo spazio degli inupt in due regioni. Un caso particolarmente semplice è quello in cui l'input ha dimensione uno, e quindi le funzioni che il modello può esprimere sono tutte le rette in $\mathbb{R}^2$ (tranne le rette parallele all'asse y) e quindi durante la fase di training verranno selezionati i parametri (che altro non sono che il coefficiente angolare e il termine noto della retta) affinché questi minimizzino l'errore E la cui formula in questo caso particolare diventa:

$$E(w) = \frac{1}{2} \sum_{i=1}^{p} (y_i - w_1 x - w_0))^2$$

Essendo questa una funzione differenziabile, è possibile calcolarne le derivate parziali rispetto a w_0 e w_1 per trovare il gradiente che in questo caso è:

$$\nabla E(w) = -\begin{bmatrix} \sum_{i=1}^{p} y_i - w_1 x_i - w_0 \\ \sum_{i=1}^{p} (y_i - w_1 x_i - w_0) x_i \end{bmatrix}$$

Ponendo il gradiente a 0 e risolvendo le equazioni che ne risultano (ricordando che le incognite sono w_0 e w_1) si ottiene il seguente risultato:

$$w_1 = \frac{p\sum_{i=1}^{p} x_i y_i - \sum_i x_i \sum_{i=1}^{p} y_i}{p\sum_{i=1}^{p} x_i^2 - (\sum_{i=1}^{p} x_i)^2}$$
$$w_0 = \bar{y} - w_1 \bar{x}$$

Dove $\bar{x}$ e $\bar{y}$ sono rispettivamente la media aritmetica degli x_i e degli y_i.

Questo metodo può essere esteso al caso n-dimensionale (dove quindi gli input sono in $\mathbb{R}^n$) nel quale per semplicità si pone all'inizio del vettore di input un valore uguale a 1 in modo che tale valore sia moltiplicato per il coefficiente relativo al termine noto, chiamato anche bias, per poter scrivere il modello nella forma concisa:

$$h(x) = \sum_{i=0}^{n} w_i x_i = \boldsymbol{w}^T \boldsymbol{x}$$

dove $\boldsymbol{w} = [w_0, \ldots, w_n]$ e $\boldsymbol{x} = [1, x_1, \ldots, x_n]$

La funzione di loss diventa quindi:

$$E(w) = \frac{1}{2}\sum_{i=1}^{p} (y_i - \boldsymbol{w}^T \boldsymbol{x}_i)^2 = \frac{1}{2}\sum_{i=1}^{p} (y_i - \boldsymbol{w}^T \boldsymbol{x}_i)(y_i - \boldsymbol{w}^T \boldsymbol{x}_i) = \frac{1}{2}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^T (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})$$

Dove $\boldsymbol{X}$ è una matrice $p \times n$ in cui ogni riga equivale a uno dei vettori di input del dataset (esteso con un 1 in prima posizione) e $\boldsymbol{y}$ è un vettore costituito dagli output del dataset (ovviamente nello stesso ordine utilizzato per la matrice $\boldsymbol{X}$) Derivando e ponendo il gradiente uguale a zero (come fatto nel caso precedente) si ottiene la seguente equazione:

$$-\boldsymbol{X}^T (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}) = \boldsymbol{0}$$

la cui soluzione si ricava calcolando

$$\boldsymbol{w} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T \boldsymbol{y}$$

Come esposto in precedenza una soluzione del genere (che come unico obbiettivo ha la minimizzazione dell'errore sui dati di training) può incorrere, anche nel caso di un modello semplice come quello lineare, nel problema dell'overfitting, che in questo modello particolare si manifesta con un aumento, in modulo, del valore dei parametri. Per combattere questo fenomeno è possibile regolarizzare il modello usando la ridge regression (o Tikhonov regularization) che consiste nell'aggiungere una penalità proporzionale al valore dei pesi del modello. In questo modo la funzione di loss da minimizzare diventa:

$$E(w) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \boldsymbol{w}^T\boldsymbol{x}_i)^2 + \frac{1}{2}\lambda\sum_{i=1}^{n}w_i^2 = \frac{1}{2}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w})^T(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w}) + \frac{1}{2}\lambda\boldsymbol{w}^T\boldsymbol{w}$$

dove λ è un iperparametro che regola quanto regolarizzare il modello, un λ alto implica una grande importanza della somma dei quadrati dei pesi nella funzione da minimizzare e quindi tenderà a rendere la funzione scelta una funzione meno complessa (quindi a ridurre l'overfitting ma col rischio di finire in underfitting), mentre un λ più basso porta a dare poca importanza alla somma dei quadrati dei pesi (fino all'estremo in cui $\lambda = 0$ che coincide col caso trattato in precedenza). A questo punto derivando si ottiene:

$$-\boldsymbol{X}^T(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w}) - \lambda\boldsymbol{w} = \boldsymbol{0}$$

e quindi il risultato viene calcolato con la formula:

$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

Un'altro tipo di regolarizzazione è la cosiddetta lasso regression, che, esattamente come la ridge regression, impone una penalità relativa alla somma dei pesi, ma anzichè utilizzare la somma del quadrato dei pesi utilizza la somma del valore assoluto. La funzione di loss in questo caso è:

$$E(w) = \frac{1}{2}\sum_{i=1}^{n}(y_i - \boldsymbol{w}^T\boldsymbol{x}_i)^2 + \lambda\sum_{i=1}^{n}|w_i| = \frac{1}{2}(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w})^T(\boldsymbol{y}-\boldsymbol{X}\boldsymbol{w}) + \lambda||\boldsymbol{w}||_1$$

Il vantaggio della lasso regression è che tende ad azzerare il valore dei pesi meno rilevanti, permettendo quindi di escludere alcune dimensioni dell'input.

Con questo tipo di regolarizzazione però la funzione di loss non è più differenziabile, quindi non è possibile utilizzare il metodo di risoluzione utilizzato finora. Per allenare il modello lineare quando si utilizza la lasso regression sono quindi necessari metodi di ottimizzazione, tipicamente iterativi, che permettono di trovare il minimo di una funzione convessa anche quando la funzione non è differenziabile [25].

2.9 Reti neurali artificiali

Spesso la complessità del modello lineare non è sufficiente per esprimere la funzione incognita e questo conduce inevitabilmente all'underfitting. Quando si ha necessità di modelli in grado di esprimere funzioni non lineari si può ricorrere alle reti neurali artificiali (ANN dall'inglese artificial neural network). Il componente elementare di un ANN è il neurone artificiale (spesso chiamato perceptrone anche se il perceptrone è un particolare tipo di neurone artificiale [17]) che prende in input una serie di valori e restituisce come output il risultato di una funzione non lineare (nel caso del perceptrone la funzione di attivazione è la funzione segno), detta funzione di soglia o di attivazione, applicata ad una combinazione lineare degli input. Lo schema di un singolo neurone artificiale può essere osservato nella Figura 2.5. Nel tipo di ANN più semplice, le feedforward neural network, questi neuroni, che prendono anche il nome di nodi, sono organizzati in strati (layer in inglese) in modo che ogni nodo di uno strato riceva in inupt gli output di tutti i nodi dello strato precedente (i nodi del primo strato avranno come input i valori di input dell'intera rete). I parametri del modello sono i pesi mediante i quali i nodi calcolano la combinazione lineare degli input, mentre gli iperparametri sono il numero di strati, il numero di nodi per ogni strato e la funzione di attivazione da utilizzare.

Avendo a che fare con funzioni non lineari minimizzare la funzione di loss per eseguire il traning del modello risulta più complicato, non è infatti possibile ottenere una formula chiusa per il calcolo dei parametri (come per il modello lineare) ma è comunque possibile calcolare il gradiente della funzione di loss se la funzione di attivazione usata nei neuroni è differenziabile. Per un singolo neurone, indicando con o l'output del neurone (cioè il risultato dell'applicazione della funzione di attivazione) con *net* il risultato della

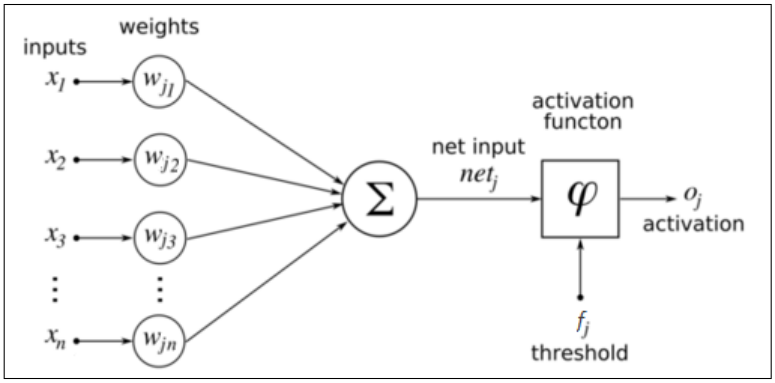

Figura 2.5: Diagramma della struttura di un singolo neurone artificiale

combinazione lineare, utilizzando la chain rule otteniamo che

$$\frac{\partial E}{\partial w_i} = \frac{\partial E}{\partial o} \cdot \frac{\partial o}{\partial w_i} = \frac{\partial E}{\partial o} \cdot \frac{\partial o}{\partial net} \cdot \frac{\partial net}{\partial w_i} = -\sum_{k=1}^{p} (y_k - o_k) \frac{\partial f(net)}{\partial net} x_{ki}$$

Questo gradiente può essere utilizzato per guidare l'aggiornamento dei parametri del modello, infatti il gradiente ci dirà in quale direzione cresce il valore della loss e quindi muovendosi nella direzione opposta quest'ultima si ridurrà (nella pratica non sempre dopo un aggiornamento dei parametri si osserva una riduzione dell'errore). Il metodo che sfrutta questo approccio si chiama discesa del gradiente nel quale i pesi dei nodi che compongono la rete, che inizialmente vengono inizializzati randomicamente (con valori tipicamente piccoli), vengono aggiornati utilizzando il gradiente durante varie iterazioni chiamate epoche. L'aggiornamento dei pesi viene quindi effettuato sottraendo ai pesi correnti il gradiente moltiplicato per un iperparametro η chiamato learning rate che ha il ruolo di stabilire di quanto spostarsi lungo la direzione individuata dal gradiente

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla E(\boldsymbol{w})$$

Questo approccio (praticabile anche nel caso dei modelli lineari) non garantisce però né che il valore dei parametri converga, né di individuare il minimo globale, in quanto la funzione di loss potrebbe presentare vari minimi locali dai quali, una volta raggiunti, utilizzando la discesa del gradiente non è possibile uscire (in quanto il gradiente in un punto di minimo locale è uguale a 0 e quindi nessun aggiornamento dei pesi viene effettuato). Nonostante ciò la discesa del gradiente si è rivelato comunque un ottimo metodo per il

training di molti modelli grazie alla sua versatilità e anche perché spesso un minimo locale è più che sufficiente per ottenenre dei buoni risultati (spesso raggiungere il minimo globale è controproducente perché è molto probabile che si verifichi una situazione overfitting). La discesa del gradiente può essere utilizzata anche nel caso delle ANN (tutte le operazioni che vengono svolte per calcolare l'output della rete sono differenziabili quindi è possibile calcolare il gradiente della funzione di loss) nelle quali viene implementata mediante un algoritmo noto come backpropagation algorithm che è in grado di calcolare tutte le derivate parziali della funzione di loss rispetto ai pesi di tutti i nodi in maniera efficiente partendo dall'ultimo strato e procedendo all'indietro.

2.10 Discesa del gradiente e overfitting

L'algoritmo per la discesa del gradiente deve essere iterato per un certo numero di epoche, ma conoscere a priori questo numero non è possibile. È però fondamentale che il numero di epoche non sia né troppo basso né troppo alto. Infatti un numero di epoche troppo basso non darebbe il tempo al modello di allenarsi sufficientemente e quindi il risultato sarebbe un modello in underfitting, mentre un numero di epoche troppo alto porterebbe al problema opposto, ossia quello dell'overfitting, con il modello che, per via dell'elevato numero di epoche, si è adattato troppo ai dati di training e non è quindi in grado di generalizzare. Per risolvere questo problema può essere utilizzata una tecnica di regolarizzazione chiamata arresto anticipato, o early stopping in inglese, che consiste nel valutare durante le epoche le prestazioni del modello anche sul validation set (senza che questo venga utilizzato direttamente nel calcolo del gradiente) e interrompendo il training quando le performance sul validation set incominciano a peggiorare. Ovviamente delle fluttuazioni nelle prestazioni (soprattutto sul validation set) durante le epoche sono normali quindi è necessario stabilire un altro iperparametro, chiamato pazienza, che indica quante epoche attendere prima di interrompere il training quando le performance sul validation set non migliorano.

Come si può notare in Figura 2.6 utilizzando l'early stopping possiamo facilmente individuare quando il modello inizia ad essere in overfitting ed arrestare il training per

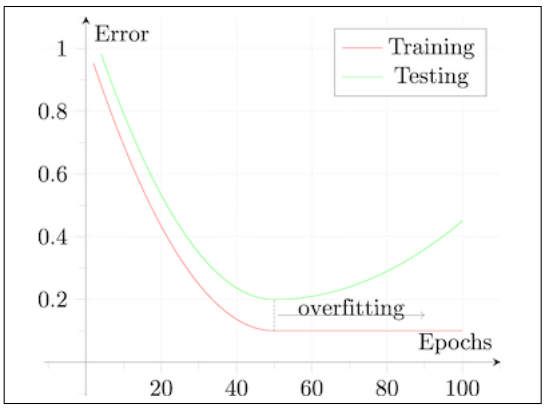

Figura 2.6: Valore della funzione di loss sul training set e sul validation set all'aumentare del numero di epoche

mantenere la capacita di generalizzazione del modello.

2.11 Reti neurali ricorrenti

Quando si ha a che fare con sequenze di dati (come nel caso dell'ECG) l'utilizzo di reti neurali feedforward spesso non porta a buoni risultati, infatti quest'ultime non sono in grado di gestire sequenze di lunghezza variabile e soprattutto la loro struttura non permette di sfruttare efficacemente l'ordinamento delle sequenze temporali che invece è spesso una componente fondamentale del problema [6] [2]. Per questo tipo di problemi uno degli approcci più comuni è utilizzare le reti neurali ricorrenti che, a differenza delle reti neurali feedforward permettono di creare dei cicli all'interno della struttura della rete e posseggono uno stato che muta via via che la sequenza viene processata. Anche per il training delle reti neurali ricorrenti è possibile utilizzare la discesa del gradiente (infatti tutte le operazioni che vengono eseguite per il calcolo dell'output della rete rimangono differenziabili), ma risulta un approccio molto dispendioso e complicato (esistono varianti del backpropagation algorithm che possono essere applicate anche alle reti neurali ricorrenti) che però non sempre producono buoni risultati [9].

3. Reservoir Computing

Come introdotto in precedenza nella Sezione 2.11, usare la discesa del gradiente per le reti neurali ricorrenti non è semplice. L'output dell'intera rete è frutto di numerose iterazioni (e di conseguenza di numerose trasformazioni non lineari) e quindi il calcolo del gradiente si complica molto. Inoltre, anche in assenza di training sulle connessioni interne, le architetture neurali ricorrenti mostrano capacità di calcolo utili in molti contesti applicativi [8]. Per questo motivo un approccio utilizzato quando si ha a che fare con reti neurali ricorrenti è quello del reservoir computing [10][21]. Nel reservoir computing si distinguono due parti: il reservoir e il readout. Il reservoir è una rete neurale ricorrente i cui pesi, una volta inizializzati, non vengono più modificati, mentre il readout è un modello semplice da allenare (tipicamente un modello lineare) che riceve in ingresso l'output del reservoir ed è l'unica parte di cui viene effettuato il training, rendendo quindi molto meno dispendioso allenare il modello [20].

3.1 Echo State Network

All'interno del paradigma del reservoir computing le Echo State Network (ESN) [8][7] rappresentano il modello più utilizzato. Le ESN possono essere utilizzate sia per compiti di regressione che di classificazione, Poiché il problema affrontato in questa tesi è un problema di classificazione verrà introdotto il modello nell'ambito di un problema di classificazione su serie temporali, in cui ogni esempio consiste in una sequenza multivariata (una dimensione per ogni derivazione dell'ECG), a cui è associata un'etichetta di classe.

Il reservoir del modello costituisce lo strato ricorrente della rete, che dato lo stato del reservoir al tempo $t-1$ e il valore dell'input al tempo t ne calcola una combinazione lineare e vi applica una funzione di attivazione (non lineare). Lo stato così ottenuto viene interpolato con lo stato del reservoir al tempo $t-1$ per ottenere il nuovo stato del reservoir.

Formalmente, sia n il numero di neuroni nel reservoir, m la dimensione dell'input ad ogni istante (nel caso degli ECG il numero di derivazioni), $\boldsymbol{x}_t$ lo stato del reservoir al tempo

t e $\boldsymbol{u}_t$ il valore dell'input al tempo t. Avremo una matrice $\boldsymbol{W} \in \mathbb{R}^{n \times m}$ in cui w_{ij} è il peso che il nodo i attribuisce alla j-esima dimensione dell'input, una matrice $\hat{\boldsymbol{W}} \in \mathbb{R}^{n \times n}$ in cui $\hat{w}_{ij}$ è il peso che il nodo i attribuisce alla j-esima dimensione dello stato precedente del reservoir (può essere visto come l'importanza del collegamento ricorrente tra il nodo i e il nodo j) e un vettore $\boldsymbol{b} \in \mathbb{R}^n$ con b_i il bias dell'i-esimo neurone del reservoir [9]. La formula per calcolare lo stato del reservoir nell'istante t è:

$$\boldsymbol{x}_t = (1-\alpha)\boldsymbol{x}_{t-1} + \alpha f(\boldsymbol{W}\boldsymbol{u}_t + \hat{\boldsymbol{W}}\boldsymbol{x}_{t-1} + \boldsymbol{b})$$

dove $\alpha \in [0,1]$ è un iperparametro chiamato leaking rate che determina quanto in fretta il modello segue il segnale di input. Un leaking rate vicino a 1 darà un peso maggiore all'aggiornamento dello stato dipendente dal nuovo input, mentre un valore vicino a 0 darà un peso maggiore allo stato precedente del reservoir e meno al contributo proveniente dal nuovo input (rallentando di fatto le dinamiche del reservoir rispetto a quelle del segnale di input).

Una volta calcolati gli stati del reservoir per ogni istante di tempo t ne viene fatta la media e il risultato viene dato come input al readout, un modello lineare che, dopo essere stato allenato, avrà il compito di classificare correttamente il vettore ricevuto. Nella Figura 3.1 è rappresentata in maniera semplificata la struttura di un modello ESN.

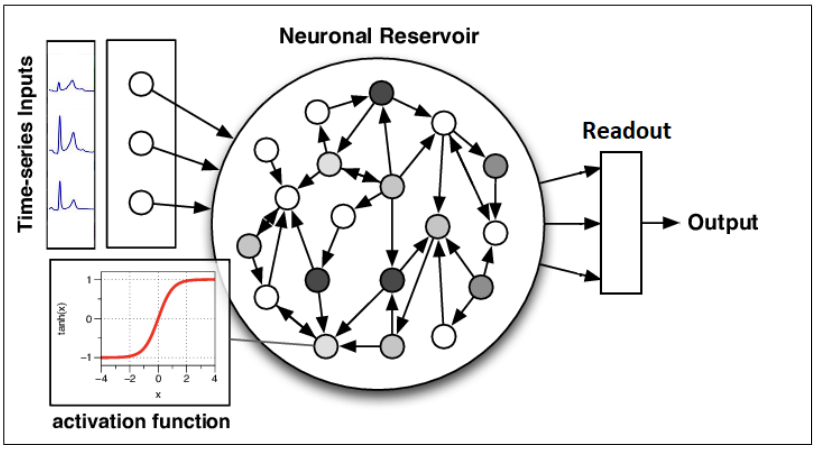

Figura 3.1: Rappresentazione schematica del modello ESN

Le due matrici $\boldsymbol{W}$ e $\hat{\boldsymbol{W}}$ sono inizializzate casualmente, ma affinché il modello abbia delle buone prestazioni è necessario che sia verificata una proprietà chiamata Echo state

property [7] [24] che garantisce che lo stato del reservoir nell'istante n sia indipendente dallo stato iniziale (che tipicamente viene posto a 0) e che le dinamiche del reservoir siano stabili quando n tende all'infinito. Una condizione necessaria affinchè questa proprietà sia soddisfatta è che il raggio spettrale (cioè il modulo dell'autovalore di modulo massimo) della matrice $\hat{\boldsymbol{W}}$ sia minore di 1 [7]. Il raggio spettrale della matrice $\hat{\boldsymbol{W}}$ deve essere opportunamente scelto e costituisce quindi un iperparametro del modello e viene indicato con ρ. Per ottenere una matrice con valori casuali, ma con un particolare raggio spettrale $\hat{\rho}$, è sufficiente inizializzare una matrice $\tilde{\boldsymbol{W}}$ con valori casuali e, sia $\tilde{\rho}$ il raggio spettrale di $\tilde{\boldsymbol{W}}$, calcolare:

$$\hat{\boldsymbol{W}} = \frac{\hat{\rho}}{\tilde{\rho}}\tilde{\boldsymbol{W}}$$

infatti:

$$\rho(\hat{\boldsymbol{W}}) = \rho(\frac{\hat{\rho}}{\tilde{\rho}}\tilde{\boldsymbol{W}}) = \frac{\hat{\rho}}{\tilde{\rho}}\rho(\tilde{\boldsymbol{W}}) = \hat{\rho}$$

Questo procedimento però richiede il calcolo del raggio spettrale della matrice $\tilde{\boldsymbol{W}}$ che per matrici grandi è un procedimento lento. Per ovviare a questo problema è possibile approssimare il raggio spettrale [4]; infatti in una matrice $\hat{\boldsymbol{W}} \in \mathbb{R}^{n\times n}$ i cui valori sono copie indipendenti e identicamente distribuite di una variabile aleatoria con distribuzione uniforme in $\left[-\frac{\hat{\rho}}{\sqrt{n}}\frac{6}{\sqrt{12}}, +\frac{\hat{\rho}}{\sqrt{n}}\frac{6}{\sqrt{12}}\right]$ vale:

$$\rho(\hat{\boldsymbol{W}}) \to \hat{\rho} \text{ per } n \to \infty$$

Dato che nei modelli ESN si ha spesso a che fare con reservoir di grandi dimensioni, è possibile sfruttare questo risultato per inizializzare velocemente i valori della matrice ricorrente in modo che abbia il raggio spettrale desiderato senza che il valore effettivo del raggio spettrale si allontani significativamente da quello desiderato.

Altri due iperparametri fondamentali sono input scaling e bias scaling che hanno il compito di determinare gli estremi della distribuzione uniforme con cui sono inizializzati rispettivamente la matrice $\boldsymbol{W}$, che viene moltiplicata per l'input, e il vettore $\boldsymbol{b}$ che rappresenta il bias dei neuroni del reservoir.

3.2 Input Routed Echo State Network

Nel modello ESN le diverse dimensioni dell'input vengono "mescolate" all'interno del reservoir, dove tutti i nodi sono collegati fra di loro. Questo, in una situazione in cui le varie dimensioni dell'input sono indipendenti o non sono tutte rilevanti per la classificazione (come nel caso dell'ECG con 12 derivazioni), non è auspicabile in quanto il modello non è più in grado di distinguere le dimensioni rilevanti da quelle che contengono poca informazione (che di fatto agiscono come da rumore, quindi potenzialmente rischiando di abbassare le performance del modello) e di apprendere a riconoscere eventuali pattern che si presentano solo in alcune delle dimensioni dell'input. Per questi motivi ha senso considerare il modello Input Routed Echo State Network (IRESN) che ha come principio quello di utilizzare dei sub-reservoir separati per ogni dimensione dell'input, in modo che il readout, mediante la fase di training, apprenda a quali dimensioni (e quindi a quali componenti del segnale di input) dare più importanza e quali invece non sono utili per la classificazione della sequenza [1]. La formula per il calcolo degli stati del reservoir in un modello con m sub-reservoir diventa quindi:

$$\forall i = 1, \ldots, m$$

$$\boldsymbol{x}_t^{(i)} = (1-\alpha)\boldsymbol{x}_{t-1}^{(i)} + \alpha f(\boldsymbol{w}^{(i)} u_t^{(i)} + \hat{\boldsymbol{W}}^{(i)} \boldsymbol{x}_{t-1}^{(i)} + \boldsymbol{b}^{(i)})$$

Ogni sub-reservoir i avrà quindi i suoi stati $\boldsymbol{x}_t^{(i)}$, la sua matrice $\hat{\boldsymbol{W}}^{(i)}$, e il vettore $\boldsymbol{w}^{(i)}$ che rappresenta il peso che ogni nodo del sub-reservoir attribuisce al valore di input $u_t^{(i)}$ ($\boldsymbol{w}^{(i)}$ è un vettore e non una matrice perchè l'input $u_t^{(i)}$ è uno scalare e non un vettore). Il readout rimane lo stesso e, ricevendo gli stati di tutti i sub-reservoir, avrà sempre il compito di restituire l'output corretto. Il comportamento del modello è rappresentato nella Figura 3.2.

Il modello IRESN può essere visto come un modello ESN con qualche modifica alla definizione di $\boldsymbol{W}$ e $\hat{\boldsymbol{W}}$. Infatti, se i nodi nei reservoir devono essere suddivisi in m sub-reservoir (un sub-reservoir per ogni dimensione dell'input) indipendenti tra loro, la matrice $\hat{\boldsymbol{W}}$ deve essere definita in modo che ogni nodo i sia collegato solo ai nodi appartenenti allo stesso sub-reservoir a cui appartiene anche lui. Per un reservoir con n nodi e m sub-

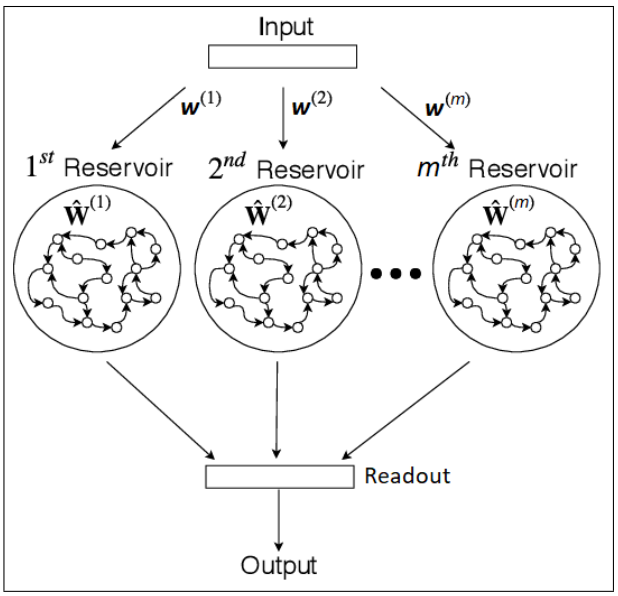

Figura 3.2: Rappresentazione schematica del modello IRESN

reservoir, la matrice che si ottiene è una matrice a blocchi della forma

$$\hat{\boldsymbol{W}} = \left(\begin{array}{c|c|c|c} \hat{\boldsymbol{W}}^{(1)} & \mathbf{0} & \cdots & \mathbf{0} \\ \hline \mathbf{0} & \hat{\boldsymbol{W}}^{(2)} & \ddots & \vdots \\ \hline \vdots & \ddots & \ddots & \mathbf{0} \\ \hline \mathbf{0} & \cdots & \mathbf{0} & \hat{\boldsymbol{W}}^{(m)} \end{array}\right)$$

dove $\hat{\boldsymbol{W}}_i \in \mathbb{R}^{k_i \times k_i}$ e k_i è il numero di nodi presenti nell'i-esimo sub-reservoir (ovviamente $\sum_{i=1}^{m} k_i = n$). Similmente affinché l'i-esima dimensione dell'input sia utilizzata solo dai nodi dell'i-esimo sub-reservoir la matrice $\boldsymbol{W}$ sarà

$$\boldsymbol{W} = \left(\begin{array}{c|c|c|c} \boldsymbol{w}^{(1)} & \mathbf{0} & \cdots & \mathbf{0} \\ \hline \mathbf{0} & \boldsymbol{w}^{(2)} & \ddots & \vdots \\ \hline \vdots & \ddots & \ddots & \mathbf{0} \\ \hline \mathbf{0} & \cdots & \mathbf{0} & \boldsymbol{w}^{(m)} \end{array}\right)$$

dove $\boldsymbol{w}^{(i)} \in \mathbb{R}^{k_i}$ sono vettori con $\boldsymbol{w}_j^{(i)}$ che indica il peso che il j-esimo nodo dell'i-esimo sub-reservoir attribuisce all'input che riceve (per semplicità è stato considerato solo il caso in cui ad ogni sub-reservoir viene fornito come input una sola dimensione dell'input)

3.3 Interconnessioni fra sub-reservoir

Separando i sub-reservoir si ha sicuramente un vantaggio nei casi in cui i segnali che compongono le serie multivariate da classificare sono completamente indipendenti. Nel particolare della diagnosi della Sindrome di Brugada non è scontato che ciò sia vero. In casi come questo potrebbe essere utile utilizzare un modello più flessibile in cui i vari sub-reservoir sono in grado di scambiarsi informazioni [1]. Per rendere possibile questa interconnessione fra sub-reservoir la formula per il calcolo degli stati di ogni sub-reservoir diventa:

$$\forall i = 1, \dots, m$$

$$\boldsymbol{x}_t^{(i)} = (1-\alpha)\boldsymbol{x}_{t-1}^{(i)} + \alpha f(\boldsymbol{w}^{(i)} u_t^{(i)} + \hat{\boldsymbol{W}}^{(i)} \boldsymbol{x}_{t-1}^{(i)} + \boldsymbol{b}^{(i)} + \sum_{j \neq i} \boldsymbol{W}_{int}^{(ij)} \boldsymbol{x}_{t-1}^{(j)})$$

dove $\boldsymbol{W}_{int}^{(ij)} \in \mathbb{R}^{k_i \times k_j}$ è la cosiddetta matrice di interconnessione fra il sub-reservoir j e il sub-reservoir i.

Anche in questo caso si possono apportare delle modifiche alla matrice $\hat{\boldsymbol{W}}$ per rappresentare questo modello con solo le due matrici $\boldsymbol{W}$ e $\hat{\boldsymbol{W}}$. La matrice $\hat{\boldsymbol{W}}$ diventa:

$$\hat{\boldsymbol{W}} = \left(\begin{array}{c|c|c|c} \hat{\boldsymbol{W}}_1 & \boldsymbol{W}_{int}^{(12)} & \cdots & \boldsymbol{W}_{int}^{(m1)} \\ \hline \boldsymbol{W}_{int}^{(21)} & \hat{\boldsymbol{W}}_2 & \ddots & \vdots \\ \hline \vdots & \ddots & \ddots & \boldsymbol{W}_{int}^{((m-1)m)} \\ \hline \boldsymbol{W}_{int}^{(m1)} & \cdots & \boldsymbol{W}_{int}^{(m(m-1))} & \hat{\boldsymbol{W}}_m \end{array}\right)$$

Spesso, per semplicità si utilizza una sola matrice di interconnessione per ogni sub-reservoir, quindi $\boldsymbol{W}_{int}^{(i)} = \boldsymbol{W}_{int}^{(12)} = \cdots = \boldsymbol{W}_{int}^{(1m)}$. Ovviamente anche i valori contenuti in queste matrici di interconnessione andranno scalate opportunamente per un certo iperparametro chiamato inter-reservoir scaling.

È posibile utilizzare un solo iperparametro per tutte le matrici, oppure un iperparametro per ogni sub-reservoir (soprattutto se si sceglie di utilizzare una sola matrice per sub-reservoir), o anche un iperparametro per ogni matrice (caso estremo che aumenta notevolmente il numero di iperparametri).

3.4 Training degli iperparametri

L'introduzione del modello IRESN porta quindi ad un aumento del numero di iperparametri che dipende dal numero di dimensioni dell'input, in quanto ogni sub-reservoir avrà un proprio raggio spettrale, un proprio input scaling, bias scaling e, se utilizzato, anche un inter-reservoir scaling. Un aumento del numero di iperparmetri comporta un aumento delle dimensioni dello spazio degli iperparametri redendo più dispendiosa (o meno efficace a parità di tempo) la fase di model selection del modello, durante la quale vanno scelti gli iperparametri. Questo è un problema ricorrente in machine learning quando si aumenta il numero di dimensioni dell'input, che in questo caso implica un aumento esponenziale del numero di tentativi da effettuare per decidere quale sia la combinazione di iperparametri migliori. Ad esempio, supponendo di avere p iperparametri relativi ad ognuno degli m sub-reservoir, il numero totale di iperparametri sarebbe mp. Se indichiamo con b il numero di valori che vogliamo testare, il numero totale di combinazioni è b^{mp}, esponenziale nel numero di sub-reservoir e quindi di dimensioni dell'input. Nel caso specifico del modello IRESN, se supponiamo di dover scegliere solo input scaling e bias scaling e di voler testare per ognuno 3 valori, il numero totale di tentativi in un modello con 3 derivazioni sarebbe $3^6 = 729$, alto ma ancora praticabile, mentre con 12 derivazioni sarebbe 3^{24}, che decisamente non è trattabile.

La soluzione più semplice è sicuramente quella di utilizzare lo stesso iperparametro per tutti i sub-reservoir, ma questa operazione potrebbe danneggiare le performance del modello in quanto le diverse dimensioni dell'input, essendo indipendenti, potrebbero richiedere che il modello usi diversi iperparametri fra i vari sub-reservoir per classificare al meglio le sequenze.

Una soluzione analizzata in questa tesi è quella di introdurre una fase di training anche per il reservoir ma, anziché modificare i pesi contenuti in $\hat{\boldsymbol{W}}$ e $\boldsymbol{W}$, modificare i valori degli iperparametri, che a questo puto diventano in realtà dei parametri del modello.

Per modificare il valore di questi parametri interni al reservoir è necessario utilizzare la discesa del gradiente (introdotta nella Sezione 2.10), ma ottenere una formula per il calcolo del gradiente della funzione di loss rispetto a questi nuovi parametri (come ad

esempio viene fatto per i pesi delle feedforward neural network con l'algoritmo di back-propagation) è molto complicato. Per questo, per ottenere il gradiente, è stata utilizzata la differenziazione automatica (spesso chiamata autodiff) presente nei principali framework per implemetare reti neurali (come TensorFlow e PyTorch). Mediante la differenziazione automatica, via via che vengono eseguite le operazioni per calcolare l'output del modello, viene costruito un grafo che tiene traccia di tutte le operazioni effettuate, e mediante questo grafo è poi possibile ricavare il gradiente relativo ai parametri a cui si è interessati (che devono essere specificati in anticipo).

In questo modo il tempo impiegato per il training del modello aumenta (sia perché costruire questo grafo e calcolare il gradiente è un overhead importante, sia perché è necessario calcolare l'output del reservoir ad ogni epoca della discesa del gradiente), ma diminuisce quello legato alla scelta degli iperparametri. Inoltre non si pone più il problema della scelta dei valori da testare per gli iperparametri (o di un intervallo di valori da cui scegliere), ma basta semplicemente scegliere un valore iniziale per ogni iperparametro da cui partire e poi durante la fase di training il valore verrà automaticamente modificato, senza vincoli sui valori che tale iperparametro possa assumere (non sempre questo è positivo, basta pensare ad esempio al leaking rate che necessariamente deve essere compreso fra 0 e 1), affinché le performance migliorino. È comunque importante ricordare che, nonostante non sia necessario scegliere i valori degli iperparametri, è comunque necessario eseguire la validazione del modello per evitare situazioni di overfitting.

4. Metodo utilizzato negli esperimenti

All'interno di questa tesi sono stati valutati diversi modelli per classificare correttamente ECG sia con 3 che con 12 derivazioni.

I dati utilizzati sono stati raccolti presso centri clinici e medici sul territorio della Toscana ed in totale sono 306 ECG di cui 183 negativi e 123 positivi. Gli ECG positivi sono relativi sia a pazienti in cui la sindrome si è presentata in modo spontaneo sia a pazienti i cui ECG sono risultati positivi dopo la somministrazione di farmaci antiaritmici.

Le sequenze relative agli ECG sono state preprocessate ritagliandole in modo che contenessero un solo battito cardiaco e sono state traslate sull'asse y in modo che il primo valore di ogni sequenza fosse 0. Le sequenze sono state inoltre normalizzate sull'asse y per derivazione, ossia per ogni derivazione sono state divise tutte le sequenze relative a quella derivazione per il massimo del modulo dei valori presenti in tali sequenze. In questo modo ogni derivazione di ogni sequenza è compresa tra -1 e 1 senza però perdere le informazioni sull'ampiezza delle onde dell'ECG (fenomeno che sarebbe accaduto se si fosse normalizzata ogni sequenza indipendentemente dalle altre) in quanto questa informazione potrebbe essere utile per la diagnosi della sindrome. Inoltre le sequenze sono state normalizzate anche sull'asse x (ossia rispetto al tempo) in modo che tutte le sequenze avessero la stessa lunghezza (che è stata fissata a 500).

Nei modelli testati gli iperparametri utilizzati sono stati il numero di unità nel reservoir, il raggio spettrale del reservoir, l'input scaling, il bias scaling e il leaking rate. L'inter-reservoir scaling è stato testato solo nei modelli con training degli iperparametri. La funzione di attivazione utilizzata da tutte le unità del reservoir è stata la tangente iperbolica (che consente di mantenere l'output di ogni nodo fra -1 e 1). Per quanto riguarda il readout nella maggior parte dei modelli è stato usato un modello lineare che utilizzava per il training il metodo di risoluzione diretta illustrato nella Sezione 2.8. Fanno eccezione i modelli con training degli iperparametri sui quali, poiché è stato usato un algoritmo iterativo per la discesa del gradiente, è stato utilizzato come readout un singolo neurone artificiale con la funzione sigmoidea come funzione di attivazione per ottenere output fra

0 e 1. Gli unici iperparametri necessari per il readout sono stati il coefficiente di regolarizzazione per il readout con risoluzione diretta (per controllare la regolarizzazione) e il learning rate quando è stata usata la discesa del gradiente. Per regolarizzare il modello nella discesa del gradiente è stato utilizzato l'arresto anticipato, con una pazienza di 100 epoche, per terminare anticipatamente il training del modello, recuperando i parametri con cui si erano ottenute i migliori risultati sul validation set (quindi i parametri presenti 100 epoche prima dell'arresto). Il numero massimo di epoche è stato fissato a 1000, un valore molto alto, in modo che la terminazione del training avvenisse quando il valore della loss sul validation set iniziava ad aumentare.

Per la selezione degli iperparametri (nei modelli senza training degli iperparametri) è stata utilizzata una grid search ed in particolare i valori testati sono stati:

- input scaling: 0.5, 1.0, 1.5
- bias scaling: 0.1, 0.5, 1.0
- leaking rate: 0.1, 0.5, 1.0
- coefficiente di regolarizzazione: 1.0, 0.1, 0.01, 0.001, 0.0001

Il numero di unità del resevoir è variato a seconda dei modelli testati. Nei modelli con sub-reservoir sono stati usati numeri di unità che fossero divisibili per il numero di sub-reservoir, ma in generale per ogni modello sono stati esplorati 4 valori compresi fra le 500 e le 2000 unità. Il valore per raggio spettrale è stato fissato a 0.999, in quanto evidenze empiriche suggeriscono che un valore vicino al limite teorico per il soddisfacimento dell'Echo State Property sia una buona scelta per ottenere delle buone prestazioni.

Nei modelli con training degli iperparametri, gli iperparametri sottoposti a training sono stati input scaling, bias scaling e, quando presente, inter-reservoir scaling. Non tutti gli iperparametri sono stati sottoposti a training in quanto ottenere la convergenza della discesa del gradiente allenando tutti gli iperparametri non è semplice e complica ulteriormente (anche in termini di tempo necessario) la fase di training. In particolare il leaking rate, per via della sua funzione cruciale nel determinare le dinamiche del modello e della necessità di limitare il suo valore fra 0 e 1, è stato escluso dal training fissando il suo valore a 0.1

(anche sulla base dei risultati della validation dei modelli senza training degli iperparametri). Anche il raggio spettrale è stato escluso, utilizzando per tutti i sub-reservoir il valore di 0.999, per mantenere soddisfatta l'Echo State Property. Il numero di unità, che ovviamente non poteva essere sottoposto a training, è stato fissato a 1200 (anche in questo caso i risultati della validation nei modelli senza training degli iperparametri suggerivano che non fosse necessario un numero particolarmente elevato di unità nel reservoir, neanche quando si usavano tutte e 12 le derivazioni). Il learning rate utilizzato per la discesa del gradiente è stato 0.01 per i modelli senza interconnessioni fra sub-reservoir, mentre è stato di 0.005 per i modelli con interconnessioni. Questi valori sono stati individuati effettuando vari tentativi con valori differenti e osservando la regolarità delle curve di training del modello (ad esempio una curva troppo irregolare indica la necessità di un learning rate più basso).

Per eseguire la validation e la model assesment è stata utilizzata la double k-fold validation con 5 fold esterni e 4 interni, in modo che il validation set e il test set avessero la stessa dimensione (pari a un quinto dei dati utilizzati) ed in modo da sfruttare al meglio tutti i dati del dataset, che consiste di soli 306 ECG. All'interno di ogni fold esterna sono state fatte 3 reinizializzazioni (tranne che per i modelli con training degli iperparametri che richiedevano un tempo notevolmente maggiore per il training rispetto agli altri modelli), dei cui risultati è stata poi fatta la media, per evitare che situazioni particolarmente fortunate o sfortunate andassero a influenzare il risultato finale.

Ogni modello è stato testato sia sul caso con 3 derivazioni che sul caso con 12 per osservare come variano le performance dei modelli all'aumentare delle dimensioni dell'input.

5. Risultati

In questo capitolo sono esposti i risultati degli esperimenti effettuati. In particolare nella Sezione 5.1 sono presentati i risultati ottenuti col modello ESN che costituiscono la baseline con la quale confrontare gli altri modelli. Nella Sezione 5.2 sono riportati i risultati del modello a comitato. Nella Sezione 5.3 sono presentati i risultati ottenuti da vari esperimenti effettuati col modello IRESN nei quali si vanno anche a testare altri metodi di regolarizzazione nel readout per discriminare meglio fra derivazioni rilevanti e irrilevanti. Nella sezione 5.4 sono presentati i risultati ottenuti col modello IRESN utilizzando il training degli iperparametri. Nella Sezione 5.5 sono riportati i risultati degli esperimenti col modello IRESN con interconnessioni fra sub-reservoir. Infine nella Sezione 5.6 sono presentati i risultati di esperimenti preliminari sul modello IRESN utilizzando sub-reservoir di dimensione variabile.

5.1 ESN

Il primo modello testato è stato il modello di reservoir computing che era già stato utilizzato per la diagnosi della Sindrome di Brugada all'interno del progetto BrAID, ossia il modello ESN su 3 derivazioni. I risultati di questo modello, riportati nella Tabella 5.1, saranno utilizzati come baseline per valutare le performance dei prossimi modelli che verranno testati.

In questo modello, come in tutti i modelli testati, si osserva una sensibilità (il rapporto di veri positivi su veri positivi e falsi negativi) del modello inferiore alla specificità (il rapporto di veri negativi su veri negativi e falsi positivi). Ciò è probabilmente dovuto alla presenza nel dataset di più ECG negativi che positivi.

training accuracy	validation accuracy	test accuracy	sensitivity	specificity
$95.5\% \pm 0.57$	$87.7\% \pm 0.89$	$85.7\% \pm 3.72$	$79.9\% \pm 7.09$	$89.4\% \pm 3.81$

Tabella 5.1: Risultati del modello ESN con 3 derivazioni

Il modello ESN è stato poi testato con 12 derivazioni, e come atteso, i risultati sono peggiorati. Come si può vedere nella Tabella 5.2 non solo la precisione sul validation e test set è scesa, ma è anche aumentata la precisone sul training set. L'introduzione di più derivazioni fa si che il modello abbia più difficoltà a generalizzare (anche se sul readout è comunque stata fatta la regolarizzazione), fatto probabilmente dovuto all'aumento del "rumore", causato dalle derivazioni non rilevanti per la diagnosi.

training accuracy	validation accuracy	test accuracy	sensitivity	specificity
$98.6\% \pm 0.45$	$84.8\% \pm 1.30$	$83.2\% \pm 3.38$	$77.7\% \pm 4.41$	$87.1\% \pm 3.05$

Tabella 5.2: Risultati del modello ESN con 12 derivazioni

5.2 Comitato ESN

Un modo immediato per separare le dimensioni dell'input è quello di utilizzare un modello ESN per ogni dimensione in modo da formare un comitato in cui l'output finale viene deciso da un voto a maggioranza (dove ogni modello nel comitato ha uguale peso). Tipicamente nel machine learning i modelli a comitato sono utilizzati per combinare modelli diversi (o reinizializzazioni diverse dello stesso modello) che prendono in input gli stessi dati. In questo caso invece l'input per ogni modello del comitato sarà diverso, poichè l'obbiettivo del comitato è separare le dimensioni dell'input. Ogni modello che fa parte del comitato verrà quindi allenato indipendentemente dagli altri ed avrà anche la propria validazione (effettuata con la k-fold cross validation). Per evitare situazioni di parità (che potevano verificarsi nel comitato con 12 derivazioni) anzichè utilizzare, per ogni modello, un voto binario è stato utilizzato come voto il risultato del classificatore lineare senza applicare la funzione di soglia alla fine. In questo modo ogni modello del comitato fornisce come voto un numero reale. Alla media di questi voti è stata poi applicata la funzione di soglia per determinare l'output del comitato.

Nella Tabelle 5.3 e 5.5, nelle quali sono riportati i risultati del comitato con 3 e 12 derivazioni, non è presente la validation accuracy in quanto i singoli modelli che compongono il comitato sono stati riallenati, dopo la k-fold cross validation, su training set e

validation set per sfruttare tutti i dati presenti nel dataset. Valutare quindi le performance del comitato sul validation set sarebbe stato sbagliato, in quanto i modelli che compongono il comitato sono stati allenati direttamente anche sui dati del validation set (la training accuracy è infatti stata misurata su training e validation set).

I risultati del comitato con 3 derivazioni, Tabella 5.3, sono molto buoni e migliorano quelli ottenuti col modello ESN. Questo indica che separare le derivazioni porta un significativo vantaggio nella classificazione degli ECG. Inoltre analizzando le performance dei singoli modelli che compongono il comitato è possibile notare, nella Tabella 5.4, come i modelli che ricevono in input la derivazione V2 o V1 abbiano delle performance notevolmente migliori rispetto a quelle dei modelli che ricevono V3.

training accuracy	test accuracy	sensitivity	specificity
$95.2\% \pm 0.50$	$89.2\% \pm 3.61$	$80.9\% \pm 5.91$	$95.0\% \pm 4.11$

Tabella 5.3: Risultati del comitato di ESN con 3 derivazioni

derivazione	training accuracy	validation accuracy	test accuracy	sensitivity	specificity
V1	$90.9\% \pm 1.77$	$87.1\% \pm 1.82$	$85.2\% \pm 5.60$	$76.9\% \pm 7.99$	$90.4\% \pm 6.36$
V2	$93.2\% \pm 1.25$	$89.4\% \pm 1.53$	$89.0\% \pm 3.10$	$83.1\% \pm 5.02$	$93.3\% \pm 5.34$
V3	$82.0\% \pm 2.96$	$75.6\% \pm 1.04$	$72.0\% \pm 4.67$	$55.5\% \pm 8.49$	$83.2\% \pm 6.55$

Tabella 5.4: Risultati dei singoli modelli ESN che compongono il comitato con 3 derivazioni

Quando si utilizza il comitato con 12 derivazioni i risultati, Tabella 5.5, sono analoghi a quelli del modello ESN, senza quindi ottenere alcun miglioramento. Questo fatto non sorprende, infatti considerando che molte delle 12 derivazioni sono poco rilevanti il voto finale dipenderà per la maggior parte da modelli con una bassa accuracy. Questo fatto può essere chiaramente osservato nella Tabella 5.6, dove si osserva che i modelli che ottengono dei buoni risultati sono solo quelli che ricevono V1 o V2. Si può osservare anche che ci sono delle derivazioni che, basandosi sui risultati, sembrano più rilevanti di V3 come ad esempio aVF o V5.

training accuracy	test accuracy	sensitivity	specificity
$93.2\% \pm 0.76$	$82.9\% \pm 4.54$	$65.2\% \pm 7.50$	$94.8\% \pm 3.93$

Tabella 5.5: Risultati del comitato di ESN con 12 derivazioni

derivazione	training accuracy	validation accuracy	test accuracy	sensitivity	specificity
aVF	$86.2\% \pm 3.83$	$75.8\% \pm 0.71$	$74.8\% \pm 4.05$	$60.7\% \pm 8.34$	$84.3\% \pm 2.93$
aVL	$81.5\% \pm 3.22$	$72.2\% \pm 1.93$	$68.0\% \pm 3.59$	$46.2\% \pm 5.29$	$82.8\% \pm 4.51$
aVR	$84.0\% \pm 2.40$	$77.0\% \pm 1.23$	$74.7\% \pm 3.34$	$54.0\% \pm 8.21$	$89.1\% \pm 5.22$
I	$82.0\% \pm 2.93$	$72.7\% \pm 1.97$	$66.1\% \pm 6.65$	$49.6\% \pm 8.18$	$77.5\% \pm 9.80$
II	$84.8\% \pm 4.42$	$74.9\% \pm 2.01$	$74.7\% \pm 2.76$	$56.1\% \pm 8.94$	$87.8\% \pm 4.90$
III	$83.0\% \pm 4.19$	$73.6\% \pm 1.77$	$70.7\% \pm 3.53$	$55.6\% \pm 6.54$	$81.7\% \pm 7.55$
V1	$92.6\% \pm 1.28$	$87.4\% \pm 1.03$	$85.6\% \pm 3.75$	$78.4\% \pm 6.37$	$90.7\% \pm 4.30$
V2	$93.4\% \pm 1.56$	$89.2\% \pm 1.16$	$87.3\% \pm 5.22$	$78.5\% \pm 10.30$	$93.9\% \pm 5.03$
V3	$84.5\% \pm 3.20$	$74.6\% \pm 1.27$	$72.2\% \pm 4.32$	$55.6\% \pm 6.96$	$83.5\% \pm 5.56$
V4	$81.6\% \pm 4.06$	$75.1\% \pm 1.94$	$71.9\% \pm 7.74$	$55.3\% \pm 11.54$	$83.7\% \pm 7.49$
V5	$82.7\% \pm 4.18$	$74.2\% \pm 1.86$	$72.2\% \pm 4.86$	$57.4\% \pm 10.87$	$83.3\% \pm 7.17$
V6	$79.6\% \pm 2.29$	$71.2\% \pm 1.76$	$69.1\% \pm 4.24$	$47.9\% \pm 8.00$	$83.4\% \pm 4.57$

Tabella 5.6: Risultati dei singoli modelli ESN che compongono il comitato con 12 derivazioni

5.3 IRESN

Il modello IRESN è stato introdotto per far in modo che il readout, una volta separate le varie derivazioni, fosse in grado di apprendere dal dataset quali fossero quelle più rilevanti. Con 3 derivazioni, dai risultati riportati nella Tabella 5.7, si osserva come si ottenga un aumento di performance sostanziale rispetto al modello ESN (i risultati sono paragonabili a quelli del modello a comitato), segno del fatto che separare le derivazioni, anzichè unirle in un unico reservoir, aiuti il readout a classificare meglio le sequenze. Inoltre proprio perchè le derivazioni sono separate è possibile, osservando i pesi che il readout assegna agli stati ottenuti dai vari sub-reservoir, stabilire a quali derivazioni il readout, dopo il training, assegna una maggior rilevanza. I risultati di questa analisi sono riportati nella

Tabella 5.8 dove si osserva come V3 sia reputato dal modello meno rilevante rispetto a V1 e V2.

training accuracy	validation accuracy	test accuracy	sensitivity	specificity
$95.7\% \pm 1.04$	$90.9\% \pm 0.78$	$90.1\% \pm 4.88$	$83.6\% \pm 5.66$	$94.6\% \pm 4.20$

Tabella 5.7: Risultati del modello IRESN con 3 derivazioni

derivazione	peso readout
V1	$38.8\% \pm 2.50$
V2	$39.9\% \pm 2.96$
V3	$21.3\% \pm 1.82$

Tabella 5.8: Peso (in percentuale rispetto al totale) attribuito dal readout del modello IRESN alle 3 derivazioni

L'uso del modello IRESN con 12 derivazioni invece non porta, a differenza di quanto ipotizzato, a un miglioramento delle performance rispetto al modello ESN con 12 derivazioni, come evidenziano i risultati nella Tabella 5.9, che su validation e test set pareggiano quelli ottenuti col modello ESN. È comunque interessante osservare i pesi che il readout del modello ha assegnato alle varie derivazioni, per capire se il modello è stato in grado di apprendere quali, fra le 12 derivazioni, siano le più rilevanti. I risultati nella Tabella 5.10 mostrano come le derivazioni ritenute più importanti siano V1 e V2 (soprattutto quest'ultima), ma si osserva anche che il peso associato alle derivazioni non importanti non è comunque trascurabile. La somma dei pesi delle derivazioni con peso minore del 10% è del 67.8%, più della somma dei pesi di V1 e V2.

training accuracy	validation accuracy	test accuracy	sensitivity	specificity
$96.8\% \pm 1.78$	$84.5\% \pm 1.00$	$82.1\% \pm 2.81$	$72.2\% \pm 7.59$	$89.2\% \pm 1.94$

Tabella 5.9: Risultati del modello IRESN con 12 derivazioni

Per cercare di risolvere questo problema è stata testata sul readout la lasso regression, per far si che i pesi relativi alle derivazioni meno rilevanti fossero azzerati. I risultati sulla

derivazione	peso readout
aVF	$8.8\% \pm 4.00$
aVL	$4.7\% \pm 2.28$
aVR	$6.1\% \pm 1.92$
I	$4.8\% \pm 2.48$
II	$9.0\% \pm 3.76$
II	$5.6\% \pm 2.34$
V1	$10.3\% \pm 6.81$
V2	$21.9\% \pm 8.98$
V3	$7.9\% \pm 2.84$
V4	$7.1\% \pm 3.45$
V5	$7.3\% \pm 2.39$
V6	$6.4\% \pm 2.82$

Tabella 5.10: Peso (in percentuale rispetto al totale) attribuito dal readout del modello IRESN alle 12 derivazioni

precisione utililzzando la lasso regression, Tabella 5.11, sono analoghi a quelli ottenuti con la ridge regression, anche se si osserva un'abbassamento dell'accuracy sul training set (a fronte di un'equivalente accuracy sul test set), mentre per quanto riguarda la situazione dei pesi attirbuiti dal readout, Tabella 5.12, l'utilizzo della lasso regression ha avuto successo nell'abbassare il peso delle derivazioni meno rilevanti. È infatti aumentato il peso attribuito a V1, V2 e aVF mentre tutti gli altri sono scesi (fatta eccezione per II che è rimasto lo stesso) e la somma dei pesi inferiori al 10% è scesa al 39.6%.

training accuracy	validation accuracy	test accuracy	sensitivity	specificity
$95.6\% \pm 0.61$	$84.0\% \pm 1.53$	$81.9\% \pm 5.16$	$72.2\% \pm 6.75$	$88.5\% \pm 7.22$

Tabella 5.11: Risultati del modello IRESN con 12 derivazioni utilizzando la lasso regression

derivazione	peso readout
aVF	$12.3\% \pm 14.54$
aVL	$2.8\% \pm 3.52$
aVR	$3.7\% \pm 3.32$
I	$2.2\% \pm 2.25$
II	$9.1\% \pm 10.17$
II	$3.1\% \pm 3.94$
V1	$14.6\% \pm 11.06$
V2	$33.5\% \pm 21.59$
V3	$3.6\% \pm 2.96$
V4	$5.3\% \pm 5.80$
V5	$4.8\% \pm 6.66$
V6	$5.1\% \pm 6.28$

Tabella 5.12: Peso (in percentuale rispetto al totale) attribuito dal readout regolarizzato con la lasso regression alle 12 derivazioni

Alla luce della scarsa rilevanza di V3 e della rilevanza di aVF, è stato testato un modello IRESN con 3 derivazioni, ma sostituendo V3 con aVF, per vedere se era possibile superare le performance del modello con V1, V2 e V3 (Tabella 5.7). I risultati riportati nella Tabella 5.13 essenzialmente coincidono con quelli ottenuti precedentemente, confermando che V3 può essere sostituito con aVF senza che le performance si abbassino, ma senza neanche ottenere miglioramenti.

training accuracy	validation accuracy	test accuracy	sensitivity	specificity
$95.9\% \pm 0.96$	$91.1\% \pm 0.76$	$89.7\% \pm 3.65$	$84.8\% \pm 6.67$	$93.0\% \pm 4.22$

Tabella 5.13: Risultati del modello IRESN con V1, V2 e aVF come input

Anche limitarsi a considerare un modello con solo V1 e V2 (dato che chiaramente vengono indicate come le derivazioni più importanti) non porta a un miglioramento, come si può vedere dalla Tabella 5.15, ma anzi si ottiene un leggero peggioramento.

derivazione	peso readout
aVF	$17.0\% \pm 5.19$
V1	$40.1\% \pm 6.53$
V2	$42.9\% \pm 6.63$

Tabella 5.14: Peso (in percentuale rispetto al totale) attribuito dal readout alle 3 derivazioni utilizzate (V1, V2, aVF)

training accuracy	validation accuracy	test accuracy	sensitivity	specificity
$95.9\% \pm 1.24$	$91.8\% \pm 1.73$	$89.0\% \pm 5.94$	$85.1\% \pm 10.30$	$92.4\% \pm 2.05$

Tabella 5.15: Risultati del modello IRESN con solo V1 e V2 come input

derivazione	peso readout
V1	$47.0\% \pm 9.47$
V2	$53.0\% \pm 9.47$

Tabella 5.16: Peso (in percentuale rispetto al totale) attribuito dal readout alle 2 derivazioni utilizzate (V1 e V2)

5.4 IRESN con training degli iperparametri

Nei modelli IRESN testati fino ad ora il valore degli iperparametri era lo stesso per tutti i sub-reservoir. Questo, soprattutto nel modello con 12 derivazioni, potrebbe impedire di ottenere dei buoni risultati. Come discusso in precedenza l'approccio utilizzato in questa tesi per gestire l'aumento esponenziale del numero di iperparametri all'aumentare del numero di sub-reservoir è quello di introdurre una fase di training per il reservoir dove il valore di alcuni iperparametri viene modificato usando la discesa del gradiente.

Per questo modello, come detto in precedenza, durante la fase di training degli iperparametri è stato utilizzato un readout allenato in modo iterativo (assieme agli iperparametri del reservoir). I risultati di questo readout erano però inferiori a quelli ottenuti con un readout allenato col metodo diretto, quindi, per massimizzare le performance, il training del

modello è stato suddiviso in due fasi. Nella prima fase mediante la discesa del gradiente venivano modificati gli iperparametri e i pesi del readout iterativo, mentre nella seconda al readout allenato in precedenza veniva sostituito un readout allenato col metodo diretto (fare il training degli iperparametri utilizzando fin da subito il readout da allenare col metodo diretto complicava notevolmente il training in quanto in ogni epoca della discesa del gradiente sarebbe stato necessario riallenare il readout). Per ogni modello con training degli iperparametri è riportato anche il grafico, con le curve dell'errore sul training set e sul validation set, relativo al training effettuato in uno dei 5 fold esterni (di ogni modello è stato selezionato un grafico che fosse rappresentativo del training nei 5 fold). In tutti i grafici sono riportate le curve d'errore fino al momento dell'arresto anticipato, ma i valori dei parametri che poi sono stati utilizzati sono quelli con cui si sono ottenuti i risultati migliori sul validation set, e non quelli presenti all'ultima epoca.

Nella Tabella 5.17 sono riportati i risultati di entrambi i tipi di readout e i risultati ottenuti col readout diretto sono analoghi a quelli ottenuti in precedenza col modello IRESN senza training degli iperparametri. Nella Tabella 5.18 si può osservare come anche i valori appresi per l'input scaling sembrano concordare con i pesi del readout nell'assegnare a V3 meno importanza rispetto a V1 e V2.

readout	training accuracy	validation accuracy	test accuracy	sensitivity	specificity
iterativo	$88.6\% \pm 2.03$	$87.1\% \pm 7.97$	$85.0\% \pm 5.03$	$75.9\% \pm 7.35$	$91.0\% \pm 6.08$
diretto	$95.9\% \pm 2.33$	$90.0\% \pm 2.11$	$90.5\% \pm 4.31$	$83.6\% \pm 7.12$	$95.2\% \pm 3.35$

Tabella 5.17: Risultati del modello IRESN con training degli iperparametri su 3 derivazioni

Con 12 derivazioni, nella Tabella 5.19, si osserva un leggero miglioramento sul test set, rispetto al modello ESN. In questo modello però sul test set si ottiene un'accuracy più alta di quella ottenuta sul validation set. Questo fatto non sorprende in quanto, nei modelli con training degli iperparametri, il risultato sul validation set non è frutto dei numerosi tentativi per la selezione degli iperparametri. Il validation set viene utilzzato solo per l'arresto anticipato durante la discesa del gradiente e per la selezione del coefficiente di regolarizzazione del readout. Inoltre dopo la validation il modello viene riallenato anche

derivazione	peso readout	input scaling
V1	$45.7\% \pm 6.77$	3.10 ± 0.8745
V2	$36.8\% \pm 6.70$	2.56 ± 0.9082
V3	$17.5\% \pm 9.15$	2.03 ± 0.5066

Tabella 5.18: Peso (in percentuale rispetto al totale) e input scaling attribuito training del modello alle 3 derivazioni

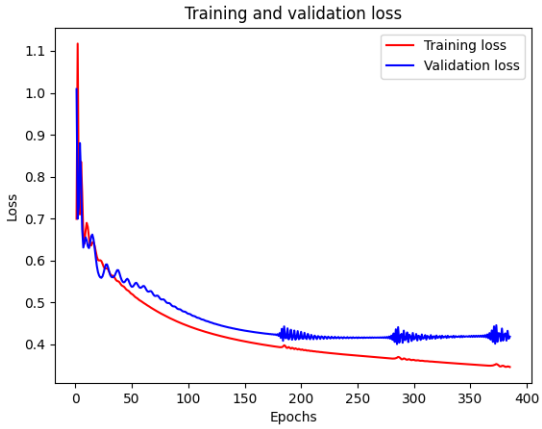

Figura 5.1: Curva della loss sul training set e validation set sul modello IRESN con 3 derivazioni

sul validation set, contribuendo ad un aumento di performance sul test set. Nonostante questo piccolo miglioramento il modello presenta ancora divario fra accuracy sul test set e sul test set, problema che interessano tutti i modelli con 12 derivazioni (fatta eccezione per il comitato).

Anche con 12 derivazioni, nella Tabella 5.20, i valori appresi per l'input scaling dei vari sub-reservoir spesso sono proporzionali al peso assegnato dal readout, anche se non è evidente, come sperato, una netta separazione fra derivazioni rilevanti e irrilevanti.

5.5 IRESN con interconnessioni

L'ultimo modello testato è stato il modello IRESN con interconnessioni fra sub-reservoir e training degli iperparametri. Per le interconnessioni è stata utilizzata una sola matrice

readout	training accuracy	validation accuracy	test accuracy	sensitivity	specificity
iterativo	$86.0\% \pm 3.50$	$78.6\% \pm 4.35$	$76.1\% \pm 2.44$	$61.0\% \pm 12.95$	$85.5\% \pm 5.65$
diretto	$98.9\% \pm 0.74$	$83.0\% \pm 1.97$	$85.0\% \pm 3.83$	$75.3\% \pm 11.37$	$91.0\% \pm 5.38$

Tabella 5.19: Risultati del modello IRESN con training degli iperparametri su 12 derivazioni

derivazione	peso readout	input scaling
aVF	$8.8\% \pm 4.84$	2.70 ± 0.3816
aVL	$6.0\% \pm 1.95$	2.46 ± 0.4939
aVR	$2.7\% \pm 1.74$	1.32 ± 0.5580
I	$4.4\% \pm 1.62$	1.55 ± 0.6388
II	$8.2\% \pm 3.24$	2.16 ± 0.5728
III	$6.2\% \pm 3.17$	2.02 ± 0.5472
V1	$12.4\% \pm 2.80$	2.72 ± 1.2658
V2	$25.1\% \pm 9.80$	2.84 ± 0.3852
V3	$5.8\% \pm 0.66$	2.11 ± 0.3330
V4	$8.0\% \pm 3.15$	2.38 ± 0.7983
V5	$6.6\% \pm 2.66$	2.77 ± 0.6358
V6	$5.7\% \pm 3.61$	1.83 ± 0.5454

Tabella 5.20: Peso (in percentuale rispetto al totale) e input scaling attribuito training del modello alle 12 derivazioni

di interconnessione all'interno di ogni sub-reservoir, e conseguentemente un solo inter-reservoir scaling per ogni sub reservoir.

I risultati di questo modello, sia con 3 derivazioni (Tabella 5.21) che con 12 derivazioni (Tabella 5.23) sono nettamente inferiori rispetto a quelli ottenuti senza interconnessioni, e anche nelle Tabelle 5.22 e 5.24 si vede come né il readout né il training degli iperparametri sono stati in grado di individuare correttamente le derivazioni più rilevanti, e di conseguenza i risultati sono peggiorati. In questi modelli non si è ottenuta una convergenza della discesa del gradiente verso dei valori corretti per gli iperparametri, inclusi

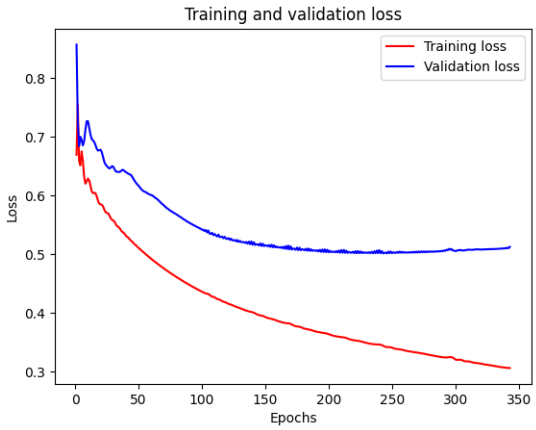

Figura 5.2: Curva della loss sul training set e validation set sul modello IRESN con 12 derivazioni

gli inter-reservoir scaling dei vari sub-reservoir. Per via dei valori non adeguati degli iperparametri il reservoir ha difficoltà (anche quando allenato col metodo diretto) a classificare correttamente gli stati calcolati nei sub-reservoir e di conseguenza i risultati sono addirittura peggiori del modello ESN (che univa in un unico reservoir tutte le derivazioni).

readout	training accuracy	validation accuracy	test accuracy	sensitivity	specificity
iterativo	$87.9\% \pm 2.96$	$85.4\% \pm 3.56$	$82.0\% \pm 5.71$	$72.8\% \pm 9.07$	$88.5\% \pm 5.60$
diretto	$94.1\% \pm 3.00$	$85.7\% \pm 2.56$	$85.0\% \pm 7.78$	$78.5\% \pm 13.69$	$89.4\% \pm 4.71$

Tabella 5.21: Risultati del modello IRESN con interconnessioni fra sub-reservoir e training degli iperparametri su 3 derivazioni

derivazione	peso readout	input scaling
V1	$25.7\% \pm 20.30$	2.03 ± 0.3096
V2	$17.1\% \pm 9.24$	2.05 ± 0.2545
V3	$57.2\% \pm 18.76$	1.91 ± 0.4225

Tabella 5.22: Peso (in percentuale rispetto al totale) e input scaling attribuito training del modello con interconnessioni alle 3 derivazioni

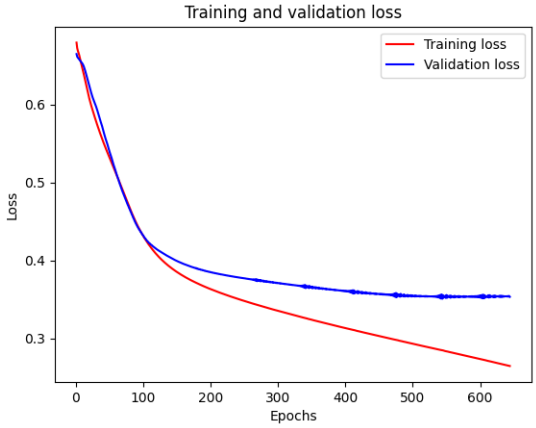

Figura 5.3: Curva della loss sul training set e validation set sul modello IRESN con 3 derivazioni e interconnessioni fra sub-reservoir

readout	training accuracy	validation accuracy	test accuracy	sensitivity	specificity
iterativo	$87.3\% \pm 3.70$	$73.8\% \pm 3.43$	$72.5\% \pm 7.25$	$59.8\% \pm 8.41$	$81.1\% \pm 5.73$
diretto	$97.3\% \pm 2.37$	$80.3\% \pm 1.04$	$75.5\% \pm 7.09$	$62.9\% \pm 13.40$	$83.9\% \pm 5.34$

Tabella 5.23: Risultati del modello IRESN con interconnessioni fra sub-reservoir e training degli iperparametri su 12 derivazioni

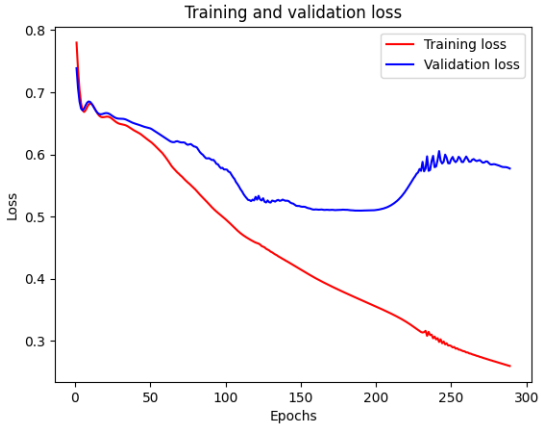

Figura 5.4: Curva della loss sul training set e validation set sul modello IRESN con 12 derivazioni e interconnessioni fra sub-reservoir

derivazione	peso readout	input scaling
aVF	$10.7\% \pm 8.58$	1.61 ± 0.0922
aVL	$6.7\% \pm 3.75$	1.53 ± 0.1757
aVR	$7.7\% \pm 4.88$	1.44 ± 0.0846
I	$7.0\% \pm 6.56$	1.38 ± 0.1063
II	$4.7\% \pm 2.22$	1.59 ± 0.0949
III	$6.6\% \pm 6.46$	1.57 ± 0.1222
V1	$7.0\% \pm 3.68$	1.58 ± 0.1334
V2	$9.2\% \pm 4.47$	1.74 ± 0.0503
V3	$10.3\% \pm 3.33$	1.63 ± 0.0569
V4	$9.9\% \pm 8.94$	1.44 ± 0.1674
V5	$11.5\% \pm 6.14$	1.57 ± 0.1587
V6	$8.8\% \pm 5.34$	1.40 ± 0.1450

Tabella 5.24: Peso (in percentuale rispetto al totale) e input scaling attribuito training del modello con interconnessioni alle 12 derivazioni

5.6 IRESN con sub-reservoir di dimensioni variabili

Sono stati tentati anche degli esperimenti preliminari sull'utilizzo del modello IRESN con sub-reservoir di dimensioni differenti nel caso con 12 derivazioni. In particolare per ogni sub-reservoir è stato introdotto un nuovo iperparametro n_i che regola il numero di unità k_i, rispetto al totale N, assegnate al reservoir i. La formula utilizzata è stata:

$$k_i = N \frac{n_i}{\sum_j n_j}$$

In questo modo alcuni sub-reservoir possono avere più unità di altri e quindi avere un peso maggiore sull'output del readout. Per scegliere il valore di questo iperparametro però non è possibile utilizzare il training degli iperparametri in quanto, una volta inizializzato il modello, il numero di unità non può essere modificato. Per gestire l'esplosione combinatoria del numero di iperparametri è stata quindi utilizzata una strategia di ricerca informata, la Bayesian search [23]. A differenza della grid search, la Bayesian search non

valuta le performance del modello in ogni combinazione possibile di valori degli iperparametri, ma da priorità a valutare le combinazioni che ritiene più interessanti, basandosi sui risultati ottenuti fino a quel momento. In questo modo tipicamente è possibile ottenere dei buoni valori per gli iperparametri senza dover esplorare tutto lo spazio. I risultati, riportati in Tabella 5.25, non portano però miglioramenti sul test set rispetto ai modelli IRESN precedentemente testati, nonostante sul validation set si osservino invece dei miglioramenti.

training accuracy	validation accuracy	test accuracy	sensitivity	specificity
$96.6\% \pm 0.36$	$85.8\% \pm 0.69$	$81.1\% \pm 4.32$	$73.1\% \pm 8.35$	$86.0\% \pm 6.29$

Tabella 5.25: Risultati del modello IRESN con sub-reservoir di dimensione variabile e Bayesian search

6. Analisi dei risultati

Innanzitutto si osserva come tutti i modelli con 3 derivazioni ottengano performance migliori rispetto ai modelli equivalenti con 12 derivazioni, il che è ragionevole perché molti dei dati presenti nelle 12 derivazioni non sono utili ai fini della diagnosi e costituiscono quindi del rumore che il modello deve filtrare per classificare correttamente le sequenze. La presenza di questo rumore, nei modelli con 12 derivazioni, impedisce ai modelli di ben generalizzare, nonostante venga effettuata la regolarizzazione sul readout. Come si vede nelle tabelle 5.2, 5.9 e 5.19 l'accuracy in training è molto alta, anche più alta di quella dei modelli con 3 derivazioni, ma sul validation set e test set le performance sono notevolmente peggiori. Una leggera diminuzione di questo fenomeno (nonostante poi i risultati sul test set rimangano equivalenti) si può ottenere utilizzando la lasso regression che, nei problemi l'input ha molte dimensioni, assieme al modello IRESN è in grado di indicare quali siano le dimensioni più rilevanti. Se ad esempio nel problema della diagnosi della sindrome di Brugada non fosse stato noto quali fossero le derivazioni più rilevanti, utilizzando IRESN con la lasso regression sarebbe stato possibile individuare 3 derivazioni rilevanti (V1, V2 e aVF) ed utilizzarle da sole per ottenere delle performance molto buone.

Nei modelli con 3 derivazioni si osserva come l'utilizzo del modello a comitato o del modello IRESN porti significativi miglioramenti rispetto al modello ESN, confermando l'utilità, nel caso della sindrome di Brugada, di processare separatamente le differenti derivazioni.

Nel confrontare il modello a comitato col modello IRESN va tenuto in conto, oltre alle prestazioni (che nel modello IRESN sono leggermente superiori), anche il tempo impiegato per il training e per calcolare l'output del modello. Durante il training, sia dei modelli ESN che compongono il comitato, sia del modello IRESN, l'operazione che richiede più tempo è il calcolo degli stati del reservoir, mentre il training del readout è molto più rapido e quindi trascurabile ai fini di questa analisi. All'interno del calcolo degli stati l'operazione più dispendiosa in termini di tempo è sicuramente il prodotto fra la matrice $\hat{\boldsymbol{W}}$ e il vettore degli stati $\boldsymbol{x}_{t-1}$. Si tratta del prodotto fra una matrice $n \times n$ e un vettore di lunghezza n,

dove n è il numero di unità del reservoir. Il numero di operazioni elementari (somme e moltiplicazioni) da eseguire per calcolare questo prodotto è $n(2n-1)$ che è quindi quadratico nella dimensione del reservoir. Se supponiamo che ogni modello ESN che compone un comitato di k modelli richieda $\frac{1}{k}$ delle unità di un modello IRESN il numero totale di operazioni per il comitato diventa $k\frac{n}{k}(\frac{n}{k}-1) = n(\frac{n}{k}-1)$. La complessità risulta comunque quadratica, ma diminuisce rispetto al modello IRESN equivalente all'aumentare di k.

Nella pratica però, per reservoir non molto grandi (meno di 2000 unità) come quelli dei modelli testati in questa tesi, questa maggior efficienza in tempo dei modelli a comitato non è evidente, anzi si osserva un aumento del tempo impiegato all'aumentare del numero di modelli nel comitato (mantenendo costante il numero totale di unità). Questo fenomeno è probabilmente legato all'implementazione del framework e delle librerie utilizzate (TensorFlow e Keras) che introducono un ulteriore costo costante per ogni modello. Nel caso del modello IRESN questo costo deve essere sommato al tempo di calcolo degli stati una sola volta, mentre nel caso del modello a comitato questo costo viene moltiplicato per il numero di modelli che compongono il comitato. Per reservoir di grandi dimensioni questo costo diventa trascurabile e quindi il tempo necessario al modello a comitato diventa inferiore a quello necessario al modello IRESN. Inoltre, come facilmente immaginabile, il calcolo degli stati del modello a comitato è molto semplice da parallelizzare, anche se durante il training di qualsiasi modello è possibile sfruttare il parallelismo all'interno della fase di validation (purchè non si utilizzino strategie informate come ad esempio la Bayesian search) o nella fase di model assessment se si utilizza la k-fold cross validation. Nella fase di predizione, quindi una volta terminato il training, nel modello a comitato è ancora possibile sfruttare il parallelismo, mentre nel modello IRESN no, ma, a meno che il problema non richieda di predire l'output relativo a molti input, il tempo risparmiato non è molto. Inoltre durante i test effettuati sui modelli a comitato, durante la fase di validation il numero di unità scelte, in media, dai modelli ESN che compongono il comitato è stato maggiore di $\frac{1}{k}$ di quelle scelte in media dai modelli IRESN, soprattutto nel caso con 12 derivazioni.

Finché è utilizzato un numero di unità contenuto nel reservoir (come è avvenuto in questa tesi), è quindi più conveniente in termini di tempo utilizzare il modello IRESN

rispetto al modello a comitato. Inoltre considerando che i risultati dei due modelli sono simili questo porta a preferire, come metodo per separare le derivazioni, il modello IRESN rispetto a quello a comitato.

Per quanto riguarda il training degli iperparametri (senza interconnessione fra sub-reservoir) si osserva come con 3 derivazioni i risultati siano analoghi a quelli ottenuti senza training degli iperparametri, dove quindi venivano utilizzati gli stessi iperparametri per tutti i sub-reservoir. Probabilmente in questo caso, in cui le dimensioni più rilevanti dell'input sono già state individuate e le altre sono state completamente scartate, avere dei valori degli iperparametri indipendenti per ogni sub-reservoir non è utile (anche se non è dannoso) e quindi i risultati sono sostanzialmente identici.

Con 12 derivazioni il training degli iperparametri risulta più complicato. Come si vede anche dalle curve di training in Figura 5.2, il modello raggiunge molto presto la condizione per l'arresto anticipato. Inoltre è presente un ampio divario fra accuracy sul training set e accuracy su validation e test set sia col readout iterativo che con quello diretto che evidenzia le difficoltà del modello a generalizzare. In termini di efficienza in tempo i modelli con training degli iperparametri sono nettamente inferiori agli altri. Ogni epoca della discesa del gradiente richiede che vengano calcolati gli stati del reservoir, ed al tempo necessario per il calcolo degli stati si somma il tempo necessario per tenere traccia delle operazioni effettuate e per calcolare il gradiente.

È però interessante notare come i modelli IRESN sia con che senza training degli iperparametri assegnino una minore importanza alla derivazione V3 rispetto a V1 e V2 (come si osserva nelle tabelle 5.8 e 5.18), indicando quindi una minor rilevanza per la diagnosi di V3. Questa ipotesi è sostenuta anche dai risultati del modello a comitato, dove se si osservano i risultati dei modelli che compongono il comitato sulle singole derivazioni (tabella 5.4) si nota come su V3 i risultati siano molto più bassi (sia in training che in validation e test).

I modelli con interconnessione fra sub-reservoir invece, oltre a richiedere più tempo per il training, hanno performance inferiori a quelle ottenute con il modello ESN. Questi modelli evidentemente non sono addatti al problema (con questo numero di dati a disposizione), e falliscono nel filtrare il rumore introdotto dalle derivazioni superflue, ottenendo

quindi scarsi risultati. Inoltre anche ottenere una convergenza con la discesa del gradiente in questo modello risulta più complicato rispetto a quello senza interconnessioni. Per questo motivo il valore del learning rate è stato abbassato ma, come si nota in Figura 5.4, la validation loss è ancora abbastanza irregolare, mentre la training loss non presenta la classica forma di una curva di apprendimento per via del basso learning rate.

Neanche il modello IRESN con sub-reservoir con dimensioni variabili ha portato a dei miglioramenti nei risultati. L'accuracy sul validation set è più alta rispetto ai precedenti mdelli con 12 derivazioni, ma un corrispondente aumento di accuracy sul test set non avviene, segno del fatto che il modello, dopo la validation, non è in grado di generalizzare al meglio su nuovi dati.

Conclusioni

Sono stati analizzati vari modelli di machine learninig, basati sul reservoir computing, per affrontare il problema della diagnosi della sindrome di Brugada dall'ECG come serie temporale multivariata.

Dai risultati si osserva che utilizzando tutte e 12 le derivazioni, con i modelli testati, non è stato possibile ottenere performance migliori rispetto alla baseline con 3 derivazioni. La capacità dei modelli di individuare le derivazioni più importanti, benché presente, non è stata sufficiente per colmare le difficoltà causate dal "rumore" che le derivazioni poco rilevanti hanno introdotto.

Il modello IRESN con il training degli iperparmetri, introdotto in questa tesi per affrontare l'aumento esponenziale del numero di combinazioni di iperparametri, nel caso di 12 derivazioni ha ottenuto un lieve miglioramento rispetto al modello ESN con 12 derivazioni, dimostrando delle potenzialità, a fronte però di un maggior dispendio di tempo per il training del modello.

Nel caso con 3 derivazioni i risultati ottenuti col training degli iperparametri sono equivalenti a quelli ottenuti col modello IRESN semplice (in cui si usava lo stesso iperparametro per ogni sub-reservoir) indicando che in questo caso, in cui le derivazioni sono tutte rilevanti (anche se alcune più di altre), non sia necessario diversificare il valore degli iperparametri in ogni sub-reservoir.

Osservando il valore dei pesi del readout in quasi tutti modelli IRESN testati (fatta eccezione per i modelli con interconnessione fra sub-reservor che non hanno ottenuto dei buoni risultati) e i valori di input scaling nei modelli con training degli iperparametri, è possibile notare come i modelli, in media, siano stati in grado di discriminare fra derivazioni rilevanti e non (soprattutto con l'utilizzo della lasso regression nel readout).

Le derivazioni ritenute più importanti dai modelli coincidono quasi completamente con quelle individuate dall'esperienza clinica ed è stato anche possibile osservare come utilizzando le derivazioni più importanti individuate dai modelli IRESN (V1 V2 e aVF) i risultati ottenuti siano stati in linea con quelli ottenuti con le 3 derivazioni classiche

(V1 V2 e V3). La capacità di individuare le dimensioni più importanti dell'input non è comunque sufficiente per ottenere dei risultati al pari dei modelli con 3 derivazioni quando vengono usate tutte e 12 le derivazioni. Nonostante ciò, in applicazioni in cui non è noto a priori quali siano le dimensioni dell'input più rilevanti, considerare solo le dimensioni indicate come rilevanti da un insieme di più modelli IRESN può rappresentare una valida strategia per approcciare il problema e ridurre la dimensionalità dell'input.

La separazione delle dimensioni dell'input, nei modelli con 3 derivazioni, ha portato a dei significativi miglioramenti rispetto alla baseline (il modello ESN che processava tutte le derivazioni in un unico reservoir), sia con i modelli IRESN che con quelli a comitato. L'utilizzo delle interconnessioni fra sub-reservoir invece non ha portato dei buoni risultati, forse perchè il numero di dati di training a disposizione non era sufficiente per far si che la discesa del gradiente individuasse dei buoni valori per gli inter-reservoir scaling.

Questi risultati indicando quindi che, nel caso specifico della Sindrome di Brugada, analizzare le derivazioni separatamente sia più vantaggioso, ma solo nel caso con 3 derivazioni questo vantaggio riesce a concretizzarsi sul datset disponibie in questa fase del progetto.

In futuro, disponendo di un maggior numero di dati, sarebbe interessante ritestare tutti i modelli IRESN con 12 derivazioni per verificare se il divario fra accuracy sul training set e sul test set possa ridursi e se i modelli riescano a pareggiare le performance del modello ESN con 3 derivazioni (entrambi testati con la stessa quantità di dati). Il modello IRESN su 12 derivazioni probabilmente gioverebbe maggiormente di un aumento dei dati nel dataset, in quanto potrebbe riuscire a discriminare meglio tra derivazioni importanti e non rilevanti per il problema.

Altri miglioramenti ai modelli presentati in questa tesi possono essere testati in futuro. Per i modelli a comitato sarebbe interessante testare altre strategie di costruzione del comitato (come il boosting) o politiche di voto differenti, ricorrendo all'ensamble stacking (anziché usare un voto a maggioranza utilizzare l'accuracy dei modello, misurata sul validation set, per determinare il peso di quel modello sul voto finale). In questo modo ad esempio il modello ESN che riceve la derivazione V2 avrebbe un peso maggiore sul voto finale rispetto agli altri modelli nel comitato e si potrebbe quindi ottenere un

miglioramento delle performance sia con 3 che con 12 derivazioni.

Per i modelli IRESN sarebbe interessante valutare altri approcci per il problema dell'aumento esponenziale del numero di combinazioni di valori degli iperparametri. Il training degli iperparmetri, sperimentato in questa tesi, ha portato solo dei piccoli miglioramenti, soprattutto rispetto a quelli sperati, e necessita di molto tempo per la fase di training, mentre ulteriori ed estesi esperimenti con la Bayesian search sarebbero necessari per stabilire se i risultati ottenuti negli esperimenti preliminari possano essere migliorati.

Ringraziamenti

Ringrazio il Professor Alessio Micheli, il Dottor Claudio Gallicchio e il Dottor Luca Pedrelli per il supporto dato durante lo svolgimento della tesi e per l'opportunità offertami di lavorare a questo progetto. Ringrazio anche la famiglia e gli amici per il sostegno.

Bibliografia

[1] L. Argentieri, C. Gallicchio, and A. Micheli. Input routed echo state networks. In *Proc. of the 30th European Symposium on Artificial Neural Networks (ESANN)*, pages 405–410. i6doc.com, 2022.

[2] F. Chollet. *Deep Learning with Python*, chapter 10 Deep learning for timeseries. Manning Pubblications, 2 edition, 2021.

[3] G. M. Dimitri, C. Gallicchio, A. Micheli, M. A. Morales, E. Ungaro, and F. Vozzi. A preliminary evaluation of echo state networks for brugada syndrome classification. In *2021 IEEE Symposium Series on Computational Intelligence (SSCI)*, pages 01–08, 2021.

[4] C. Gallicchio, A. Micheli, and L. Pedrelli. Fast spectral radius initialization for recurrent neural networks. In L. Oneto, N. Navarin, A. Sperduti, and D. Anguita, editors, *Recent Advances in Big Data and Deep Learning*, pages 380–390, Cham, 2020. Springer International Publishing.

[5] D.B. Geselowitz. On the theory of the electrocardiogram. *Proceedings of the IEEE*, 77(6):857–876, 1989.

[6] S. Haykin. *Neural Networks and Learning Machines*, chapter 13 Neurodynamics. Pearson Education, 3 edition, 2008.

[7] H. Jaeger. The" echo state" approach to analysing and training recurrent neural networks-with an erratum note'. *Bonn, Germany: German National Research Center for Information Technology GMD Technical Report*, 148, 01 2001.

[8] H. Jaeger and H. Haas. Harnessing nonlinearity: Predicting chaotic systems and saving energy in wireless communication. *Science*, 304(5667):78–80, 2004.

[9] M. Lukoševičius. *A Practical Guide to Applying Echo State Networks*, pages 659–686. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[10] M. Lukoševičius and H. Jaeger. Reservoir computing approaches to recurrent neural network training. *Computer Science Review*, 3(3):127–149, 2009.

[11] S. McStay. Recording a 12-lead electrocardiogram (ecg). *British Journal of Nursing*, 28(12):756–760, 2019.

[12] T. Mitchell, B. Buchanan, G. DeJong, T. Dietterich, P. Rosenbloom, and A. Waibel. Machine learning. *Annual Review of Computer Science*, 4(1):417–433, 1990.

[13] T.M. Mitchell. *Machine Learning*, chapter 4 Aritficial Neural Network. McGraw-Hill Education, 1997.

[14] T.M. Mitchell. *Machine Learning*, chapter 3 Decison Tree learning. McGraw-Hill Education, 1997.

[15] M.N. Obeyesekere, G.J. Klein, S. Modi, P. Leong-Sit, L.J. Gula, R. Yee, A.C. Skanes, and A.D. Krahn. How to perform and interpret provocative testing for the diagnosis of brugada syndrome, long-qt syndrome, and catecholaminergic polymorphic ventricular tachycardia. *Circulation: Arrhythmia and Electrophysiology*, 4(6):958–964, 2011.

[16] M.M. Polovina, M. Vukicevic, B. Banko, G.Y.H. Lip, and T.S. Potpara. Brugada syndrome: A general cardiologist's perspective. *European Journal of Internal Medicine*, 44:19–27, 2017.

[17] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[18] A. Shimizu. Indication of icd in brugada syndrome. *Journal of Arrhythmia*, 29(2):110–116, 2013. Special Issue: Brugada Syndrome from bench to bedside.

[19] P. Norvig S.J. Russell. *Artificial Intelligence: A Modern Approach*, chapter 18 Learning from examples. Pearson Education, 3 edition, 2010.

[20] G. Tanaka, T. Yamane, J.B. Héroux, R. Nakane, N. Kanazawa, S. Takeda, H. Numata, D. Nakano, and A. Hirose. Recent advances in physical reservoir computing: A review. *Neural Networks*, 115:100–123, 2019.

[21] D. Verstraeten, B. Schrauwen, M. D'Haene, and D. Stroobandt. An experimental unification of reservoir computing methods. *Neural Networks*, 20(3):391–403, 2007. Echo State Networks and Liquid State Machines.

[22] A.A.M. Wilde, C. Antzelevitch, M. Borggrefe, J. Brugada, R. Brugada, P. Brugada, D. Corrado, R.N.W. Hauer, R.S. Kass, K. Nademanee, S.G. Priori, and J.A. Towbin. Proposed diagnostic criteria for the brugada syndrome. *Circulation*, 106(19):2514–2519, 2002.

[23] J. Wu, X. Y. Chen, H. Zhang, L. D. Xiong, H. Lei, and S. H. Deng. Hyperparameter optimization for machine learning models based on bayesian optimizationb. *Journal of Electronic Science and Technology*, 17(1):26–40, 2019.

[24] I. B. Yildiz, H. Jaeger, and S. J. Kiebel. Re-visiting the echo state property. *Neural Networks*, 35:1–9, 2012.

[25] J. Zowe. Nondifferentiable optimization. In *Computational Mathematical Programming*, pages 323–356, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.