

UNIMORE

UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

Dipartimento Scienze e Metodi dell'Ingegneria

Corso di Laurea Magistrale in Ingegneria Gestionale

Reti neurali feed-forward per la predizione delle band gap in metamateriali a travi periodiche piegate

Relatore:

Prof.ssa Stefania Monica

Laureanda:

Martina Mantovani

Anno Accademico 2025/2026

INDICE

INDICE FIGURE e TABELLE V
INDICE FORMULE VII
SOMMARIO IX
INTRODUZIONE XI
CAPITOLO 1: I Metamateriali 1
1.1 Contesto generale dei metamateriali 1
1.1.1 La definizione e la storia 1
1.1.2 Le caratteristiche e le proprietà 2
1.1.3 Classificazione dei metamateriali 3
1.1.4 Contesti applicativi 4
1.2 Struttura periodica 5
1.2.1 La cella unitaria nelle travi piegate periodicamente 5
1.3 Bande proibite nelle strutture periodiche 7
1.3.1 Meccanismi di formazione delle band gap 8
1.3.2 Le curve di dispersione per l'analisi grafica delle band gap 9
1.3.3 Analisi parametrica per la formazione di band gap 12
CAPITOLO 2: Machine Learning 15
2.1 Introduzione al concetto di machine learning 15
2.1.1 Modelli di apprendimento 16
2.1.2 Principali problemi in cui viene utilizzato il Machine Learning 17
2.2 I principali algoritmi di Machine Learning 19
2.3 Introduzione al concetto di reti neurali 23
2.3.1 Dal concetto di percettrone e sigmoid neuron alla definizione delle reti neurali 24
2.3.2 L'architettura delle reti neurali 26
2.3.3 Principali componenti delle reti neurali 28
2.3.4 Processo di addestramento 36
2.3.5 Metriche per la valutazione delle prestazioni 43
2.3.6 Contesti di applicazione delle reti neurali 49
CAPITOLO 3: Applicazione di un modello surrogato di machine learning per la predizione di bande di stop in metamateriali a trave piegata periodicamente 51
3.1 Introduzione al problema e idea per la risoluzione 51
3.2 Fase 0: generazione del dataset 54
3.3 Fase 1: analisi dei dati 55

3.3.1 Distribuzione dei valori di input 56
3.3.2 Influenza che i diversi valori hanno sulla stop band 58
3.3.3 Distribuzione delle frequenze di BG_start e BG_end 60
3.4 Fase 2: preprocessing del dataset 63
3.4.1 Esecuzione dell'encoding delle variabili di input 64
3.4.2 Trasformazione logaritmica dei valori delle frequenze 64
3.4.3 Regole per la divisione del dataset 66
3.4.4 Normalizzazione di tipo Min-Max dei valori di input 70
3.4.5 Definizione delle class weights per il classificatore 71
3.5 Fase 3: training della rete 74
3.5.1 Primo tentativo: architettura con backbone condiviso per classificazione e regressione 74
3.5.2 Secondo tentativo: riduzione dell'architettura a backbone condiviso 83
3.5.3 Soluzione definitiva: definizione di due reti completamente indipendenti 86
3.6 Fase 4: ottimizzazione inversa 93
3.6.1 Implementazione funzione di ottimizzazione 94
3.6.2 Analisi e validazione dei risultati finali ottenuti dal processo di ottimizzazione inversa 95
3.6.3 Fase di validazione tramite codice Matlab 98
3.6.4 Ottimizzazione inversa con vincolo sulla frequenza 104
3.7 Introduzione del rumore 106
3.7.1 Motivazioni alla base dell'introduzione del rumore nei dati di input 106
3.7.2 Fase 1: introduzione del modello di rumore gaussiano 107
3.7.3 Fase 2: valutazione dei risultati della rete sul dataset originale con dati perturbati 108
3.7.4 Fase 3: generazione del nuovo dataset con valori perturbati tramite il codice MATLAB 111
3.7.5 Fase 4: confronto tra risultati ottenuti con il dataset originale e con quello perturbato 112
3.7.6 Conclusioni finali sull'introduzione del rumore 114
CONCLUSIONI 115
BIBLIOGRAFIA e SITOGRAFIA 118

INDICE FIGURE e TABELLE

Figura 1: Classificazione dei metamateriali 3
Figura 2:Rappresentazione della cella unitaria e della struttura periodica Fonte: Manconi et al. (2026) 6
Figura 3: Rappresentazione di una configurazione accoppiata e disaccoppiata di una trave piegata periodicamente tramite metodo esatto DSM Fonte: Manconi et al. (2026) 6
Figura 4: Curve di dispersione per la rappresentazione grafica delle bande di stop Fonte: Manconi et al. (2026) 11
Figura 5: Grafico di dispersione di una band gap completa Fonte: Manconi et al. (2026) 12
Figura 6: Rappresentazione grafica delle due possibilità analizzate per la variazione di α Fonte: Manconi et al. (2026) 13
Figura 7: Grafici di dispersione al variare dell'angolo di piegatura Fonte: Manconi et al. (2026) 13
Figura 8: Grafici di dispersione rappresentanti le band gap al variare dello spessore della cella unitaria Fonte: Manconi et al. (2026) 13
Figura 9: Curve di dispersione rappresentanti le band gap al variare della curvatura della piega periodica Fonte: Manconi et al. (2026) 14
Figura 10: Analisi grafica delle relazioni tra i concetti di Data Science, AI, ML, DL e reti neurali Fonte: Choi et al. (2020) 16
Figura 11: Problemi di classificazione a confronto con problemi di regressione Fonte: Sarker et al. (2021) 18
Figura 12: Esempio di Decision Tree Fonte: Sarker et al. (2021) 22
Figura 13: Esempio di algoritmo di Random Forest Fonte: Sarker et al. (2021) 22
Figura 14: Rappresentazione grafica dei principali componenti di un neurone Fonte: Zilouchian et al. (2001) 24
Figura 15: Esemplificazione della struttura di un percettrone Fonte: Haykin et al. (2009) 24
Figura 16: Rappresentazione grafica di una rete neurale con due hidden layers Fonte: LeCun et al. (2015) 26
Figura 17: Esempio di shallow e deep neural network Fonte: Mienye & Swart (2024) 27
Figura 18: Esempio di grafo computazionale Fonte: Goodfellow et al. (2016) 40
Figura 19: Curva ad U rappresentate la relazione tra learning rate e errore di training Fonte: Goodfellow et al. (2016) 41
Figura 20: Figura che rappresenta il funzionamento del dropout Fonte: Srivastava et al. (2014) 43
Figura 21: Training curve Fonte: Goodfellow et al. (2016) 44
Figura 22: Learning curve Fonte: Hastie et al. (2009) 44
Figura 23: Matrice di confusione Fonte: Fawcett (2006) 45
Figura 24: Esempio di trave piegata periodicamente 52
Figura 25: Grafico a barre e a torta che mostra la distribuzione delle classi 57
Figura 26: Grafico a barre che evidenzia la distribuzione della colonna Reason 57
Figura 27: Grafico a barre che mostra la distribuzione degli status per i diversi parametri di input 58

Figura 28: Grafico a barre che mostra la correlazione tra valori di input e valori di output 60
Figura 29: Grafici relativi alla distribuzione dei valori di output 61
Figura 30: Tabella con valori riassuntivi delle distribuzioni dei valori di output..... 61
Figura 31: Scatterplot che evidenzia la relazione tra BG_start e BG_end 62
Figura 32: Istogramma rappresentante la distribuzione dei valori di output prima e dopo la trasformazione logaritmica 65
Figura 33: Distribuzione valori di output dopo la normalizzazione logaritmica 66
Figura 34: Divisione training, validatione test set 68
Figura 35: Istogramma rappresentate la divisione delle classi nei diversi set dati 69
Figura 36: Grafico che rappresenta la distribuzione dei diversi valori di Alpha nei subset creati..... 70
Figura 37: tabella riassuntiva dei class weights 73
Figura 38: Grafico per la rappresentazione grafica dei classe weights delle diverse classi 73
Figura 39: Rappresentazione grafica rete neurale multi task con backbone condiviso .. 78
Figura 40: Risultati performance classificatore primo tentativo 79
Figura 41: Matrice di confusione per la valutazione della prestazione del classificatore 80
Figura 42: Risultati performance regressore primo tentativo 81
Figura 43: Grafici per la valutazione della performance del regressore per il primo tentativo 81
Figura 44: Curve di apprendimento primo tentativo 82
Figura 45: Andamento learning rate primo tentativo..... 83
Figura 46: Risultati performance classificatore primo tentativo con aumento del dropout 84
Figura 47: Matrice di confusione per la valutazione della prestazione del classificatore con dropout aumentato..... 84
Figura 48: Risultati performance regressore primo tentativo con dropout aumentato ... 84
Figura 49: Grafici per la valutazione della performance del regressore per il primo tentativo con dropout aumentato..... 85
Figura 50: Curve di apprendimento primo tentativo con dropout aumentato..... 85
Figura 51: Andamento learning rate primo tentativo con dropout aumentato..... 85
Figura 52: Rappresentazione grafica delle due reti neurali separate 89
Figura 53: Risultati performance ClassifierNet 89
Figura 54: Matrice di confusione ClassifierNet..... 90
Figura 55: Valori metriche performance RegressorNet..... 91
Figura 56: Grafico valutazione performance RegressorNet 91
Figura 57: Curve di apprendimento ClassifierNet e RegressorNet 92
Figura 58: Classifica configurazioni che massimizzano la band gap 96
Figura 59: Grafici relativi all'ottimizzazione inversa con n=50 96
Figura 60: Grafici relativi all'ottimizzazione inversa con n=10 97
Figura 61: Configurazioni che massimizzano la dimensione della band gap con n=10 . 98
Figura 62: Configurazioni che massimizzano l'ampiezza della band gap con n=10 99
Figura 63: Risultati validazione Matlab con n=10..... 99
Figura 64: Risultati consuntivi validazione configurazioni n=10..... 99

Figura 65: Figura con i risultati di validazione migliori per n=10 100
Figura 66: Configurazioni che massimizzano l'ampiezza delle band gap con n=50 100
Figura 67: Configurazioni e risultati validazione Matlab con n = 50 101
Figura 68: Risultati consuntivi validazione Matlab con n=50 101
Figura 69: Configurazioni e risultati testati su codice Matlab con distribuzione su tutto lo spazio 102
Figura 70: Risultati validazione con configurazioni su tutto lo spazio 103
Figura 71: Risultati finali validazione Matlab 104
Figura 72: Risultati classificatore sulle 5 versioni di dataset perturbati negli input 108
Figura 73: Risultati metriche di performance del regressore BG_start sulle 5 versioni di dataset perturbato 109
Figura 74: Risultati metriche di performance del regressore BG_end sulle 5 versioni di dataset perturbato 109
Figura 75: Valore metriche performance dataset originale 110
Figura 76: Confronto misure di performance tra test set originale e le 5 versioni perturbate 110
Figura 77: Divisioni classi dataset perturbato 112
Figura 78: Variazioni valori di output dataset originale e perturbato 112
Figura 79: Effetto rumore su valori di output 113
Figura 80: Tabelle confronto metriche performance dataset originale e perturbato 114

INDICE FORMULE

Equazione 1: Equazione di Maxwell (Indice di rifrazione) 2
Equazione 2: Equazione retta problemi di regressione lineare 20
Equazione 3: Equazione sigmoide 20
Equazione 4: Equazione per il calcolo dell'output di un percettrone 25
Equazione 5: Equazione per il calcolo degli output con l'introduzione di un bias 25
Equazione 6: Calcolo dell'output di un sigmoid neuron 25
Equazione 7: Funzione sigmoide 25
Equazione 8: Equazione di una funzione lineare 29
Equazione 9: Equazione di una funzione binaria o bipolare 29
Equazione 10: Equazione di una piecewise linear function 30
Equazione 11: Equazione di una funzione iperbolica tangente 30
Equazione 12: Equazione di una funzione di attivazione di tipo softmax(z) 30
Equazione 13: Equazione del Mean Squared Error 31
Equazione 14: Equazione della Cross Entropy 32
Equazione 15: Funzione di Huber Loss 32
Equazione 16: Regola SGD 33
Equazione 17: Somma dei quadrati dei gradienti 34
Equazione 18: Equazione di aggiornamento funzione AdaGrad 34
Equazione 19: Equazione della media mobile esponenziale dei quadrati dei gradienti . 35
Equazione 20: Equazione aggiornamento RMSProp 35

Equazione 21: Formula aggiornamento funzione Adam 35
Equazione 22: Formule per il primo e il secondo momento 35
Equazione 23: Formula per il calcolo della recall 46
Equazione 24: Formula per il calcolo del false positive rate 46
Equazione 25: Formula per il calcolo della specificity 46
Equazione 26: Formula per il calcolo della precision 46
Equazione 27: Formula per il calcolo dell'accuracy 47
Equazione 28: Formula per il calcolo di F1-score 47
Equazione 29: Formula per il calcolo MAE 48
Equazione 30: Formula per il calcolo RMSE 48
Equazione 31: Formula per il calcolo del coefficiente di determinazione 48
Equazione 32: Formula per il passaggio da scala logaritmica a valore in Hz 65
Equazione 33: Formula per la normalizzazione Min-Max 71
Equazione 34: Formula per la definizione die classe weights 72
Equazione 35: Equazione strato lineare 75
Equazione 36: Equazione per il calcolo della loss function totale nel primo tentativo 76
Equazione 37: Equazione valore di input perturbato 107

SOMMARIO

Il seguente lavoro di tesi ha l'obiettivo di sviluppare un modello di machine learning surrogato, il quale permetta di predire un comportamento specifico dei metamateriali, ossia quello di essere in grado di inibire, in certi intervalli di frequenza, le onde elettromagnetiche che li attraversano, queste zone di inibizione sono chiamate bande di stop. Il progetto si concentra su strutture periodiche a trave piegata, composte da celle unitarie ripetute, in quanto, come sarà descritto nel corso dell'elaborato, la periodicità è la caratteristica fisica in grado di bloccare, o deviare, le onde elettromagnetiche. Attualmente, tramite il Dynamic Stiffness Method, che viene implementato tramite un codice Matlab, è possibile calcolare, partendo dai parametri fondamentali dell'unità ripetitiva, i valori di inizio e di fine delle bande di stop, il problema principale di questo modello è il costo computazionale, esso, infatti, impiega alcuni minuti, per ogni configurazione, per la generazione dell'output, rendendo, come è facile intuire, impossibile l'analisi di un numero elevato di strutture, in quanto richiederebbe tempi non sostenibili. Per questo motivo entra in gioco il machine learning, nello specifico nel corso di questo lavoro di tesi, è stato realizzato un modello surrogato basato su reti neurali feed-forward, in grado di approssimare il comportamento del codice Matlab in pochissimi secondi, risolvendo, quindi, il problema principale del modello. A partire da un dataset di configurazioni affidabili, generato direttamente dal codice originale, sono stati realizzati due modelli di reti neurali completamente indipendenti, una dedicata alla classificazione, in grado di garantire la validità fisica delle configurazioni, mentre l'altra dedicata alla regressione, la quale, quindi, si occupa di predire i valori di inizio e fine delle bande di inibizione. La rete dedicata alla classificazione è in grado di raggiungere un'accuratezza del 98%, mentre per la regressione il coefficiente R^2 ha un valore superiore a 0.95, i valori assunti da queste metriche garantiscono l'eccellente prestazione delle reti neurali create, che sono in grado di simulare milioni di configurazioni in pochissimi secondi. Per ampliare l'analisi è stato aggiunto anche il calcolo dei parametri di input della configurazione che massimizza la banda di stop, valori che sono stati validati direttamente con simulazioni Matlab, oltre all'aggiunta di un sistema che permettesse di predire i migliori parametri di input, che permettano di ottenere una banda di inibizione, se esiste, centrata in una frequenza decisa dall'operatore. Infine, tramite un'analisi di robustezza, è stato possibile dimostrare che il modello mantiene gli stessi valori di prestazione, o, comunque, presenta variazioni minime trascurabili, anche dopo l'introduzione di valori di incertezza di manifattura nell'ordine del 1-2%.

INTRODUZIONE

I metamateriali sono materiali che possiedono proprietà e caratteristiche non presenti spontaneamente in natura, che dipendono esclusivamente dalla loro struttura. I metamateriali a struttura periodica, in particolare le travi piegate, sono sistemi nei quali una cella unitaria si ripete nello spazio, seguendo un pattern preciso e definito, a meno di alcuni valori di incertezza dovuti alle tolleranze delle macchine dedicate alla manifattura. Grazie alla periodicità, queste strutture, sono in grado di mostrare una delle peculiarità più importanti, ossia quella di essere in grado di inibire, a certe frequenze, le onde elettromagnetiche che le colpiscono, queste zone di inibizione sono chiamate bande di stop. Quindi, a partire dai parametri di input caratteristici delle celle unitarie, è possibile derivare i valori di frequenza di inizio e di fine delle bande di inibizione, dai quali è possibile calcolare la dimensione e la media delle suddette zone. Il concetto di inibizione delle onde, è uno degli argomenti più studiati da ingegneri e ricercatori negli ultimi anni, in quanto garantisce molteplici applicazioni in campi differenti. Gli studiosi dell'università di Parma, nello specifico, hanno un duplice obiettivo: il primo quello di predire rapidamente, ma con precisione e accuratezza, se, a una certa frequenza, esiste una banda di stop, parallelamente vogliono essere in grado di identificare i parametri di input che permettono di massimizzare la banda, per garantire il miglior isolamento vibrazionale. Per raggiungere questi obiettivi, attualmente, viene utilizzato un codice Matlab in grado di svolgere una simulazione numerica utilizzando il Dynamic Stiffness Method. Il problema principale di questo modello sono i costi computazionali, esso, infatti, richiede, per ogni configurazione, qualche minuto per il calcolo dei valori della banda di inibizione, rendendo impossibile l'analisi di centinaia, o migliaia, di strutture differenti. Negli ultimi anni, sono molteplici le pubblicazioni in letteratura le quali introducono modelli surrogati di machine learning per superare la limitazione dei costi computazionali elevati dei modelli classici, il machine learning, infatti, riesce a garantire di ottenere risultati con una precisione paragonabile a quella dei codici originali, ma in tempi inferiori. Nel campo dei metamateriali sono stati pubblicati, negli ultimi anni, diversi contributi relativi all'implementazione di modelli di machine learning nel campo dei metamateriali, come: Jin et al. [2022] che propongono un framework di machine learning per il design di metamateriali fononici, Liu et al. [2023] i quali generano reti neurali per la predizione del design di metamateriali acustici, con livelli di errore del 2%, oppure ancora Tran et al. [2025] i quali propongono un modello di reti neurali in grado di predire il design inverso di strutture 2D della cella unitaria ripetitiva delle strutture periodiche dei metamateriali. A differenza delle precedenti pubblicazioni, il seguente progetto di tesi ha lo scopo di sviluppare un modello surrogato, basato su reti neurali feed-forward, in grado di approssimare i valori delle simulazioni del codice Matlab citato precedentemente, differenziandosi dalle pubblicazioni, sopra riportate, in quanto si basa su strutture a trave piegata periodicamente, elemento centrale degli studi condotti dalla professoressa Manconi presso l'Università di Parma (Manconi et al. [2021]; Manconi et al. [2023]; Manconi et al. [2025]). Nello specifico l'elaborato affronta lo sviluppo e addestramento di due reti neurali completamente indipendenti, in modo tale da risolvere il problema legato all'interferenza nel calcolo dei gradienti nei modelli multi-task, aggiungendo, poi, anche un'analisi relativa all'incertezza dovuta alle tolleranze delle

macchine manifatturiere. Il modello surrogato realizzato, quindi, può essere utilizzato per scopi differenti, come: ottimizzazione inversa, per il calcolo dei parametri di struttura in grado di garantire la massimizzazione della banda di inibizione, oppure per un'analisi di robustezza, in modo tale da quantificare l'effetto che le incertezze di manifattura hanno sul modello, ma anche sulla struttura periodica, oltre ad essere in grado di calcolare i parametri caratteristici della cella unitaria che garantiscono una banda di stop centrata in una frequenza definita dall'operatore. Per la realizzazione del progetto è stato, in primo luogo, generato un dataset di configurazioni, partendo direttamente dal codice Matlab originale, del quale si conosce la precisione e la massima affidabilità. Il set di dati è stato, poi, sottoposto a una fase di analisi esplorativa e di preprocessing, le quali includono attività come: riconoscimento di pattern, caratteristiche o proprietà intrinseche dei dati, trasformazione logaritmica dei valori di output, normalizzazione Min-Max degli input, suddivisione in training, validation e test set e assegnazione dei class weights. Al termine di queste due fasi è stato possibile ottenere un dataset pulito e pronto per l'addestramento delle reti. Successivamente si è passati al cuore del progetto, ossia la realizzazione, tramite Pytorch, delle due reti feed-forward indipendenti: la prima ClassifeirNet dedicata alla classificazione, la quale permette di garantire la validità fisica delle configurazioni, mentre la seconda, RegressorNet, dedicata alla regressione, ossia alla predizione dei valori della banda di stop. Il percorso seguito per la generazione del modello di machine learning basato su due reti neurali indipendenti, è iterativo e ha permesso di dimostrare una profonda interferenza dei due task, nel calcolo dei gradienti per l'aggiornamento dei pesi, rendendo impossibile l'implementazione di un singolo modello multi-task. Dopo la realizzazione del modello, esso viene utilizzato per l'ottimizzazione inversa, per il calcolo dei parametri fondamentali per la generazione di bande di inibizione centrate in una frequenza definita dall'operatore e, anche, per la valutazione delle prestazioni dopo l'introduzione di rumore nei parametri. La tesi è organizzata in tre capitoli, di cui i primi due prettamente teorici e compilativi, volti a fornire le conoscenze teoriche necessarie per la comprensione dell'ultimo capitolo, andando più nel dettaglio: il primo è dedicato all'introduzione al concetto di metamateriali, la loro storia, le loro proprietà e caratteristiche, oltre a un approfondimento sul concetto di strutture periodiche a trave piegata composte da celle unitarie, la definizione delle bande di inibizione con le loro caratteristiche e proprietà, oltre al meccanismo di formazione, con particolare attenzione al Dynamic Stiffness Method, le curve di dispersione e mettendo in luce le problematiche e limitazioni del codice Matlab, infine il primo capitolo si concluderà con un breve elenco di alcune delle più famose e importanti applicazioni dei metamateriali. Il secondo capitolo, invece, è dedicato alla trattazione del machine learning, in particolare ai modelli di reti neurali feed-forward, con le proprie tecniche di regolarizzazione, la struttura e i principali componenti, gli iperparametri fondamentali, il processo di addestramento e le metriche di valutazione delle performance. Infine, l'ultimo capitolo, è dedicato alla descrizione delle fasi e caratteristiche del progetto di tesi svolto: a partire dalla generazione del dataset direttamente dal codice Matlab, la fase di esplorazione e preprocessing, per ottenere un set di dati raffinato da poter utilizzare per l'addestramento delle reti, la fase di realizzazione, addestramento e, infine, di valutazione della performance del modello creato, la fase di ottimizzazione inversa e l'analisi della robustezza con l'introduzione del rumore nei parametri di input. Per terminare verranno

analizzate le conclusioni tratte dai risultati ottenuti e i possibili sviluppi futuri, per rendere il modello ancora più efficiente ed efficacie.

CAPITOLO 1: I Metamateriali

1.1 Contesto generale dei metamateriali

1.1.1 La definizione e la storia

Il termine "metamateriali" viene coniato nel 1999 presso l'università di Austin, in Texas, da Rodger Walser, che decise di mettere insieme il prefisso "oltre", che in greco si scrive "μετα", con il termine materials, riuscendo, così, a definire per la prima volta i "metamateriali". Essi, come suggerisce il nome stesso, sono materiali che vanno oltre quelli già esistenti in natura. Uno degli elementi cardine di questi materiali, sta proprio nel fatto che le loro caratteristiche dipendono esclusivamente dalla struttura e non dalle proprietà intrinseche dei diversi materiali che li compongono. Essi sono costituiti unendo elementi che sono realizzati con materiali compositi differenti, come plastiche o metalli, in una struttura periodica.

Nella storia, però, si sente parlare per la prima volta del concetto di metamateriali molto prima del 1999, grazie al fisico russo Victor Veselago. Nel 1968 egli era convinto che fosse possibile ottenere dei materiali che presentassero costante dielettrica (ϵ) e permeabilità magnetica (μ) entrambe con valore negativo, facendosi sì che la luce si comportasse in modo innaturale, non rispettando più la legge di Snell. Veselago partì dalla propria intuizione per effettuare studi e ricerche relativamente all'elettrodinamica di questi nuovi materiali, in particolare voleva comprendere come si comportassero le onde elettromagnetiche, oltre a compiere studi sulle proprietà e le caratteristiche di tali nuovi materiali. Gli studi richiesero decenni prima di giungere a conclusioni valide e consolidate, infatti, solamente nel 1996 John B. Pendry propose, per la prima volta, la realizzazione di un metamateriale, composto da una matrice periodica di fili di rame, struttura che doveva garantire una permittività elettrica negativa. Da questa prima configurazione, negli anni successivi, ci fu un susseguirsi di nuovi tentativi di realizzazione, come per esempio nel 1999 Pendry propose una struttura composta da una serie periodica di risonatori ad anello diviso (Split Ring Resonator - SRR), con la quale egli riuscì a dare vita a un materiale che aveva esattamente le caratteristiche previste da Veselago 30 anni prima. Il primo metamateriale che venne effettivamente realizzato e utilizzato fu creato, nel 2000, intrecciando fili di rame con SRR, quindi unendo le due proposte avanzate da Pendry, esso mostrò una banda di frequenza con indice di rifrazione negativo. Nel 2002, poi, i metamateriali riuscirono ad ottenere un ulteriore spinta, in quanto vennero utilizzati dalla Lucent Technology per la produzione di antenne risonanti. Successivamente nel 2003 W. Ziolkowski, all'Università dell'Arizona, avanzò un'ulteriore ipotesi sulla composizione di questi materiali unendo un substrato di strisce caricate capacitivamente (Capacity Loaded Strips - CLS) e un substrato di risonatori ad anello quadrato (SRR). A partire da queste ricerche, effettuate da dei pionieri dei metamateriali, vennero effettuate numerose ricerche con l'obiettivo di evidenziare e studiare l'efficacia e il potenziale di tali materiali. Nel 2005 furono approfonditi per cercare ottenere caratteristiche elettromagnetiche particolari, che i materiali in natura non possedevano. Negli anni si è approfondito ulteriormente lo studio relativamente a questi

materiali fino a giungere nel 2017, dove un software ha reso possibile la realizzazione di antenne satellitari basate completamente ed esclusivamente sui metamateriali.

Ad oggi questi materiali sono conosciuti sotto diversi nomi, tra i quali si ricordano: mezzi ad onda regressiva (backward wave media), materiali mancini (left hand material), materiali a indice negativo o mezzi doppiamente negativi. Essi sono stati oggetto, nel corso degli anni, di diversi studi che tutt'ora vengono portati avanti, per essere in grado di scoprire tutte le possibilità che questi materiali offrono. Grazie, poi, alla manifattura additiva lo sviluppo dei metamateriali è fortemente accelerato.

1.1.2 Le caratteristiche e le proprietà

Come già citato nel paragrafo precedente l'elemento caratterizzante dei metamateriali è la loro struttura e non i materiali di cui sono composti. È, appunto, la struttura che gli garantisce di avere determinate caratteristiche e proprietà, che non possono essere riconosciute nei materiali che si trovano liberamente in natura. In particolare la geometria, la forma, la disposizione, l'orientamento e le dimensioni garantiscono a questi materiali di acquisire proprietà uniche, soprattutto nell'ambito delle onde elettromagnetiche. Essi, infatti, sono in grado di bloccare, assorbire o deviare le onde. Tra le proprietà più importanti che caratterizzano questi materiali, si può riconoscere certamente l'indice di rifrazione che, secondo la legge di Maxwell, è definito in funzione del prodotto dei due elementi cardine dei metamateriali:

$$n^2 = \epsilon\mu$$

Equazione 1: Equazione di Maxwell (Indice di rifrazione)

Con:

- n: indice di rifrazione
- ϵ: permittività elettrica (costante dielettrica), che rappresenta quanto facilmente gli elettroni riescono a muoversi nel materiale in base alla luce incidente su di esso
- μ: permeabilità magnetica del mezzo

Dall'intuizione di Veselago è noto che nei metamateriali sia la costante dielettrica, che la permeabilità magnetica abbiano valore negativo, di conseguenza, applicando la formula precedente, si determina un indice di rifrazione negativo, questo, secondo la legge di Snell, permette a questi materiali di riorientare le onde nell'ambiente nel quale si trovano, generando un indice di rifrazione superiore per l'onda. I materiali che si trovano in natura, invece, presentano un indice di rifrazione positivo, garantendo angoli di rifrazione minori per le onde. Quindi la grande dote dei metamateriali sta proprio nel fatto di riuscire a controllare l'andamento e l'orientamento delle onde, proprio grazie ai valori negativi di costante dielettrica e permeabilità magnetica. Questa è la proprietà cardine su cui si sono basati tutti gli studi effettuati fino ad oggi sui metamateriali. Affinché queste proprietà

possano essere sfruttate è necessario, però, che le geometrie ripetute, di cui sono formati, abbiano una dimensione inferiore rispetto alla lunghezza d'onda che si vuole andare a manipolare.

Oltre alle importanti proprietà elettromagnetiche, questi materiali garantiscono una forte flessibilità a livello progettuale, con proprietà che sono personalizzabili. Grazie a questo essi, attualmente, vengono utilizzati in diversi ambiti e per diversi scopi molto differenti tra loro.

1.1.3 Classificazione dei metamateriali

La classificazione dei metamateriali avviene in base ai valori assunti dalle costanti di: permeabilità magnetica e permittività elettrica. In particolare è possibile riconoscere 4 principali categorie:

1) Epsilon Negative Medium (ENG) nei quali la permittività elettrica è negativa, mentre la permeabilità magnetica è positiva. In questa categoria ricadono tutti i metalli nobili come oro e argento.
2) Double Positive Medium (DPS), in questa categoria i materiali presentano permittività elettrica e permeabilità magnetica positive. Essa è la categoria più popolata in cui ricadono la maggior parte dei materiali che si trovano in natura, come le sostanze dielettriche.
3) μ Negative Medium (MNG), in questa classificazione, invece, come suggerisce il nome stesso, la permeabilità magnetica presenta valore negativo, mentre la permittività elettrica ha valore positivo e ne fanno parte i mezzi girotropici.
4) Double Negative Medium (DNG), in quest'ultima categoria, invece, ricadono i metamateriali, in quanto presentano i valori di entrambe le costanti negativi.

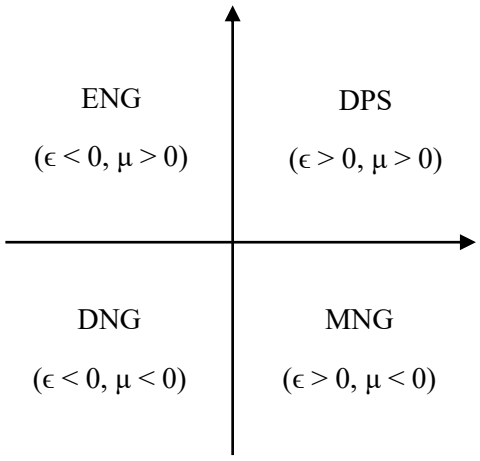


Figura 1: Classificazione dei metamateriali

Questa è la classificazione principale per la distinzione delle varie tipologie di materiali, a sua volta la classe dei metamateriali può essere ulteriormente divisa in base alle proprietà che, quello specifico materiale, riesce a modificare, come: proprietà termiche, acustiche o di trasporto, meccaniche o elettromagnetiche. Riuscendo così a riconoscere:

- Metamateriali acustici.
- Metamateriali elettromagnetici.
- Metamateriali meccanici o strutturali.
- Metamateriali elastici.

1.1.4 Contesti applicativi

Grazie alle grandi potenzialità delle caratteristiche dei metamateriali essi trovano applicazione in contesti molto differenti tra di loro, inoltre, a seguito dei numerosi studi che si stanno continuando ad effettuare su di essi nel corso degli anni, sono aumentati molto i diversi ambiti in cui essi vengono sfruttati, anche con scopi molto differenti tra di loro, cercando di esprimere al meglio le loro caratteristiche e proprietà. Proprio grazie a questa grande varietà di utilizzi il Grand View Reserch ha deciso di classificare il mercato mondiale dei metamateriali in base: all'utilizzo finale che ne verrà fatto, al tipo di prodotto, al paese e alle prospettive applicative.

Alcune delle applicazioni più importanti ad oggi sono:

- Settore medico, nei quali questi materiali vengono sfruttati all'interno di antenne per aumentare la potenza irradiata e migliorarne le caratteristiche e l'efficacia, garantendo un miglioramento della qualità di vita dei pazienti e una riduzione delle dimensioni dell'antenna. Esse presentano la caratteristica di riuscire, per la prima volta, a superare i limiti imposti di efficienza-larghezza della banda, riuscendo così ad essere utilizzate anche in spazi più ristretti. Questa tipologia di utilizzo non si limita solo al settore medico, ma si espande anche nell'uso per sensori di sorveglianza, collegamenti per la comunicazione, sistemi di navigazione e sistemi di comando e controllo.
- Settore automobilistico, in questo contesto, invece, i metamateriali vengono richiesti, soprattutto, per il funzionamento di motori elettrici e per il miglioramento della performance.
- Settore aerospaziale, vengono utilizzati per schermi, lenti, antenne e riduzione delle dimensioni di strutture di supporto. In questo contesto applicativo, però, ci sono limiti che devono essere rispettati, uno di questi, per esempio, è il peso, aumentandolo troppo la performance dell'aeromobile cala. Inoltre, viene spesso sfruttato per ridurre il rumore causato dal motore.
- Biosensori, utilizzati per il monitoraggio dell'ambiente e lo studio di fenomeni biologici, oltre all'analisi e alla diagnostica di malattie.
- Metamateriali sismici, sfruttati per contrastare gli effetti delle onde sismiche sulle strutture.
- Metamateriali acustici, vengono sfruttati per manipolare le onde sonore.

I metamateriali, in relazione con la formazione di bande di stop, risultano estremamente interessanti per l'isolamento vibro-acustico, che garantiscono grazie alle proprietà che hanno.

Queste sono solo alcune delle possibili applicazioni dei metamateriali, grazie, però, al forte sviluppo della scienza e al forte interesse verso questi materiali essi stanno sempre più prendendo piede dentro a diversi contesti, che puntano a sfruttare e massimizzare le loro caratteristiche e proprietà.

1.2 Struttura periodica

Come detto nella sezione precedente i metamateriali sono oggetto di numerosi studi e ricerche che si focalizzano, prevalentemente, sulla dispersione delle onde e la formazione di bande proibite, o band gap[1]. Questi ultimi due fenomeni dipendono da due meccanismi che sono: la periodicità, la quale dà origine a bande proibite di Bragg, e la risonanza locale, che permette di generare band gap che risultano regolabili tramite dei risonatori che sono incorporati nel mezzo, entrambi questi metodi, insieme al concetto di banda proibita, verranno discussi meglio in seguito. Oltre ai due processi precedenti esiste un'ulteriore possibilità che è l'accoppiamento di onde contradirezionali: tale metodologia permette di superare i limiti imposti dalle prime due tecniche citate. Anche quest'ultimo metodo verrà descritto meglio in seguito e sarà quello su cui si baserà principalmente la trattazione durante tutta la tesi.

Le strutture periodiche possono essere classificate in mono-, bi- oppure tridimensionali, e, inoltre, possono essere rappresentate da travi, barre, piastre oppure da gusci, nel corso della tesi verrà preso in considerazione il caso di travi piane piegate periodicamente.

1.2.1 La cella unitaria nelle travi piegate periodicamente

Prima di passare alla descrizione tecnica delle metodologie verrà effettuata una digressione riguardante le strutture periodiche dei metamateriali, concetto alla base della formazione delle bande proibite. Il fondamento della periodicità è la cella unitaria, infatti, le strutture periodiche vengono definite come un elemento base che si ripete uguale a sé stesso per tutta l'estensione della struttura. Le celle unitarie garantiscono un grande vantaggio per lo studio e l'analisi dei metamateriali, in quanto permettono di effettuare lo studio solo su una di esse ed estendere il comportamento a tutta la struttura, senza doverla analizzare tutta nei minimi dettagli. Nella prima figura che segue viene rappresentata una trave, basata sulla teoria di Eulero-Bernoulli, che viene piegata, a intervalli regolari, formando delle celle unitarie a forma di "V" che si ripetono, anche all'infinito, in entrambe le direzioni (folded periodic structure).

[1] Con il termine band gap viene indicato l'intervallo di frequenza nel quale le onde elettromagnetiche, a cui sono sottoposti i metamateriali, vengono bloccate e non possono più propagarsi all'interno del mezzo.

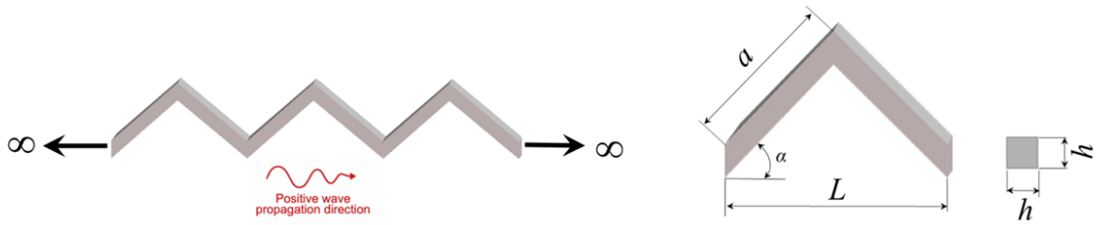


Figura 2:Rappresentazione della cella unitaria e della struttura periodica
Fonte: Manconi et al. (2026)

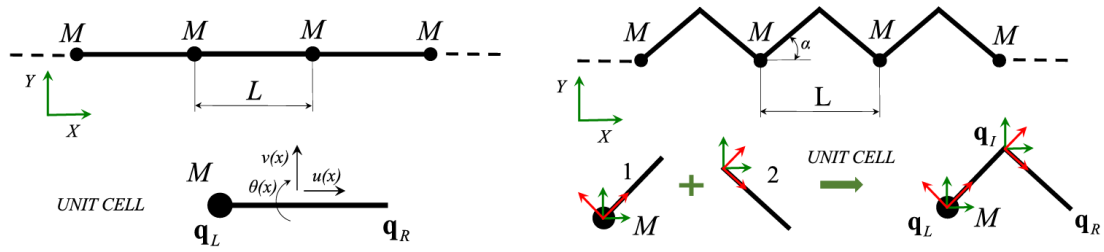


Figura 3: Rappresentazione di una configurazione accoppiata e disaccoppiata di una trave piegata periodicamente tramite metodo esatto DSM
Fonte: Manconi et al. (2026)

Oltre poi alla struttura ripetitiva sono indicati tutti i parametri geometrici di riferimento:

- α: angolo di piegatura della trave.
- L: lunghezza della cella unitaria periodica, quindi passo con cui la struttura si ripete.
- a: lunghezza del segmento di trave piegato, il quale risulta proporzionale a L e α.
- h: spessore della trave.

Nella seconda immagine, invece, è sempre rappresentata la trave piegata periodicamente tramite il metodo esatto della Dynamic Stiffness Matrix (DSM)[2], ma sono anche indicati i vettori:

- u(x): che rappresenta lo spostamento assiale, quindi descrive quanto la trave si allunga o si comprime lungo il suo stesso asse.
- v(x): che rappresenta lo spostamento trasversale, ossia il movimento perpendicolare della trave rispetto al proprio asse.

[2] Il Dynamic Stiffness Method (DSM) è un metodo che viene utilizzato per definire la propagazione delle onde elastiche all'interno di strutture periodiche. A differenza di altri metodi che effettuano la stessa cosa, esso garantisce di avere un'elevata accuratezza nella descrizione del comportamento della struttura in analisi, sottoposta a una frequenza che può essere attenuta o completamente inibita.

– θ(x): che rappresenta la rotazione della sezione.

A questi movimenti sono associati forze e momenti:

– U(x): rappresentante la forza assiale.
– V(x): rappresentante la forza di taglio.
– M(x): rappresentate il momento flettente.

Nella figura 3 sopra riportata, sono identificate due configurazioni differenti, che permettono di mettere in luce il concetto citato precedentemente di accoppiamento delle onde dovuto alla piegatura periodica. Nella prima immagine viene rappresentata una configurazione disaccoppiata, che corrisponde a una trave composta da celle unitarie, ma completamente dritta, nella quale, quindi, le onde non si accoppiano e non interagiscono tra loro, di conseguenza, la band gap non si forma. A fianco, invece, è rappresentata una trave piegata periodicamente di un angolo α, dove la cella unitaria può essere divisa tra una parte di sinistra ruotata di un angolo + α e una parte di destra ruotata di un angolo corrispondente a – α, questa configurazione, a differenza della prima, garantisce l'accoppiamento delle onde e la conseguente formazione di bande proibite. Questo è un concetto molto importante da sottolineare, perché è proprio la piegatura periodica che garantisce l'interazione di onde che altrimenti, in travi dritte, non interagirebbero mai.

Il progetto affrontato nel corso della tesi non si focalizzerà sull'aspetto tecnico e meccanico dei metamateriali, ma a titolo informativo, si vogliono citare alcune tecniche alla base del concetto di periodicità e, in particolare, per descrivere il comportamento dinamico della cella unitaria di una struttura periodica: il metodo della matrice di rigidezza dinamica (DSM), il quale si fonde alla teoria di Floquet-Bloch[3] che garantisce la periodicità e il metodo Wave Finite Element (WFE)[4] che permette di estendere i concetti di periodicità anche a strutture più complesse e tridimensionali.

1.3 Bande proibite nelle strutture periodiche

Come detto all'inizio della sezione precedente, la struttura periodica influenza la formazione di band gap, che rappresentano degli intervalli di frequenza nei quali le onde vengono attenuate o inibite completamente, quindi non si propagano più nel mezzo, questo è uno dei fenomeni più importanti e interessanti dei metamateriali, oggetto di

[3] Il teorema di Floquet-Bloch permette di imporre il comportamento periodico della struttura. Garantendo il fenomeno, descritto in precedenza, che permette di studiare il comportamento di una singola cella unitaria e generalizzare i risultati a tutta la struttura, evitando di dover analizzare ogni singola cella e permettendo di dividere le onde in evanescenti, propaganti o attenuate in base al valore assunto da μ nell'equazione di questa metodologia. Inoltre dove il valore di μ è diverso da zero è presente una band gap.

[4] Il Wave Finite Element (WFE) è un metodo che permette di ampliare il concetto di periodicità a strutture tridimensionali, quindi non limitandosi alla trave, che per semplicità è stata presa in considerazione in questa trattazione.

numerosi studi e ricerche che tutt'oggi vengono portate avanti, in quanto grazie alla struttura dei metamateriali è possibile controllare le onde elettromagnetiche.

Le band gap si possono distinguere in:

- Parziali, in questo caso solo una famiglia di coppie di modi d'onda viene bloccato, e quindi dimostra un fenomeno di locking, mentre tutte le restanti onde elettromagnetiche continuano a propagarsi senza difficoltà.
- Complete, a differenza del caso precedente e come suggerisce il nome stesso, in questo caso tutte le onde vengono bloccate, ciò avviene nel momento in cui vi è un accoppiamento sia delle onde longitudinali-flessionali, che delle onde torsionali-flessionali, di conseguenza tutte le vibrazioni assiali, flessionali e torsionali vengono attenuate senza distinzione.

Nel corso della tesi prenderemo in considerazione le bande proibite complete, che permettono di ottenere una totale inibizione dell'onda a cui il materiale è sottoposto.

1.3.1 Meccanismi di formazione delle band gap

La formazione di band gap dipende da alcuni principali meccanismi, che sono:

- Bragg scattering, questo meccanismo si origina dalla periodicità della struttura, le bande proibite di Bragg si verificano quando la lunghezza d'onda diventa paragonabile alla spaziatura tra le celle unitarie, quindi il punto di inizio della banda dipende da questa relazione. Queste band gap, solitamente, si verificano ad alte frequenze, limitandone l'uso in caso di onde a bassa frequenza, infatti, una diretta conseguenza di questo metodo è che: più piccola è la lunghezza dell'onda elettromagnetica, più alta sarà la frequenza dell'onda stessa, ciò comporta che una spaziatura minore tra le celle unitarie della struttura periodica dia origine a una frequenza di apertura della band gap maggiore, ossia il limite citato precedentemente, in quanto se si volessero ottenere bande di stop a basse frequenze, bisognerebbe avere delle celle unitarie molto grandi.

- Risonanza locale, a differenza del metodo precedente, in questo caso, le bande proibite si originano inserendo dei risonatori all'interno del mezzo in analisi, inoltre, in questo caso, può essere presente, o meno, la periodicità della struttura. Solitamente in questo modello la frequenza di apertura della band gap si verifica in prossimità della frequenza di risonanza del singolo risonatore in analisi, è possibile, quindi, superare la limitazione imposta dal Bragg scattering riguardante le basse frequenze, aggiungendo masse risonanti o delle interfacce multi-materiale molto più complesse, che comportano una maggiore complessità di fabbricazione, riducendone, però, l'efficienza strutturale.

- Accoppiamento tra i modi d'onda (wave coupling), quest'ultimo meccanismo, che va oltre metodi classici e permette di superare i limiti dei due modelli precedenti,

permette di dare origine a fenomeni di locking, ossia di bloccaggio della propagazione delle onde elettro magnetiche nel mezzo, o di veering, quindi di deviazione, grazie all'accoppiamento dei modi d'onda. Tramite questo meccanismo le band gap si originano quando viene unito il concetto di periodicità della struttura in analisi e la presenza di onde che hanno velocità opposte, quest'ultime interagendo intrinsecamente tra loro creano delle regioni di energia stazionaria ridotta che permettono di formare delle bande proibite complete. In particolare l'accoppiamento dei modi d'onda può avvenire quando le velocità delle onde che interagiscono tra loro hanno lo stesso segno, originando delle deviazioni, oppure, nel caso in cui abbiano segno opposto, dando origine a un vero e proprio bloccaggio della propagazione delle onde. Trainiti et al. ha effettuato uno studio che ha permesso di dimostrare che la curvatura di travi che vengono piegate periodicamente, permette l'accoppiamento di modi d'onda longitudinali[5], flessionali[6] e torsionali[7], che creano delle bande di stop la cui posizione e dimensione dipende dalle caratteristiche geometriche e strutturali della piegatura. In conclusioni le strutture piegate periodicamente, come le Miura-ori, sono l'elemento perfetto per studiare questo metodo.

1.3.2 Le curve di dispersione per l'analisi grafica delle band gap

Le curve di dispersione permettono di rappresentare la relazione che sussiste tra la frequenza dell'onda elettromagnetica presa in analisi e il suo numero d'onda complesso μ. In questo specifico caso viene utilizzato un numero complesso per riuscire a rappresentare il fenomeno della perdita di energia durante la fase di avanzamento dell'onda stessa, in questo numero complesso la parte reale rappresenta la costante di fase e indica quanto oscilla l'onda, quindi permette di misurare quanto è compressa, o quanto è distesa, mentre la parte immaginaria che esprime la costante di attenuazione, permette di rappresentare la velocità con cui l'onda perde la sua ampiezza durante la fase di avanzamento nel mezzo, quindi è una misura di quanto viene smorzata.

Il valore di μ può essere:

- Reale, ossia il caso in cui la parte immaginaria è uguale a zero, rappresenta il caso in cui le onde oscillano e avanzano, senza però essere attenuate, in questo caso sono definite onde propaganti.
- Immaginario, in questo caso è presente solo la parte immaginaria, senza la presenza di una parte reale, ciò si traduce nel fatto che l'onda non oscilla, ma si

[5] Con il termine onde longitudinali si prendono in considerazione tutte quelle onde che hanno un moto descritto da una fase di allungamento seguita da una fase di compressione, quindi lo spostamento avviene lungo l'asse di lunghezza principale.
[6] Con il termine onde flessionali, si rappresentano tutte quelle onde che hanno un andamento determinato da delle flessioni o dei piegamenti, quindi ove presente una deformazione trasversale con moto perpendicolare alla direzione di propagazione dell'onda.
[7] Come suggerisce il nome stesso, all'interno del gruppo delle onde torsionali ricadono tutti quei moti d'onda che hanno un avanzamento basato sulla torsione.

attenua molto rapidamente, fino al punto in cui si annulla completamente, in questa casistica rientrano le onde evanescenti.
- Complesso, in questo caso è presente sia una parte immaginaria che una parte reale, ciò si traduce nel fatto che l'onda in esame oscilla, ma si attenua anche, quindi si propaga e, mentre lo fa, perde energia. In questa classificazione le onde vengono denominate attenuate.

Nell'analisi delle band gap è fondamentale che il valore del numero d'onda sia immaginario, o complesso, solo in questo modo si verificherà l'attenuazione e non vi sarà alcun trasporto di energia vibrazionale, di conseguenza si presenteranno zone in cui la frequenza sarà completamente bloccata.

Le curve di dispersione si differenziano nel caso in cui si analizzi un sistema disaccoppiato, quindi una trave dritta in cui le onde non interagiscono tra di loro, oppure un sistema accoppiato, quindi una trave piegata di un angolo α, in cui le onde interagiscono tra loro e, per il modello dell'accoppiamento delle onde, si attenuano o si bloccano vicendevolmente.

In particolare è possibile identificare due modi d'onda principali:

- Modi d'onda longitudinali.
- Modi d'onda flessionali.

Essi, nel caso disaccoppiato, come suggerisce il nome stesso, non interagiscono tra loro, si incrociano nei punti critici P e Z, ma non hanno effetto gli uni sugli altri. Al contrario, nel momento in cui viene introdotta una piegatura, e, quindi, si entra nel caso accoppiato, esse interagiscono tra di loro dando origine a due fenomeni citati anche in precedenza:

- Veering, nei pressi del punto Z, in cui le onde con modi longitudinali e flessionale hanno stesso segno di velocità e si respingono a vicenda, attenuandosi.
- Locking, nei pressi del punto P, in questo caso, invece, le onde hanno segni opposti di velocità e si bloccano tra loro, creando una banda di bloccaggio nata grazie al meccanismo di accoppiamento delle onde citato nella sottosezione precedente.

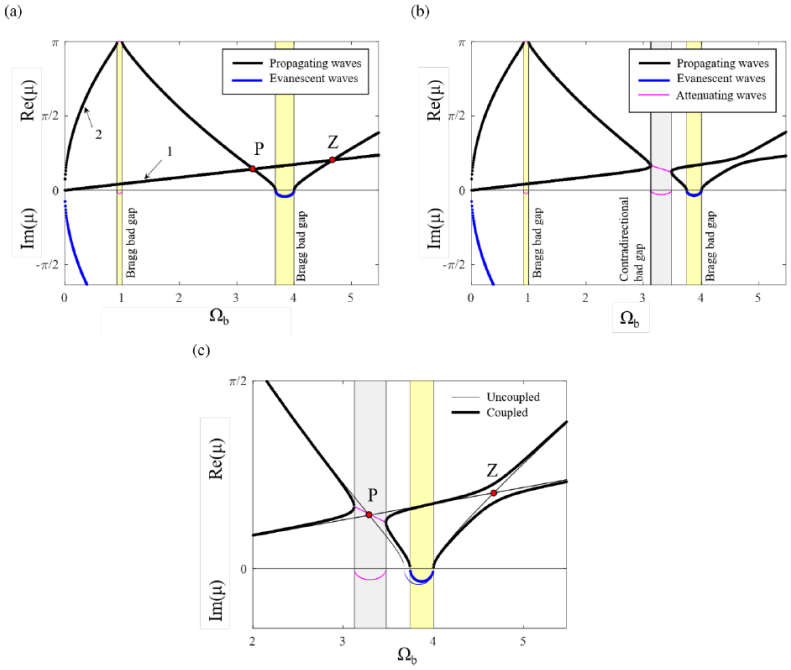


Figura 4: Curve di dispersione per la rappresentazione grafica delle bande di stop
Fonte: Manconi et al. (2026)

Quanto presentato fino ad ora viene rappresentato nella figura 4 nella quale:

- Il primo grafico rappresenta il caso di disaccoppiamento in cui l'angolo di piegatura α è pari a zero e non vi è la formazione di band gap.
- Nel secondo grafico, invece, è rappresentato un sistema di accoppiamento, nel quale, le onde interagiscono tra loro generando una banda di stop rappresentata dalla striscia grigia.

In entrambi i grafici, poi, viene sottolineato il comportamento delle onde propaganti, evanescenti e, nella seconda immagine, anche delle onde attenuanti.

- Nel terzo grafico, infine, viene messo in evidenza il comportamento delle onde che interagiscono tra loro nei punti P e Z, mettendo in luce, in particolare, come nel punto P esse si blocchino completamente (fenomeno di locking), generando una banda proibita rappresentata dalla striscia grigia, in quanto le onde si avvicinano con velocità e pendenze di segno opposto, mentre nel punto Z le onde si attenuano (fenomeno di veering), siccome hanno pendenza e velocità dello stesso segno, quindi si avvicinano e poi si respingono, senza però generare alcuna banda di stop.

In tutti i grafici, poi, in giallo è rappresentata la band gap generata da Bragg scattering.

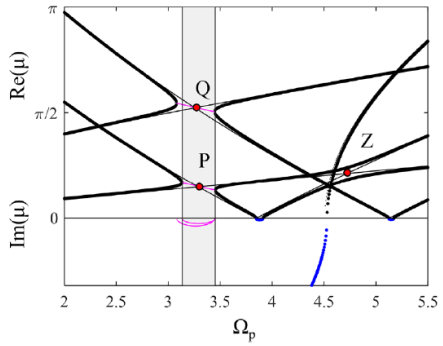


Figura 5: Grafico di dispersione di una band gap completa
Fonte: Manconi et al. (2026)

I grafici di dispersione che rappresentano le bande proibite complete sono leggermente diversi, in quanto vengono rappresentati due punti P e Q di locking. Come citato all'inizio della sezione si parla di band gap complete nel momento in cui c'è un accoppiamento dei modi d'onda completo e, quindi, tramite i due punti è possibile evidenziare questa caratteristica, nello specifico il punto P rappresenta l'accoppiamento longitudinale-flessionale, mentre il secondo quello torsionale-flessionale. Essi, interagendo tra loro, garantiscono la formazione di bande di stop complete.

1.3.3 Analisi parametrica per la formazione di band gap

La struttura periodica è dotata di parametri, la cui variazione può determinare un cambiamento nella banda di stop completa, modificandone il livello di attenuazione. Per studiare l'effetto del cambiamento dei parametri geometrici della trave piegata periodicamente sulla band gap verranno analizzate le curve di dispersione, utilizzando, in particolare come strumenti:

- Le superfici a costante di attenuazione,
- Le superfici di bordo banda.

Il primo valore che può essere fatto variare è l'angolo di piegatura α. Nel momento in cui viene fatto variare questo parametro si presentano due possibilità diverse:

1) Lunghezza del segmento costante e lunghezza periodica variabile, in questo caso l'attenuazione, e, quindi, la band gap, cresce all'aumentare dell'angolo di piegatura.

2) Lunghezza del segmento variabile e lunghezza periodica costante, in questo caso, invece, il comportamento è molto più complesso, in quanto la band gap aumenta fino a un'ampiezza dell'angolo pari ai 26°, poi incomincia a diminuire, mentre la costante di attenuazione, inversamente dalla larghezza della banda di stop, continua ad aumentare. Inoltre si nota un altro comportamento insolito con la

nascita di bande proibite a frequenze molto più alte dovute a dei cambiamenti della massa e della cella unitaria di cui è composta la struttura periodica.

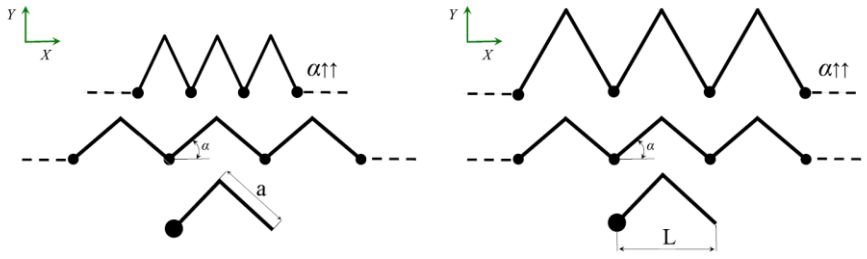


Figura 6: Rappresentazione grafica delle due possibilità analizzate per la variazione di α
Fonte: Manconi et al. (2026)

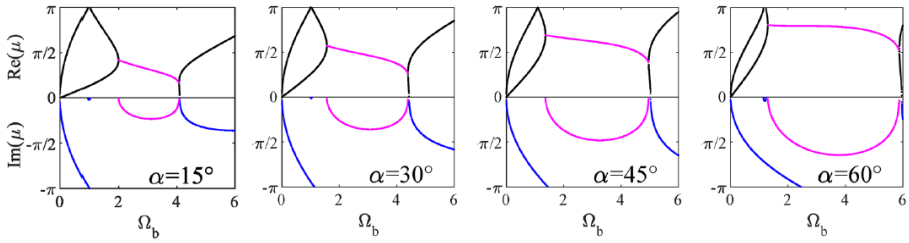


Figura 7: Grafici di dispersione al variare dell'angolo di piegatura
Fonte: Manconi et al. (2026)

Oltre all'angolo di piegatura, possono essere fatti variare anche altri due parametri fondamentali della struttura periodica, che sono:

- Lo spessore finito, aumentando questo valore, l'ampiezza della band proibita aumenta.

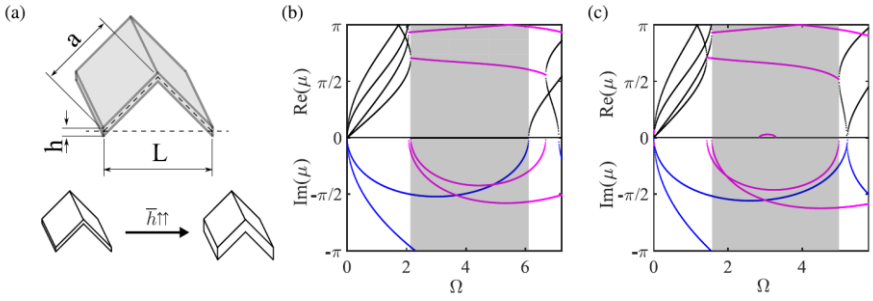


Figura 8: Grafici di dispersione rappresentanti le band gap al variare dello spessore della cella unitaria
Fonte: Manconi et al. (2026)

– La curvatura della piega, in questo caso, invece, i valori di inizio e di fine della band gap, si spostano verso frequenze sempre maggiori, ma questo non comporta un cambiamento nella larghezza della banda, che rimane pressoché identica.

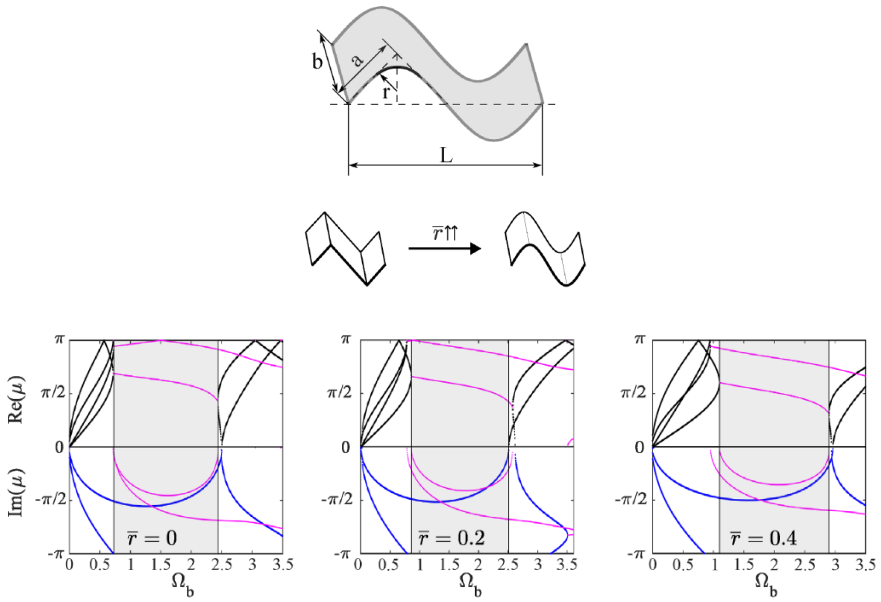


Figura 9: Curve di dispersione rappresentanti le band gap al variare della curvatura della piega periodica
Fonte: Manconi et al. (2026)

CAPITOLO 2: Machine Learning

2.1 Introduzione al concetto di machine learning

Uno degli argomenti più discussi, studiati e affrontati negli ultimi anni è, indubbiamente, l'intelligenza artificiale (Artificial Intelligence – AI). Al giorno d'oggi in qualsiasi attività quotidiana svolta da una persona si è in grado di riconosce l'intervento dell'AI.

Il concetto di intelligenza artificiale venne introdotto per la prima volta nel 1956, quando un gruppo di scienziati iniziò a pensare che il computer potesse essere in grado di pensare e ragionare, dopo essere stato programmato per farlo, essi, di conseguenza, chiamarono questa caratteristica proprio "Artificial Intelligence". L'intelligenza artificiale nacque con l'idea di voler riprodurre il comportamento del cervello umano tramite la programmazione di un calcolatore. Quindi, si basa, tutt'oggi, sul concetto di tentare di voler unire due mondi che fino a qualche anno fa erano considerati due rette parallele che mai si sarebbero potute incontrate, ossia: la capacità di pensiero e analisi della mente umana, con la grande potenza di calcolo dei computer. Questa idea nasce dal pensiero che per quanto la mente umana sia in grado di memorizzare, analizzare e studiare informazioni, le sue prestazioni non saranno mai comparabili a quelle di un calcolatore programmato per fare quel determinato compito, per esempio in termini di memorizzazione di dati e informazioni i computer sono in grado di analizzare milioni di valori in pochissimi secondi, cosa che per l'uomo è impossibile. L'AI, quindi, punta all'automatizzazione di attività che normalmente sarebbero svolte dalle persone quotidianamente. Essa risulta, quindi, molto performante nella risoluzione di problemi che sono ben definiti, chiari e specifici, mentre per problemi di livello più alto, nei quali vi è la necessità di apprendimento e di riconoscimento di pattern complessi, spesso gli algoritmi di intelligenza artificiale tendono a fallire ed è proprio in questo campo che entrano in gioco i concetti di Machine Learning (ML) e Deep Learning (DL), i quali permettono di ottenere performance superiori.

Spesso AI, ML e DL vengono, erroneamente, utilizzati per descrivere lo stesso concetto, ma questo non è propriamente corretto, essi risultano essere fortemente associati, ma non interscambiabili tra loro, infatti, il Machine Learning e il Deep Learning sono delle sotto-branche dell'intelligenza artificiale. In particolare, il Machine Learning nel corso degli anni è stato definito da diversi studiosi e ricercatori in molti modi differenti, ma tutti sono accomunati da uno stesso concetto di fondo, ossia che esso comprenda tutti i metodi, algoritmi e processi che permettono di svolgere le attività citate in precedenza per l'AI. A differenza di quest'ultima, infatti, il machine learning comprende un concetto cardine, da cui deriva anche il suo nome, ossia learning (apprendimento), quindi esso permette di realizzare sistemi che sono in grado di imparare autonomamente in base ai dati che ricevono, modificandosi in base alle informazioni che vengono analizzate ed elaborate, cercando di adattarsi sempre meglio al problema affrontato e migliorandosi continuamente. Dall'altra parte il DL, invece, è una sotto branca del ML, che permette di risolvere problemi ancora più complessi grazie all'utilizzo di reti neurali, che verranno affrontate meglio in seguito.

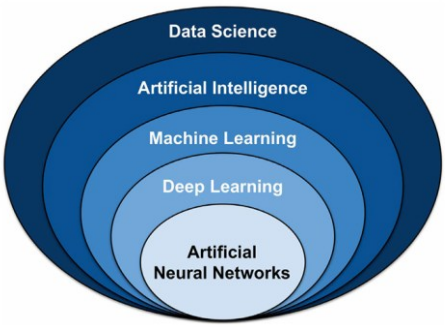


Figura 10: Analisi grafica delle relazioni tra i concetti di Data Science, AI, ML, DL e reti neurali
Fonte: Choi et al. (2020)

Il machine learning oggi è riconosciuto come una delle più importanti scoperte della quarta rivoluzione industriale.

2.1.1 Modelli di apprendimento

Il concetto di ML può essere suddiviso in quattro categorie principali:

1) Apprendimento supervisionato (supervised learning), in questa tipologia di apprendimento il modello impara da coppie di input-output che sono note e gli sono fornite direttamente, sulla base di queste coppie, il modello è in grado di dedurre informazioni e mappare funzioni. Solitamente la procedura che viene seguita è:

   – Si ricava un dataset.

   – Si divide il set di dati generato in training, validation e test set. I primi due vengono usati per addestrare il modello sulla relazione input-output, mentre l'ultimo viene utilizzato per valutare la performance.

   – Infine, avviene la vera e propria fase di addestramento, durante la quale il modello è in grado di apprendere il compito che gli è stato assegnato dall'intervento umano.

   I due esempi più classici di supervised learning sono la regressione, che permette di prevedere dati, modelli o informazioni, e la classificazione che, invece, permette di definire a quale classe appartengono determinati valori, dividendoli in base a un criterio definito. Entrambe le tipologie di problemi citati, saranno descritti meglio nelle prossime sezioni.

2) Apprendimento non supervisionato (unsupervised learning), nel quale, a differenza del precedente, viene lasciata maggiore libertà alla macchina di ricercare un pattern tra i dati che gli sono forniti, dividendoli in diverse classi in base a quanto riscontrato. Nell'apprendimento non supervisionato, quindi, non c'è

alcun intervento umano, ma il modello agisce completamente da solo. Alcuni degli esempi più classici di unsupervised learning sono il clustering che, come suggerisce il nome stesso, permette di dividere in cluster i dati forniti in base a determinate loro caratteristiche, la riduzione della dimensione dei dati e anomaly detection.

3) Apprendimento semi-supervisionato (semi- supervised learning), il quale, invece, si trova a metà tra supervised e unsupervised learning. Spesso è raro essere in possesso di una quantità sufficiente di dati che siano già stati etichettati come input-output, quindi, valori per i quali è stato riconosciuto precedentemente il pattern che si vuole insegnare al modello di ML e, al tempo stesso, i dati non etichettati, sono molto più numerosi e frequenti, per ovviare a questo problema entra in gioco questa tipologia di apprendimento misto. L'obiettivo su cui si fonda è di riuscire ad ottenere risultati migliori rispetto a quelli che si otterrebbero utilizzando solo i pochi dati etichettati a disposizione. Questo modello viene spesso utilizzato per l'imagining medico, l'tichettatura dei dati e la fraud detection.

4) Apprendimento rinforzato (reinforcement learning), questo tipo di apprendimento è completamente differente rispetto a quelli descritti fino ad ora. Esso è un algoritmo che si basa sul fatto che il modello effettui la scelta ottima, migliorandosi in base agli stimoli che riceve, l'algoritmo, infatti, viene premiato nel caso svolga correttamente il proprio compito, in alternativa gli viene aggiunta una penalità, nel caso in cui commetta degli errori. L'algoritmo, quindi, tenderà a voler massimizzare il reward, ossia il premio ricevuto, cercando di minimizzare gli errori commessi e massimizzando l'efficienza. Questo metodo è spesso utilizzato per addestrare modelli di AI, più che di ML.

Il tipo di algoritmo di ML che viene utilizzato dipende dalla natura e dalle caratteristiche dei dati che si hanno a disposizione, infatti, in base ad essi e in base al compito che deve essere svolto deve essere scelto l'algoritmo più corretto per avere i migliori risultati.

2.1.2 Principali problemi in cui viene utilizzato il Machine Learning

A titolo informativo verrà fatto un elenco di alcuni dei principali problemi affrontati tramite l'utilizzo del ML:

- Classificazione, come accennato in precedenza, questa tipologia di problemi si basano su modelli che si occupano dell'assegnazione di una determinata classe ai dati di input forniti. Uno degli esempi più classici e con cui ci confrontiamo tutti i giorni è l'algoritmo anti-spam della casella di posta elettronica, esso è in grado di classificare le email ricevute in: "Spam" e "Non spam", permettendo alla casella di posta di mostrare solo le mail che sono veritiere e affidabili. Esistono varie tipologie di problemi di classificazione: binario, nella quale esistono solo due classi, multi-class nel quale esistono più classi in cui i dati possono essere etichettati e, infine, multi-label in cui i dati possono essere classificati all'interno di più di una classe. Il processo di classificazione si basa su due fasi principali: la prima è quella di addestramento, nella quale vengono forniti all'algoritmo dei dati

già divisi in classi, i quali possiedono diverse caratteristiche e pattern che il modello deve riconoscere e imparare, successivamente avviene una fase di test, nella quale sono forniti all'algoritmo dati che non sono ancora etichettati e sarà compito del modello dover assegnare a ciascuno di loro la classe corretta.

- Regressione, in questa tipologia di problema, invece, l'obiettivo è quello di predire valori continui. Anche in questo caso è presente una prima fase di addestramento, durante la quale vengono forniti al modello dei dati di input, dei quali si conosce già il valore che dovrà essere predetto dall'algoritmo. A seguire avviene una fase di test, durante la quale vengono forniti all'algoritmo dei dati nuovi dei quali dovrà predirne il valore di output, in base a come è stato addestrato durante la prima fase. Anche in questo caso possono essere riconosciute diverse tipologie di problemi, infatti, esistono i: simple and multiple linear regression problem nei quali la variabile dipendente è continua, mentre quella indipendente può essere continua, o discreta, e la relazione tra queste due variabili è lineare, e i polynomial regression problem nei quali la relazione tra la variabile dipendente e indipendente è rappresentata da un polinomio e non più da una relazione lineare come nel caso precedente. Il problema di regressione si adatta molto bene a diverse applicazioni come la stima di costi aziendali, le serie storiche e le predizioni finanziarie.

Nella figura seguente viene rappresentata graficamente la differenza tra classificazione e regressione:

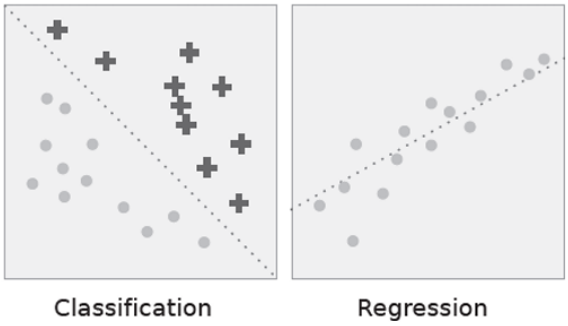


Figura 11: Problemi di classificazione a confronto con problemi di regressione
Fonte: Sarker et al. (2021)

Come si nota nei problemi di classificazione l'obiettivo è quello di suddividere i dati in classi definite a priori, i cui confini, in questo caso, sono rappresentati dalla linea tratteggiata, mentre nei problemi di regressione l'obiettivo è la predizione di valori e la linea tratteggiata, differentemente dal caso precedente, rappresenta la relazione che intercorre tra variabili dipendenti e indipendenti, ossia la relazione ottima che il modello deve cercare di imparare.

- Clustering, in questa tipologia di problemi l'obiettivo è di riuscire a dividere i valori forniti all'algoritmo in cluster, in base alle caratteristiche che i dati di input possiedono e che vengono riconosciute direttamente dal modello. A differenza della classificazione, in cui le classi sono conosciute a priori, nella clusterizzazione è compito dell'algoritmo riconoscere le caratteristiche comuni tra i dati e raggrupparli insieme. È possibile intuire come sia fondamentale la fase di preparazione dei dati che vengono forniti agli algoritmi di classificazione e clusterizzazione. Questa attività, spesso molto sottovalutata, è essenziale affinché l'algoritmo svolga, nel migliore dei modi, il proprio compito. Esistono diversi meccanismi attraverso i quali può essere effettuata la divisione in cluster, per esempio sulla base di: similarità, gerarchie, in base alla densità dei dati, oppure grid-based o model-based method. Questa tipologia di problemi viene sfruttata, ad esempio, per la suddivisione dei clienti all'interno di negozi e e-commerce, oppure, viene utilizzato anche all'interno dei social network. Uno degli utilizzi più importanti, in cui questo problema di clusterizzazione viene sfruttato, è sicuramente il riconoscimento di outliner, quindi per l'identificazione di dati che sono presenti all'interno del dataset fornito all'algoritmo, ma hanno caratteristiche completamente diverse, qui ritorna, nuovamente, il concetto di importanza nella fase di pulizia e organizzazione del set di valori forniti all'algoritmo.

- Associazione, questa tipologia di problemi, invece, si focalizzano sulla ricerca di relazioni di tipo IF-THEN, un classico esempio è "Il cliente che ha comprato i cereali ha anche acquistato il latte". Come è facile intuire, anche questa tipologia di problema, viene sfruttato in tanti contesti differenti: come l'utilizzo di servizi Internet of Things (IoT), oppure per le diagnosi mediche o la cybersecurity.

- Riduzione della dimensione dei dati, come citato anche in precedenza, l'accuratezza e la precisione dei dati è fondamentale nell'ambito del ML, spesso, però, si ha a che fare con un'enorme quantità di valori che sono difficilmente gestibili e che portano a fenomeni come quello dell'overfitting, che verrà meglio definito in seguito. Riuscire, quindi, a diminuire la dimensione dei dati semplificando il problema e il modello diventa fondamentale per aumentare l'efficienza, l'efficacia dell'algoritmo e l'accuratezza dei risultati.

Come ben si capisce, quindi, gli algoritmi di machine learning possono essere applicati a tantissimi problemi differenti, da questo si deduce la loro grandissima importanza e applicabilità.

2.2 I principali algoritmi di Machine Learning

In questa sezione, invece, verrà fatto un breve elenco dei principali e più importanti algoritmi di machine learning, non verranno citati tutti, ma solamente quelli che ad oggi vengono utilizzati maggiormente:

- Regressione lineare, questo è sicuramente uno degli algoritmi di ML più semplici e intuitivi da andare ad applicare, motivo per il quale è largamente utilizzato, ma,

al tempo stesso, a causa della sua estrema semplicità non riesce a essere esaustivo per alcuni problemi più complessi. Esso si basa sulla relazione tra una variabile dipendente e una variabile indipendente e viene utilizzato per risolvere problemi di regressione.

La relazione tra variabili dipendenti e indipendenti può essere descritta, per esempio, tramite la seguente funzione:

$$y = ax + b$$

Equazione 2: Equazione retta problemi di regressione lineare

Dove:

- a: è la pendenza della retta.
- b: è l'intercetta della retta con l'asse delle ordinate.

L'obiettivo dell'algoritmo è cercare di identificare i valori a e b in modo tale da approssimare, nel migliore dei modi, i dati del dataset, eventualmente, è possibile aggiungere pesi differenti a ciascun valore. La variabile dipendente è continua, mentre quella indipendente può essere continua oppure discreta.

- Regressione logistica, questo è un altro algoritmo statistico di ML utilizzato per risolvere, principalmente, problemi di classificazione, nonostante il nome, ma in alcuni contesti può essere applicato anche a problemi di regressione. Applicando questa tipologia di algoritmo l'obiettivo è quello di riconoscere una relazione tra alcune caratteristiche e la probabilità di ottenere un determinato risultato. A differenza del caso precedente per stimare questa probabilità non viene utilizzata l'equazione di una retta, ma piuttosto una curva, descritta da una Sigmoide (o funzione logistica, da cui prende il nome), che ha la seguente equazione:

$$y = \frac{1}{1 + e^{-x}}$$

Equazione 3: Equazione sigmoide

Questa equazione permette di generare una curva ad S, la quale è in grado di convertire dei valori discreti, o continui, ossia i dati di input che sono forniti all'algoritmo, in un unico valore compreso tra 0 e 1, generando, così, la probabilità citata in precedenza. Questo è, sicuramente, il caso più semplice, può, poi, essere complicato introducendo la possibilità di avere più risultati differenti per ciascun valore.
Questa tipologia di algoritmo è fortemente soggetto a overfitting, nel caso in cui gli venga fornito un dataset di dimensioni elevate, mentre tende a lavorare meglio, ottenendo risultati migliori, nel caso in cui i dati di input siano separabili

linearmente. Questa caratteristica di linearità tra variabili dipendenti e indipendenti, è sicuramente uno degli svantaggi maggiori di questa tecnica, che ne limita l'utilizzo ai soli problemi lineari.

- K-Nearest Neighbors (KNN), anche questo algoritmo può essere utilizzato sia per problemi di regressione che per problemi di classificazione. L'obiettivo è di classificare i dati che gli vengono forniti in base a delle misure di prossimità e similarità tra i valori che sono già stati classificati. Questo algoritmo spesso viene definito lazy learning in quanto non si occupa di creare un vero e proprio modello matematico, ma, piuttosto, memorizza solamente i dati. Sicuramente l'elemento più difficile da definire per questo algoritmo è: il numero di vicini K da andare a considerare per prendere una decisione riguardo a quale classe assegnare il valore in analisi.

- Decision Tree, questa tipologia di algoritmo fa parte dei modelli di apprendimento supervisionato e prende il nome dalla forma del grafico che si costruisce quando viene utilizzato, ossia un classico albero decisionale. Questo algoritmo viene, spesso, utilizzato in numerosi contesti differenti sia per problemi di classificazione che di regressione. Il grafico risulta formato da nodi, che sono rappresentati da cerchi, i quali identificano il punto dove i dati subiscono una divisione e, quindi, dove viene effettuata una scelta, o una predizione. In particolare il primo nodo da cui si parte viene definito root (o radice), mentre i nodi terminali, alla fine del grafico, vengono definiti leaf (foglie). Sono, inoltre, presenti gli archi, che sono segmenti che collegano i nodi. In particolare, ogni nodo rappresenta una domanda che viene posta e gli archi, collegati al nodo, sono le possibili risposte al quesito, generate in base alle miglior features per la divisione dei dati. Quindi, dal nodo radice, si prosegue effettuando domande e generando risposte per dividere i dati in base a caratteristiche comuni, finché non si è coperto tutto il dataset e non si arriva ai nodi foglia che rappresentano la predizione finale dell'algoritmo. È importante sottolineare che l'ordine in cui vengono effettuate le domande è fondamentale, in quanto, in base ai nodi che vengono generati, possono essere definiti alberi decisionali anche molto differenti tra loro. Questo risulta essere uno degli algoritmi più semplici da implementare, motivo per il quale è largamente utilizzato, ma uno dei rischi principali è sicuramente l'overfitting che esso genera, per minimizzare questo effetto è importante controllare la profondità dell'albero, quindi il numero di livelli che si generano.

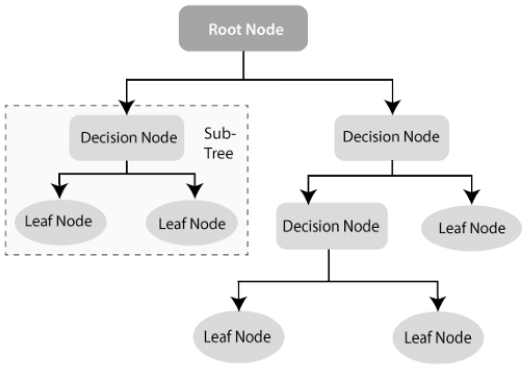


Figura 12: Esempio di Decision Tree
Fonte: Sarker et al. (2021)

- Random Forest, questa tipologia di algoritmo, parte direttamente dagli alberi decisionali, come suggerisce il nome stesso, infatti, esso è composto da diversi decision tree che vengono uniti tra di loro per formare un unico algoritmo, quindi vengono presi gli output di diversi alberi decisionali e si definisce un unico output finale, che può essere, per problemi di classificazione, la classe più numerosa e più frequente, mentre per problemi di regressione, l'output finale risulta essere il valore medio dei risultati predetti dall'algoritmo. Uno degli elementi che si deve tenere in considerazione e monitorato è sicuramente il numero di alberi decisionali che si decide di inserire all'interno della random forest. Spesso, si preferisce questo algoritmo al precedente in quanto è più semplice da implementare e meno soggetto al fenomeno dell'overfitting e il valore di quest'ultimo aumenta solo nel caso in cui gli alberi decisionali che si utilizzano per formare la random forest hanno numerosi valori, o classi, che non sono state predette correttamente. Questa tipologia di algoritmo, quindi, si adatta sia a problemi di classificazione, che di regressione, prendendo in considerazione ed utilizzando sia valori continui che categorici.

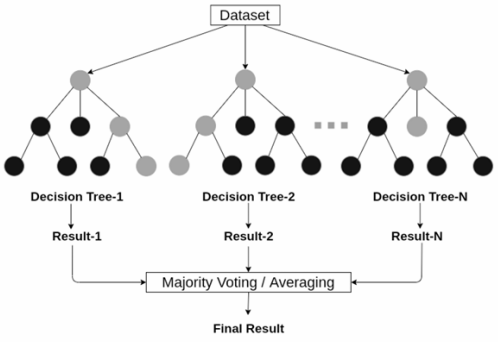


Figura 13: Esempio di algoritmo di Random Forest
Fonte: Sarker et al. (2021)

- K-means Clustering, come suggerisce il nome stesso, si tratta di un algoritmo che viene utilizzato per affrontare e risolvere problemi di clusterizzazione, esso risulta essere molto semplice e robusto permettendo di dimostrare se i dati forniti in input sono stati correttamente separati tra loro. Per utilizzare questo algoritmo bisogna partire definendo k centroidi[8], in maniera completamente casuale, ottenendo così una soluzione poco significativa inizialmente. L'obiettivo è quello di assegnare i punti ai centroidi in maniera tale da minimizzare la distanza quadrata tra i punti assegnati a quel centroide e il centroide stesso, man mano che i dati vengono assegnati si ha uno spostamento del centroide, fino ad ottenere una soluzione più consistente e veritiera. Definito il procedimento di utilizzo di questo algoritmo, si intuisce come esso sia fortemente sensibile a eventuali outliners presenti nel dataset, i quali comportano un importante spostamento della posizione dei centroidi, modificando significativamente la soluzione finale.

Questi sono solo alcuni dei principali e più utilizzati algoritmi di ML, ma non vuole essere, sicuramente, una lista esaustiva. A questi, per esempio, possono essere aggiunte le reti neurali, che verranno affrontate in maniera più approfondita nella prossima sezione.

2.3 Introduzione al concetto di reti neurali

In questa sezione verrà affrontato più nel dettaglio il concetto di Neural Network, altro algoritmo di ML cardine e largamente utilizzato.

Come accennato all'inizio del capitolo 2, l'obiettivo principale del ML è quello di riuscire a riprodurre, nella maniera più fedele possibile, il funzionamento del cervello umano, così da riuscire a sfruttare in modo congiunto le grandi potenzialità di un calcolatore e le capacità cognitive del cervello umano. Uno degli algoritmi che meglio riproduce il modo di pensare delle persone è, sicuramente, la rete neurale. Bisogna, quindi, fare un breve accenno alla neurologia: il cervello umano, infatti, è composto da neuroni che si scambiano informazioni tramite dei segnali elettrici. Sono, inoltre, presenti i dentriti tramite i quali il neurone è in grado di ricevere informazioni, segnali e dati da altri neuroni, una volta ricevuto il segnale, esso viene elaborato all'interno del neurone e successivamente trasferito all'assone che si occupa della trasmissione dei segnali ad altre unità. Il punto di contatto tra i neuroni viene chiamato sinapsi. Solo i segnali che riescono a raggiungere una certa soglia di attivazione verranno elaborati e spediti dal neurone, altrimenti saranno scartati.

[8] Con il termine centroide viene identificato il centro geometrico di una figura, o di un insieme di punti nel piano, che nel caso dell'algoritmo K-means Clustering rappresentano il dataset in analisi. Esso risulta calcolato come la media aritmetica di tutte le coordinate dei punti in analisi. È intuitivo capire come man mano che vengono assegnati nuovi punti ai centroidi deve essere ricalcolata la media e, di conseguenza, la posizione del centroide cambierà, al fine di ottenere una soluzione più veritiera.

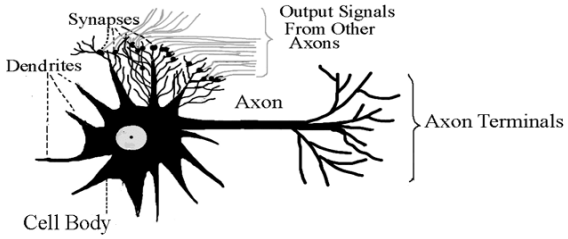


Figura 14: Rappresentazione grafica dei principali componenti di un neurone
Fonte: Zilouchian et al. (2001)

2.3.1 Dal concetto di percettrone e sigmoid neuron alla definizione delle reti neurali

Il passaggio intermedio che è stato effettuato tra lo studio della neurologia e lo sviluppo delle reti neurali è l'ideazione del percettrone. Esso è stato ideato tra la fine degli anni '50 e gli inizi degli anni '60 dal Rosenblatt, il quale, ha proseguito gli studi iniziati nei primi anni '40 da McCulloch e Pitts. L'idea sulla quale ha basato i propri studi Rosenblatt è la realizzazione di un sistema che prendesse in input diversi valori e producesse in output un valore binario.

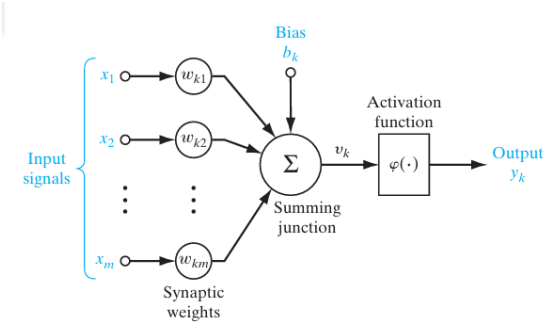


Figura 15: Esemplificazione della struttura di un percettrone
Fonte: Haykin et al. (2009)

Nella figura 15 viene mostrata la struttura semplificata di un percettrone che ha alcuni valori di input e genera un solo valore in output. Per il calcolo dell'output Rosenblatt ha introdotto il concetto di peso, che è un valore numerico che rappresenta l'importanza che viene data a un certo valore di input nella generazione dell'output, ogni dato ricevuto dal percettrone viene moltiplicato per il suo peso e il valore binario dell'output viene definito in base al valore della somma pesata degli input, secondo la seguente regola:

$$\begin{cases} 0 \; se \; \sum_j w_j x_j \leq soglia \\ 1 \; se \; \sum_j w_j x_j > soglia \end{cases}$$

Equazione 4: Equazione per il calcolo dell'output di un percettrone

Modificando i valori dei pesi e della soglia di attivazione è possibile ottenere modelli matematici e risultati molto differenti tra loro anche sullo stesso problema.

Il sistema precedente può essere modificato aggiungendo il concetto di bias[9], nella seguente maniera:

$$\begin{cases} 0 \; se \; w * x + b \leq 0 \\ 1 \; se \; w * x + b > 0 \end{cases}$$

Equazione 5: Equazione per il calcolo degli output con l'introduzione di un bias

Nell'equazione 3, w e x rappresentano rispettivamente i vettori degli input e dei pesi, mentre b = - soglia e rappresenta il bias.

Il passaggio successivo è stato lo studio dei sigmoid neurons, i quali ricevono dei valori in input e generano valori in output che, però, a differenza dei percettroni, possono assumere valori anche compresi tra 0 e 1. La differenza, rispetto al caso precedente, consiste nel metodo di calcolo dell'output:

$$\sigma(w * x + b)$$

Equazione 6: Calcolo dell'output di un sigmoid neuron

Con:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Equazione 7: Funzione sigmoide

[9] Il bias può essere visto come una misura della semplicità di ottenere un output pari a 1, quindi se il valore di b è negativo, sar molto difficile ottenere un output pari a 1, viceversa se il valore di b è positivo.

Come si deduce, quindi, i percettroni e i neuroni sigmoidei sono molto simili se non per l'output che possono generare.

Ovviamente i concetti di percettrone e di sigmoid neuron sono una semplificazione di una struttura molto più complessa, essi, infatti, non contenendo strati nascosti non permettono di risolvere problemi che non siano linearmente separabili, come l'OR esclusivo (XOR), per questo motivo, partendo da queste fondamenta, sono state ideate le reti neurali, per la prima volta studiate da Minsky e Papert nel 1969.

2.3.2 L'architettura delle reti neurali

Le neural network conosciute anche come multi layer perceptron (MLP) o feed-forward artificial neural network, sono composte da un primo livello, chiamato input layer, che come suggerisce il nome, si occupa di ricevere i dati dall'ambiente circostante che, inizialmente, sono valori grezzi che la rete dovrà studiare e migliorare. Poi, sono presenti uno o più, hidden layers, che sono il cuore della rete neurale; essi ricevono dallo strato di input le informazioni grezze, che saranno ad affinate sempre di più scambiandosi segnali e dati, infatti, man mano che si passa da uno strato al successivo le informazioni saranno sempre più complesse ed elaborate. Infine, si giunge all'ultimo strato, ossia l'output layer, che è composto da un singolo neurone, nel caso in cui la rete debba prevedere un certo valore, oppure da più neuroni, nel caso in cui la rete debba svolgere una classificazione. I neuroni presenti in questo ultimo livello ricevono le informazioni dagli strati nascosti e li inviano all'ambiente esterno come risultato dell'algoritmo. I nodi degli strati interni sono collegati ai nodi del layer precedente, dal quale ricevono dati e informazioni da elaborare. Come si deduce, quindi, gli strati sono l'elemento fondamentale per la realizzazione di una rete neurale, permettono di definire come le informazioni e i valori passano attraverso il network e, di conseguenza, come vengono analizzate. Uno dei più famosi modelli di reti neurali è la fully connected, nella quale tutti i nodi sono collegati a tutti i nodi dello strato successivo.

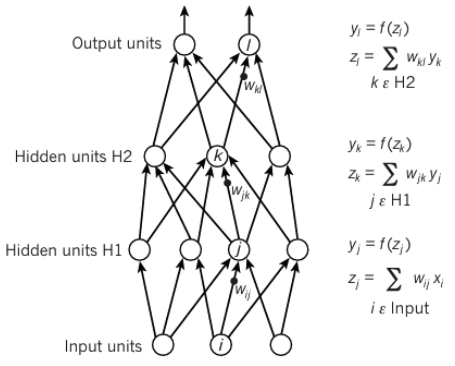


Figura 16: Rappresentazione grafica di una rete neurale con due hidden layers
Fonte: LeCun et al. (2015)

Il numero di neuroni in ogni strato e il numero di hidden layers dipende dalla complessità del problema e dei dati che compongono il dataset, la definizione di questi valori quantitativi risulta essere una delle cose più complesse della progettazione delle reti neurali, in quanto una definizione sbagliata può dare luogo al fenomeno dell'overfitting o underfitting. Generalmente il numero di neuroni di input dipende dal numero di variabili fornite dal problema che la rete neurale sta cercando di risolvere, stesso ragionamento vale per il numero di neuroni nello strato di output, che dipenderà dal numero di valori di output che la rete genererà. In base al numero di hidden layers si possono riconoscere due diverse tipologie di reti neurali: shallow network e deep neural network. Le prime son caratterizzate da uno o, al massimo, due strati nascosti, mentre nel secondo caso sono presenti una quantità maggiore di strati nascosti, questo fa sì che esse siano in grado di analizzare, studiare ed elaborare informazioni molto più complesse.

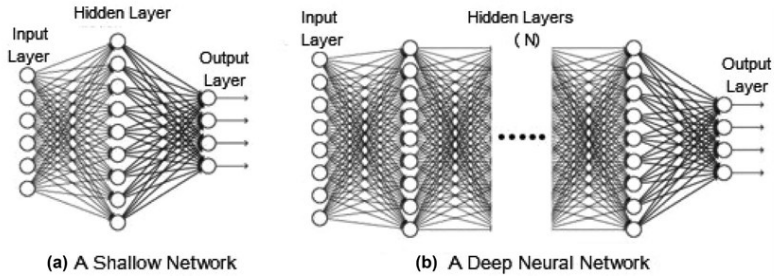


Figura 17: Esempio di shallow e deep neural network
Fonte: Mienye & Swart (2024)

Similmente ai percettroni e ai neuroni sigmoidei, anche le reti neurali possiedono i pesi per i quali ogni valore di input viene moltiplicato ed è presente, inoltre, la funzione di attivazione, la quale permette alla rete di filtrare i segnali, distinguendo quelli che devono essere analizzati e trasmessi, rispetto a quelli che devono essere eliminati o inibiti. Questa operazione di filtraggio viene effettuata attraverso la definizione di una soglia di attivazione, o bias, se il valore della funzione di attivazione supera la soglia allora l'informazione sarà trasmessa al livello successivo, altrimenti verrà eliminata.

Le reti neurali sono in grado di modificarsi internamente e variare la propria struttura, in particolare, nei collegamenti tra neuroni, iterazione dopo iterazione, adattandosi sempre meglio al problema affrontato, sfruttando rinforzi positivi e negativi per apprendere al meglio. Per questo motivo le reti neurali vengono definite learning model e applicano un processo adattivo. Esse vengono spesso utilizzate per risolvere problemi non lineari e per i quali è possibile riuscire a ricavare la soluzione ottima, cosa che risulterebbe impossibile nel caso in cui non venisse utilizzato questa tipologia di algoritmo.

Oltre alle classiche reti neurali, di cui si è parlato fino ad ora, possono essere aggiunti degli strati che si occupano di compiere operazioni specifiche, dando origine a delle nuove reti neurali:

- Le Convolution Neural Network (CNN), le quali seguono la stessa struttura descritta per le reti classiche, ma sono formate da: i convolutional layers e i pooling layers, nello specifico: i primi sono dedicati al riconoscimento di elementi caratteristici che accomunano i valori di input, ricercando gerarchie spaziali, mentre i secondi sono dedicati all'unione dei valori che hanno caratteristiche comuni, che sono state precedentemente identificate. Esse vengono utilizzate, soprattutto, per l'analisi e il riconoscimento di immagini, figure, oggetti, video in diversi campi, ad esempio in quello medico e altri problemi in cui gli input forniti alla rete risultano multiple arrays. La struttura classica delle CNN, quindi, è composta da più convolutional e pooling layers, organizzati in strati che sono completamente connessi tra loro e che forniscono in output una predizione. Solitamente, si ritiene che esse siano computazionalmente più potenti e che non richiedano un forte intervento umano per la programmazione e l'addestramento. Esse, inoltre, risultano essere molto più performanti delle classiche reti neurali per il riconoscimento delle immagini, in quanto in quest'ultime, i nodi, in uno stesso strano, non sono connessi tra loro e siccome ogni pixel di un'immagine risulta essere un input per la rete, le reti neurali classiche tenderebbero a perdere alcune informazioni, le reti convuluzionali, invece, riescono a risolvere questo problema. Esse vengono addestrate attraverso il backpropagation. Nonostante il loro grande successo esse necessitano di una grandissima quantità di dati per funzionare efficacemente.

- Le Recurrent Neural Networks (RNN), esse, a differenza del caso precedente, vengono utilizzate nel caso in cui sia presente una sequenza di valori di input, come, per esempio, nel caso dell'analisi di una conversazione, in quanto risultano in grado di mantenere memorizzate una serie di informazioni, all'interno di un vettore, per tenere traccia della storia di tutti gli input passati alla rete, riuscendo a definire dipendenze temporali e il contesto all'interno del quale queste serie si trovano. Come le CNN, le RNN sono molto potenti e utilizzabili in diversi campi e, teoricamente, potrebbero memorizzare sequenze anche molto grandi, ma il loro addestramento risulta esser molto complesso, in quanto esse soffrono del vanishing gradient[10], nonostante ciò, grazie agli studi e alle ricerche che sono tutt'oggi in corso, l'addestramento sta diventando sempre più semplice.

Come si deduce da quanto detto fino ad ora, nel corso degli anni il deep learning ha subito un grande sviluppo grazie allo studio e realizzazione delle reti neurali classiche, delle CNN e delle RNN, che sono la base da cui è nata e si è sviluppata l'AI.

2.3.3 Principali componenti delle reti neurali

Nella seguente sezione verranno trattati i principali componenti di una rete neurale a partire dalle funzioni di attivazione, che sono un elemento fondamentale, in quanto:

[10] Il problema del vanishing gradient consiste in un fenomeno per il quale, nel momento in cui vengono forniti dataset di grandi, o piccole, dimensioni alla rete neurale, la funzione tende a convergere, rimpicciolendo a ogni ciclo il proprio gradiente fino a zero.

permettono di introdurre la non linearità, superando la limitazione dei percettroni e dei sigmoid neuron, i quali potevano processare solo problemi lineari. Esse permettono alla rete di risolvere problemi molto più complessi e di ricavare informazioni, anche molto complicate, dai dati che le vengono forniti. Come definito nella sezione precedente, all'interno di una rete neurale giungono dei valori di input, i quali vengono moltiplicati per il peso che è stato assegnato all'arco che percorrono e, poi, vengono sommati tutti insieme, a questo punto il valore ottenuto viene passato alla funzione di attivazione, la quale permette di ottenere l'output di quello strato, che verrà trasferito allo strato successivo solo nel caso in cui riesca a superare la soglia di attivazione, altrimenti verrà eliminato e non proseguirà il percorso tra gli strati della rete.

Esistono varie tipologie di funzioni di attivazione, che vengono definite in base al tipo di problema che si sta affrontando, in quanto permettono di giungere a scopi differenti, oltre a garantire livelli di accuratezza molto diversi. Alcuni esempi sono:

- Funzione lineare

$$y = f(x) = \alpha x$$

Equazione 8: Equazione di una funzione lineare

  In questo caso la funzione risulta essere direttamente proporzionale ai valori forniti alla rete e la variabile α può assumere un qualsiasi valore costante scelto dall'utente. In questa funzione il gradiente ha un valore costante, diverso da zero, che è indipendente rispetto al valore di input, quindi, mentre i pesi e i bias vengono aggiornati nel corso dell'algoritmo, il fattore di aggiornamento rimane costante. Essa è una funzione molto semplice, ciò le impedisce di risolvere problemi complessi.

- Funzione a soglia, la quale si può distinguere in bipolare e binaria, questa tipologia di funzione di attivazione è molto simile a quella implementata da McCulloch e Pitts, nella definizione del primo percettrone.

$$y = f(x) = \begin{cases} 0 \; se \; x < 0 \\ 1 \; se \; x \geq 0 \end{cases}$$

Equazione 9: Equazione di una funzione binaria o bipolare

- Rectified linear Unit (ReLU), funzione non lineare il cui modello matematico è il seguente:

$$ReLU(x) = max(0, x) \qquad y = f(x) = \begin{cases} 0 \; se \; x \leq 0 \\ x \; se \; x > 0 \end{cases}$$

Equazione 10: Equazione di una piecewise linear function

Una delle caratteristiche fondamentali di questa funzione è che i neuroni non sono attivati tutti insieme e questo la rende molto efficiente. Essa risulta essere semplice, accelerando la convergenza e riducendo il fenomeno del vanishing gradient, permettendo di addestrare più velocemente reti che hanno tantissimi strati.

Una variazione di questa funzione è la Leaky ReLU, che permette di avere piccoli gradienti nel caso in cui gli input, forniti alla rete, siano negativi.

- Sigmoidal function, di cui l'equazione che rappresenta il modello matematico è stata già riportata durante la trattazione dei neuroni sigmoidei (equazione 5). Questa è la funzione di attivazione non lineare più largamente utilizzata all'interno delle reti neurali, ma soffre del vanishing gradient rendendola inadatta per deep networks. Essa viene sfruttata in problemi di classificazione binaria e permette di convertire i valori di input all'interno del range tra 0 e 1, ottenendo come output un valore percentuale indicante la probabilità.

- Funzione iperbolica tangente (Tanh), di cui l'equazione è la seguente:

$$f(x) = \frac{e^{\alpha x} - e^{-\alpha x}}{e^{\alpha x} + e^{-\alpha x}} \qquad -1 \leq f(x) \leq 1$$

Equazione 11: Equazione di una funzione iperbolica tangente

Essa è una funzione di attivazione non lineare centrata in zero che mappa gli input nell'intervallo [-1;1], quindi, risulta essere molto simile alla precedente, ma simmetrica rispetto all'origine. Anch'essa, come la funzione sigmoidea, soffre del problema del vanishing gradient per dataset di grandi dimensioni, ma riesce a limitare gli effetti che questo problema comporta, soprattutto nei deep network.

- Softmax, essa è composta da diverse funzioni sigmoidee ed è definita:

$$softmax(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

Equazione 12: Equazione di una funzione di attivazione di tipo softmax(z)

Essa fornisce un valore compreso nel range [0;1], che può essere tradotta come la probabilità di una determinata classe, essa, infatti, viene utilizzata, principalmente, per problemi di classificazione multi classe.

Ogni nodo, presente all'interno delle reti neurali, possiede la propria funzione di attivazione.

Quelle elencate fino ad ora sono solo alcune delle funzioni di attivazioni esistenti. La scelta della corretta funzione è un elemento importante e fondamentale, che richiede un grande sforzo da parte del programmatore e di numerosi tentativi. Non c'è una regola fissa che può essere seguita e che definisce quale funzione utilizzare, ma alcuni accorgimenti sono:

- I problemi di classificazione prediligono le sigmoid functions.
- Nel caso si voglia risolvere il problema di vanishing gradient non bisogna utilizzare sigmoid function o Tanh, in quanto sono funzioni fortemente soggette a questo fenomeno.
- La funzione ReLU, solitamente, è quella che permette di ottenere le performance migliori, ma può essere utilizzata solo negli strati nascosti e non per lo strato finale.

Un altro componente delle reti neurali è la Loss Function, chiamata anche funzione obiettivo o funzione di costo. Essa permette di calcolare e identificare la differenza tra la predizione della rete neurale e l'effettivo valore. Come in un qualsiasi algoritmo matematico, l'obiettivo è sempre la minimizzazione di questo errore, per far in modo che il valore predetto sia il più simile possibile al valore effettivo e reale, migliorando, di conseguenza, l'accuratezza del modello. Come per le funzioni di attivazione, anche per le loss function esistono diverse tipologie, che devono essere scelte in relazione al tipo di problema che si sta affrontando.

Tra gli esempi di funzioni di attivazione, più famose ed utilizzate, esistono:

- Mean Squared Error (MSE), esso permette di calcolare il quadrato dell'errore della differenza tra il valore predetto e il valore target, secondo la seguente formula:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

Equazione 13: Equazione del Mean Squared Error

Con:

- n pari al numero di input forniti alla rete neurale.

Essendo un errore quadratico, esso tende a penalizzare maggiormente gli errori molto grandi, di conseguenza, questo modello, risulta essere sensibile agli outliners. Questo è un grande vantaggio nel caso in cui si voglia identificare errori che potrebbero compromettere il processo e l'algoritmo, al tempo stesso potrebbe svantaggiare il sistema nel caso in cui il dataset raccolto e fornito alla rete non sia preciso e l'operatore non sia in grado di migliorare la qualità dei dati.

- Cross Entropy, a differenza della funzione precedente, questa tipologia, è molto più utile per problemi di classificazione e permette di calcolare la differenza di probabilità tra le classi predette e le classi effettive. La sua formula è la seguente:

$$Cross\ Entropy = -\sum_{i} p(y_i) * log(q(y_i))$$

Equazione 14: Equazione della Cross Entropy

Questa funzione, poi, viene adattata in base al numero di classi che la rete deve predire. Essa viene preferita per i problemi di classificazione in quanto penalizza le predizioni effettuate dalla rete, sulle quali essa risulta essere molto sicura, ma che in realtà sono sbagliate. Cercando di addestrare il modello facendo in modo di allineare la probabilità predetta con l'effettiva probabilità di divisione delle classi.

Queste sono le tipologie di Loss Function che sono più largamente utilizzate, anche grazie alla loro facilità di implementazione, ma ne esistono molte altre, che sono utilizzate per problemi molto più specifici, come ad esempio:

- Huber Loss, la quale permette di combinare MSE e Mean Absolute Error (MAE), in modo tale da avere un comportamento diverso in base a: se l'errore è grande, avrà un comportamento lineare, mentre se l'errore è piccolo avrà un comportamento quadratico. In questo modo il modello risulta essere molto meno sensibile a eventuali outliners, risolvendo il problema della presenza di dataset che sono poco precisi e di bassa qualità.
  Tale funzione è definita come segue:

$$Huber\ Loss = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & se\ |y_i - \hat{y}_i| \leq \theta \\ \theta\left(|y_i - \hat{y}_i| - \frac{1}{2}\theta\right) & altrimenti \end{cases}$$

Equazione 15: Funzione di Huber Loss

È fondamentale, quindi, riuscire a scegliere la funzione migliore in base alla tipologia di dati, alla distribuzione che i valori hanno e in base ai task che devono essere eseguiti. In quanto, in base alla scelta effettuata, si otterranno modelli e risultati molto diversi, con performance nettamente differenti.

In ultima battuta, un altro elemento fondamentale per le reti neurali, sono gli algoritmi di ottimizzazione, essi, infatti, permettono di modificare i parametri in modo tale da minimizzare la loss function. Come per le funzioni precedenti, anche in questo caso, è fondamentale scegliere l'algoritmo migliore in base al problema che si sta affrontando, in quanto esso influisce sulla velocità con la quale l'algoritmo converge, oltre ad avere effetto anche sulla qualità della soluzione finale. Anche in questo caso esistono varie tipologie di algoritmi, tra i quali i più utilizzati sono:

- Stochastic Gradient Descent (SGD), esso è l'algoritmo più semplice, permette di modificare i parametri calcolando il gradiente dei pesi e aggiornando quest'ultimi nella direzione di minimizzazione della funzione obiettivo, quindi, in quella opposta rispetto al gradiente. Di conseguenza, si deduce, che è un processo iterativo. La regola che viene applicata è la seguente:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta)$$

Equazione 16: Regola SGD

  Con:

  - η: learning rate[11], il quale è un iperparametro che definisce il salto che viene effettuato a ogni aggiornamento.
  - θ: pesi degli archi presenti all'interno della rete che è stata costruita.
  - $\nabla_\theta \mathcal{L}$: gradiente della loss function.

  Questo algoritmo calcola il gradiente utilizzando un sottoinsieme random dei dati forniti in input, questo permette di contenere i costi computazionali, velocizzando le iterazioni, ma risulta essere molto sensibile alla scelta del valore del learning rate, oltre ad avere difficoltà a convergere nel minimo globale a causa della stocasticità implicita.

- Adaptive Gradient Algorithm (AdaGrad), in questo caso si utilizzano dei valori di learning rate che sono differenti tra parametro e parametro, quindi, viene usato un tasso di apprendimento che sia adattivo e variabile, non costante come nel caso del SGD. In particolar,e si daranno valori maggiori a quei parametri che si vuole

[11] La definizione di questo valore va a tentativi, l'operatore deve essere in grado, osservando le curve di apprendimento, di determinare il valore corretto da affidare a questo parametro. È molto importante che il valore sia scelto correttamente, in quanto esso influisce notevolmente sulla performance del modello, perché se fosse un valore troppo grande tenderebbe ad avere oscillazioni troppo forti, al contrario, se fosse un valore troppo piccolo l'addestramento avverrebbe lentamente, con difficoltà a raggiungere il punto di minimo.

aggiornare sempre meno, mentre per parametri ai quali si vogliono applicare maggiori modifiche si affiderà un valore di tasso di apprendimento minore, quindi, nel momento in cui un peso ha subito diversi aggiornamenti, G_t aumenta il proprio valore, facendo diminuire η, viceversa se il peso non ha subito molti aggiornamenti mantiene un learning rate più elevato. I valori di learning rate vengono scalati in maniera inversamente proporzionale alla radice quadrata della somma di tutti i valori storici, al quadrato, del gradiente, come riportato nella seguente formula:

$$G_t = G_{t-1} + (\nabla_\theta \mathcal{L}(\theta))^2$$

Equazione 17: Somma dei quadrati dei gradienti

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \nabla_\theta \mathcal{L}(\theta)$$

Equazione 18: Equazione di aggiornamento funzione AdaGrad

Con:

- $\nabla_\theta \mathcal{L}(\theta)$: gradiente attuale.
- ϵ: è una costante che fa in modo che il denominatore non vada a zero, evitando, così, divisioni per zero.

Uno degli svantaggi maggiori è che questo algoritmo può portare il learning rate a una diminuzione costante, fermando l'algoritmo precocemente, a causa del parametro di early stop, prima di riuscire a trovare un minimo globale. Questo comportamento lo si deduce direttamente dalla forma del denominatore della funzione di aggiornamento, il quale tende ad aumentare sempre di più, facendo diminuire di conseguenza il valore di tutta la frazione, fino ad azzerarsi.

Questo algoritmo viene utilizzato soprattutto in situazioni nelle quali i dataset raccolti hanno dati che sono scarsi, o di bassa qualità, rendendolo adatto a problemi quali l'analisi del discorso e della lingua.

- Root Mean Squared Propagation (RMSProp), questo algoritmo parte direttamente da quanto citato per il precedente, ma riesce a risolvere il problema del learning rate che diventa troppo piccolo iterazione dopo iterazione. Esso cerca di mantenere una media mobile esponenziale dei quadrati dei gradienti. Questo è possibile non memorizzando tutta la storia dei gradienti, ma focalizzandosi solo sui più recenti, evitando che il learning rate crolli velocemente verso lo zero. La

formula di aggiornamento, quindi, è molto simile al quella dell'AdaGrad, tranne che per il denominatore:

$$E[g^2]_t = \beta E[g^2]_{t-1} + (1-\beta) g_t^2$$

Equazione 19: Equazione della media mobile esponenziale dei quadrati dei gradienti

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} \nabla_\theta \mathcal{L}(\theta)$$

Equazione 20: Equazione aggiornamento RMSProp

Questo algoritmo è particolarmente utile in contesti nei quali la loss function cambia rapidamente, quindi, in ambienti e problemi che non sono stazionari, infatti, viene sfruttata per l'addestramento delle reti neurali ricorrenti. Una delle difficoltà maggiori è la scelta del valore β.

- Adam Optimizer, tra tutti gli algoritmi di ottimizzazione che abbiamo citato fino ad ora, l'Adam risulta essere il più complesso, ma, ad oggi, anche il più utilizzato, in quanto riesce a unire due concetti forti e importanti di altri due algoritmi, ossia: AdaGrad e RMSProp. Esso, quindi, tende ad utilizzare sia la media dei gradienti, indicata come primo momento, sia la media quadrata dei gradienti, definito come secondo momento.
  La regola che viene seguita è:

$$\theta_{t+1} = \theta_t - \frac{\eta \hat{w}_t}{\sqrt{\hat{s}_t} + \epsilon}$$

Equazione 21: Formula aggiornamento funzione Adam

Con:

- $\hat{w}_t = \frac{\beta_1 w_t}{1-\beta_1^t}$ dove $w_t = \beta_1 w_{t-1} + (1-\beta_1)\nabla_\theta \mathcal{L}(\theta)$
- $\hat{s}_t = \frac{\beta_2 s_t}{1-\beta_2^t}$ dove $s_t = \beta_2 w_{t-1} + (1-\beta_2)(\nabla_\theta \mathcal{L}(\theta))^2$

Equazione 22: Formule per il primo e il secondo momento

Nel corso degli anni l'Adam è diventato l'algoritmo più utilizzato all'interno delle reti neurali, in quanto è in grado di gestire l'aggiornamento dei parametri mantenendolo stabile e evitando problemi in caso di un dataset con valori non precisi, ottenendo una convergenza, verso un minimo globale, molto velocemente. Risulta essere molto stabile e la scelta del learning rate influenza poco il risultato.

2.3.4 Processo di addestramento

Fino ad ora, nel corso della tesi, si è parlato dell'architettura della rete neurale e dei suoi principali componenti, nella seguente sezione, invece, si passerà a discutere della parte più dinamica, ossia l'addestramento della neural network.

Tutto parte dai dati forniti in input alla rete, che spesso sono raccolti in dataset, quest'ultimo deve affrontare una fase di preprocessing, prima di essere dato in pasto alla rete neurale. Questo processo permette all'operatore di conoscere i dati nel dettaglio, correggendo e risolvendo eventuali problematiche presenti, per evitare di fornire alla rete dei dati che sono: grezzi o poco significativi, ma, in questa sezione, non verranno affrontate le analisi e le operazioni che possono essere effettuate sui dati, che saranno, invece, citate nel prossimo capitolo più pratico. Una volta che è stata effettuata la fase di preprocessing, il dataset deve essere suddiviso, tra:

- Validation set, all'interno del quale sono presenti dati che vengono utilizzati per valutare l'errore commesso dalla rete sia durante, che dopo, la fase di training, cercando di ottimizzare i parametri e identificare la configurazione migliore. Si ha la necessità di definire questo sottoinsieme perché ci sono alcuni iperparametri, all'interno della rete, che non possono essere scelti dal training set, perché si rischierebbe il fenomeno dell'overfitting, e nemmeno dal test set.
- Training set, questo sottoinsieme, come il precedente, viene utilizzato per addestrare la rete, quindi sono dati che l'algoritmo conosce fin da subito. In particolare, esso viene utilizzato dalla rete per imparare caratteristiche e pattern presenti nei dati, che gli serviranno per eseguire il proprio compito.
- Test set, insieme dei dati, presi dal dataset iniziale, che viene utilizzato per valutare la performance della rete che è stata definita. Sono, infatti, una porzione di dati che l'algoritmo non conosce e che, di conseguenza, non viene utilizzato per la fase vera e propria di addestramento, ma solo in ultima battuta per capire se l'algoritmo sta funzionando correttamente oppure ha solo imparato i dati a memoria dal training set. Quindi, permette di effettuare una valutazione della prestazione. I dati presenti all'interno di questo sottoinsieme devono essere distribuiti nella stessa maniera di quelli presenti nel training set, affinché la rete riesca a eseguire correttamente il compito per il quale è stata generata.

Da questa divisione si capisce come l'obiettivo finale non si fermi a minimizzare l'errore del training set, ma è necessario che la rete riesca a minimizzare l'errore anche su una porzione di dati che non ha mai visto, ossia il test set. Inoltre, i dati che sono presenti in un sottoinsieme non possono, in alcun modo, essere presenti anche in altri subset. Per dividere correttamente l'insieme dei dati iniziali non c'è una regola vera e propria,

generalmente si seguono delle percentuali che sono, indicativamente: 60/70% per il training set, 10/20% per il validation set e, infine, 20/30% per il test set. Queste percentuali, però, non devono essere prese come una regola assoluta, ma devono essere valutate attentamente in base al problema con cui si ha a che fare. Bisogna, però, stare attenti, in quanto avere un test set troppo piccolo potrebbe portare a una forte incertezza statistica sull'errore che viene calcolato, con la conseguente impossibilità di comprendere se un algoritmo funziona meglio rispetto ad un altro. Generalmente questo non è un problema se l'insieme di partenza è molto grande, in quanto anche il test set, arriverà ad avere una quantità sufficiente di dati, maggiori problemi si riscontrano nel caso in cui i dati a disposizione inizialmente sono pochi. Un metodo per la risoluzione di questo problema è il K-Fold cross validation, algoritmo nel quale il dataset di partenza viene diviso in K sottoinsiemi, che non devono essere sovrapposti tra loro, a questo punto avvengono K ripetizioni, in ognuna di queste viene scelto un sottoinsieme diverso per svolgere il compito del test set, quindi, di valutazione della performance, mentre, i rimanenti set, vengono divisi tra validation e training set. L'errore finale viene trovato come media degli errori che sono stati calcolati nelle K iterazioni. Uno dei limiti di questo processo, però, è il fatto che nel corso delle iterazioni vengono utilizzate delle approssimazioni che comportano una stima del valore finale dell'errore, con conseguente perdita di precisione.

Successivamente alla divisione del set di dati, avviene la definizione dei pesi e dei bias, concetti che sono già stati introdotti all'inizio del capitolo. Questa è sicuramente una delle parti più complesse, ma anche una delle più importanti, in quanto la scelta di questi valori ha una grande influenza sul processo di addestramento, determinando: il punto in cui l'algoritmo convergerà e, anche, la velocità con la quale convergerà, oltre al costo. Tutt'oggi il processo di inizializzazione è alla base di numerosi studi e ricerche, in quanto non esiste, ancora, un sistema definito che può essere seguito, ma, piuttosto, vengono utilizzate tecniche semplicistiche ed euristiche, cercando di ottenere le migliori caratteristiche iniziali per la rete e la sua performance. Nonostante tutte le difficoltà per la definizione dei valori di partenza dei parametri, una delle cose fondamentali che bisogna tenere presente è che i valori di partenza devono rompere la simmetria tra le unità, quindi, se due neuroni, degli strati nascosti, hanno la stessa funzione di attivazione e sono connessi agli stessi input, dovranno, per forza, avere dei parametri iniziali che sono diversi gli uni con gli altri, altrimenti si rischierebbe che l'algoritmo di aggiornamento dei dati modifichi i pesi sempre nello stesso modo per entrambi i nodi, non garantendo la massima prestazione dalla rete.

Per tutte le motivazioni citate fino ad ora, una delle vie più economiche e più largamente utilizzate è sicuramente l'inizializzazione casuale dei parametri, così da evitare tutti i problemi e le difficoltà citate precedentemente. Quindi, il processo che viene seguito solitamente, è di inizializzare i pesi casualmente, estraendoli da una distribuzione gaussiana, o un'altra tipologia, mentre per i bias si impostano valori costanti scelti euristicamente. Bisogna tenere presente, però, che la scala scelta per le inizializzazioni ha un peso rilevante, più i pesi iniziali sono grandi più avranno un effetto di rottura della simmetria, ma, al tempo stesso, essi arriveranno ad esplodere, saturando le funzioni di attivazione e rendendo la rete più sensibile anche alle piccole perturbazioni. L'ottimizzazione e la regolarizzazione puntano a due obiettivi di peso diversi, per il primo

sono preferibili pesi più piccoli, mentre nel secondo caso, per garantire la propagazione corretta dell'informazione, si preferiscono pesi maggiori.

Definiti e inizializzati i parametri iniziali, il passaggio successivo è la parte più dinamica della rete, ossia il momento in cui vengono dati in input i valori iniziali al modello, che, poi, attraverseranno i diversi strati nascosti, nei quali verranno elaborati e raffinati, fino a giungere nello strato di output nel quale la rete sarà riuscita a definire una predizione finale, questo processo viene chiamato forward propagation.

In ultima battuta viene fatto un confronto tra il valore predetto dalla rete e il valore effettivo, a questo punto l'algoritmo di forward propagation è in grado di calcolare un valore scalare di costo $J(\theta)$, che rappresenta l'errore commesso dalla rete.

Uno degli aspetti fondamentali è proprio la scelta della funzione per il calcolo del costo, che, solitamente, per le reti neurali sono funzioni non convesse. Nella maggior parte dei casi, il processo che viene seguito è:

- definizione di una distribuzione $p(y \mid x; \theta)$, la quale permette di calcolare quanto è probabile ogni risultato.
- Alla distribuzione di probabilità viene applicato il principio della massima likelihood, il quale permette di scegliere la funzione di costo che, minimizzata, risulta avere i dati più probabili rispetto alle altre. A volte, per semplificare il calcolo, non viene calcolata la probabilità completa di y, ma solo alcune misure statistiche in funzione di x, che sono le più significative per lo scopo per il quale si userà la rete.

Questo comporta l'utilizzo della cross entropy[12] tra la predizione che la rete ha effettuato e i dati reali forniti. Nella scelta della funzione di costo corretta è molto importante stare attenti alla saturazione, in quanto se venisse scelta una funzione di costo non corretta il rischio che si correrebbe è che la funzione diventi piatta, con un gradiente piccolissimo che comporterebbe un apprendimento molto lento, fino a che la rete smetta completamente di imparare.

Una volta scelta la funzione di costo e calcolato il valore tramite l'algoritmo di forward propagation, la rete si occupa del calcolo del gradiente attraverso l'algoritmo di backpropagation, ripercorrendo il percorso inverso all'interno della rete, come suggerisce il nome stesso, cercando di capire quanto ogni peso ha contribuito al fine del calcolo del costo finale. L'obiettivo principale di questo processo è proprio la minimizzazione dell'errore che la rete commette durante la predizione che effettua, l'obiettivo dovrebbe essere quello di raggiungere un minimo globale, ma questo risulta essere computazionalemente molto complesso, tanto che spesso ci si accontenta di fermarsi in

[12] La cross entropy permette di calcolare la distanza che esiste tra le distribuzioni di probabilità delle predizioni effettuate dalla rete e i valori reali presenti all'interno del dataset. Vi è un collegamento diretto alla likelihood che è utilizzata per la scelta della funzione di costo e questo porta lacross entropy ad essere equivalente alla negative log-likelihood, perché questo permette proprio di rappresentare il comportamento tale per cui: se la probabilità è vicino a 1, quindi la rete è sicura della predizione fatta, il logaritmo risulterà tendente a zero, che significa che ha un costo molto basso, viceversa se la rete è meno sicura delle predizioni che ha fatto.

un minimo locale. Tutto questo processo è possibile grazie alla regola della catena, o chain rule, la quale permette di calcolare la derivata di funzioni che sono anche molto complesse e si trovano le une dentro le altre. Utilizzare questa regola nelle reti neurali, comporta, però, che molte sotto-espressioni potrebbero ripetersi più volte, fattore che non è molto determinante, nel caso in cui le reti siano poco profonde, ma quando la profondità aumenta questo comporta un'esplosione del numero dei calcoli da dover effettuare, che può essere anche esponenziale, rallentando, di conseguenza, enormemente l'apprendimento e aumentando i costi computazionali. Per ovviare a questo problema spesso viene utilizzata la programmazione dinamica, che permette di tenere traccia del calcolo dei gradienti intermedi, all'interno di una tabella, così che possano essere velocemente utilizzati, senza doverli ricalcolare tutte le volte, ottenendo un costo computazionale molto inferiore (nell'ordine di grandezza della forward propagation).

Una precisazione importante da fare è che, spesso, non vengono forniti tutti i dati alla rete per svolgere gli algoritmi di forward propagation e backpropagation, in quanto, il risultato ottenuto, potrebbe essere molto accurato e preciso, ma ciò comporterebbe elevati costi, quindi, spesso, si preferisce fornire un mini-batch di dati del training set, il quale permette di raggiungere un livello di stima buono, ma abbassando notevolmente i costi. In particolare potrebbero essere usate tre tecniche differenti, che sono:

- Batch gradient descent, per la quale viene utilizzato l'intero dataset, per esempio nel caso in cui esso risulti essere di piccole dimensioni, quindi i dati non necessitano di una selezione.
- Stochastic gradient descent, che permette di utilizzare un solo esempio alla volta.
- Mini-batch gradient descent, nel quale il dataset originale viene diviso in sottoinsiemi differenti che vengono forniti alla rete epoca dopo epoca.

Spesso, per la rappresentazione dell'algoritmo di backward vengono utilizzati dei grafi computazionali, che permettono di descrivere esattamente il comportamento seguito. Questi grafi sono composti da nodi, che rappresentano le variabili, e da operazioni che sono delle funzioni semplici di una, o più, variabili, rappresentate da archi che collegano i nodi.

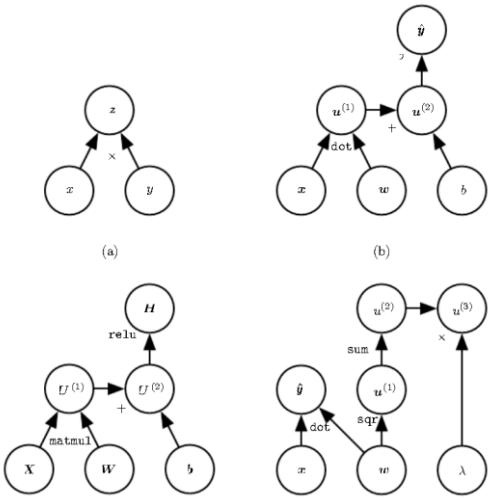


Figura 18: Esempio di grafo computazionale
Fonte: Goodfellow et al. (2016)

Come rappresentato in figura, l'algoritmo di backward è in grado di costruire un sotto-grafo, che rappresenta il percorso compiuto, nel quale è presente un nodo, per ogni variabile, in cui viene calcolata la derivata del costo rispetto al nodo considerato.

Una volta calcolato il gradiente della funzione di costo, esso diventa l'input delle funzioni di ottimizzazione, che permettono di aggiornare i valori dei parametri, con l'obiettivo di una minimizzazione dell'errore commesso dalla rete. Gli algoritmi più conosciuti ed utilizzati sono già stati affrontati nella sezione 2.3.3 di questo capitolo. La metodologia per scegliere l'algoritmo migliore è un argomento fortemente dibattuto tutt'oggi e molti ricercatori non sono d'accordo su quale sia la strada migliore da seguire. Ad oggi, spesso, gli studiosi tendono ad utilizzare l'algoritmo con il quale hanno maggiore familiarità. Scelta la funzione di ottimizzazione è importante passare a scegliere correttamente gli iperparametri presenti all'interno di essa, in quanto hanno un'influenza sul costo computazionale del modello e sulla sua performance. Per la scelta degli iperparametri esistono sia modelli manuali, nei quali bisogna capire il funzionamento di ogni valore e dell'output che la rete vuole predire, sia modelli automatici. Sicuramente tra i valori che devono essere scelti dall'operatore il più importante è il learning rate, le performance migliori per il modello si ottengono proprio quando è stato scelto correttamente il valore di questo iperparametro. Il learning rate viene rappresentato tramite una curva ad U in funzione dell'errore calcolato durante la fase di training. Il grafico permette di capire meglio che: per valori troppo grandi dell'iperparametro c'è il rischio di aumentare l'errore, invece di diminuirlo, allo stesso modo per valori troppo piccoli, si rischia di bloccarsi troppo presto su un valore di errore alto, che non è il minimo raggiungibile, diventa fondamentale riuscire a trovare, quindi, un compromesso.

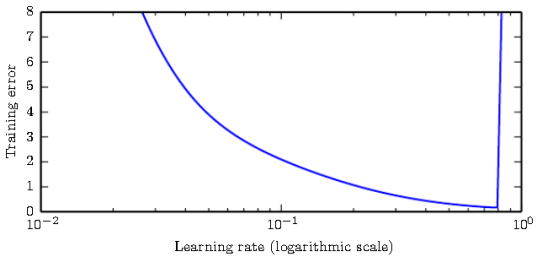


Figura 19: Curva ad U rappresentate la relazione tra learning rate e errore di training
Fonte: Goodfellow et al. (2016)

Il learning rate è solo uno degli iperparametri che devono essere definiti, ma ne esistono degli altri, per i quali bisogna monitorare sia l'errore commesso sul training set che quello sul test set, affinché si riesca ad assegnare, a tali parametri, il valore corretto.

Definiti gli iperparametri è possibile, utilizzando le funzioni di ottimizzazione per aggiornare i pesi, che sono stati definiti inizialmente, con l'obiettivo di una minimizzazione dell'errore commesso, concludendo, in questa maniera, l'algoritmo di backward.

Il ciclo completo descritto fino a questo momento, quindi forward propagation insieme a backward propagation, rappresenta un'iterazione del processo di training della rete. Durante la fase di addestramento si eseguono diverse iterazioni del processo, in ognuna delle quali viene fornito al modello un mini batch di dati differenti. Le iterazioni che devono essere svolte sono pari al numero di mini batch che sono stati identificati inizialmente, una volta che sono stati passati tutti i sottoinsiemi si passa all'epoca[13] successiva.

Alla rete viene sottoposto, durante la fase di addestramento, periodicamente e non a ogni iterazione, il validation set, con il quale essa svolge solo la fase di forward propragation. Esso serve per calcolare l'errore commesso dalla rete sui dati di questo sottoinsieme e confrontarlo con il migliore errore ottenuto fino a quel momento, se c'è stato un miglioramento vengono salvati i nuovi pesi, altrimenti si aggiorna il contatore di iterazioni per un eventuale early stop. Valutare la performance della rete, anche sul validation set, è fondamentale per capire se la rete sta incorrendo in due problemi molto gravi, che sono stati già citati diverse volte fino ad ora e che adesso verranno definiti meglio:

[13] L'epoca è un altro iperparametro molto importante, che viene definito prima della fase di training della rete, esso precisa il numero di volte che l'intero training set, suddiviso in diversi sottoinsiemi, viene interamente processato dalla rete, quindi un'epoca rappresenta un passaggio completo, di tutti i sottinsiemi di dati, all'interno della rete. Questo passaggio completo può essere eseguito più volte e la definizione del numero esatto è un elemento cruciale e importante della fase di addestramento, inserirne un numero troppo basso comporterebbe fenomeni di underfitting, dove la rete non riesce ad apprendere correttamente, al tempo stesso fissarne un numero troppo alto significherebbe essere a rischio overfitting, dove, contrariamente a prima, la rete impara troppo dai dati, imparandoli quasi a memoria. Non esiste una vera e propria regola per definire il numero corretto, spesso in fase di training si guarda come evolve l'errore.

- Underfitting, fenomeno nel quale la rete non è in grado di apprendere correttamente le caratteristiche e la struttura dei dati, non riuscendo, di conseguenza, ad abbassare sufficientemente l'errore che commette.
- Overfitting, questo è il fenomeno opposto rispetto al precedente, in questo caso, la rete, apprende troppo bene i dati presenti nel training set, riuscendo a memorizzare addirittura i rumori presenti, ma avendo performance molto inferiori nel validation e test set. Per monitorarlo spesso viene osservato l'andamento dell'errore del training set e del validation set, solitamente quando si presenta questo fenomeno il primo errore tende a diminuire costantemente, mentre il secondo diminuisce inizialmente, poi, mentre l'errore sul training continua a scendere, quello sul validation, rincomincia a salire.

Quindi, come si deduce, entrambi i problemi riguardano la capacità della rete di apprendere. Essi devono essere attentamente monitorati per essere individuati e corretti tempestivamente.

Per risolvere questi problemi si agisce in due modi differenti:

- Per l'underfitting è possibile:
  - Aumentare il numero di livelli presenti all'interno della rete, o il numero di nodi per ogni layer, così che la rete riesca ad apprendere meglio avendo un numero maggiore di layer o nodi per memorizzare i dati.
  - Aumentare il numero di iterazioni, o di epoche, così che la rete abbia più tempo per imparare i dati.
  - Ridurre, o eliminare completamente, la regolarizzazione.

- Per l'overfitting, all'opposto, è possibile utilizzare tecniche di regolarizzazione tra cui le più conosciute sono:
  - Inserire una penalità ai pesi usati, maggiore è la penalità più alta sarà la regolarizzazione.
  - Fermare l'addestramento appena l'errore di validazione smette di migliorare attraverso un early stop. Siccome l'overfitting, si basa proprio sul concetto che l'errore del training set continui a scendere mentre a un certo punto l'errore sul validation set torni a salire, una strategia che si può seguire è quella di salvare i pesi ogni volta che l'errore di validazione migliora. In caso contrario si definisce un valore, ossia la patience, iperparametro che rappresenta il numero di epoche senza miglioramento che si aspettano prima di interrompere completamente l'algoritmo.
  - Lavorare sul dropout, che è un ulteriore iperparametro che deve essere definito a priori, il quale aumenta, o diminuisce, il numero di nodi che casualmente vengono disattivati durante il training, il suo comportamento è riportato nella figura sottostante. Questo è sicuramente uno dei metodi più economici.

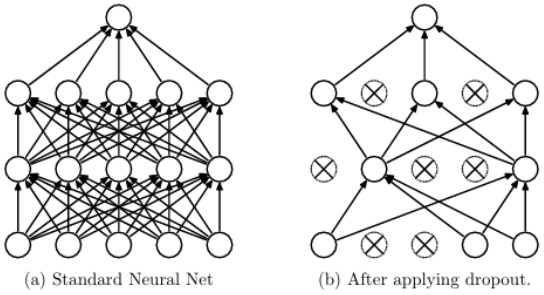


Figura 20: Figura che rappresenta il funzionamento del dropout
Fonte: Srivastava et al. (2014)

Le ultime citate, quindi, sono tutte tecniche diverse di regolarizzazione, che permettono di ridurre la varianza, senza, però, aumentare troppo l'eventuale bias.

Una volta che si presenta una condizione di stop la rete è stata ottimizzata e i pesi sono stati fissati, quindi essa è pronta per essere utilizzata per lo scopo per il quale è stata generata. A questo punto non rimane che fornirgli il test set, sul quale effettuare, nuovamente, solo la forward propagation, che fornirà gli output dai quali è possibile calcolare i valori denominati come metriche di performance della rete.

2.3.5 Metriche per la valutazione delle prestazioni

Nel corso della seguente sezione verranno introdotte alcune metriche e strumenti per la valutazione della performance della rete, partendo proprio dalle training curves, le quali sono in grado di evidenziare i fenomeni di overfitting e underfitting, citati alla fine della sotto-sezione precedente. In particolare il grafico di queste curve mette in mostra due elementi principali:

- La curva dedicata alla training loss, la quale tende a diminuire costantemente.
- La curva dedicata alla validation loss, la quale presenta una prima fase di discesa, per poi tornare a salire, formando un grafico a U. Il punto di minimo di questa curva, dopo il quale essa torna a crescere, rappresenta l'epoca ottima in cui fermare l'addestramento tramite un early stop, in quanto si ottiene una funzione di costo con il valore più basso possibile.

Queste due curve, rappresentate nel grafico, permettono di mettere in evidenza come si evolve la funzione di costo, relativa al training e al validation set, nel corso del tempo dedicato all'addestramento della rete, il quale si è soliti misurarlo in epoche. Dal grafico che si definisce è possibile riconoscere fenomeni di:

- Overfitting, il quale si presenta nel caso in cui la loss function continua a scendere per il training set, mentre per il validation tende a risalire, denotando di aver imparato molto bene i dati del training, ma non riuscendo a generalizzare i risultati anche al validation set.

- Underfitting, nel caso in cui le curve abbiano ancora valori alti, che continuano a diminuire, nel momento in cui l'addestramento si ferma.

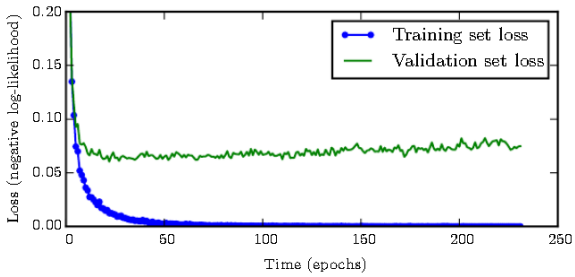


Figura 21: Training curve
Fonte: Goodfellow et al. (2016)

Oltre alle curve ad U, per l'addestramento, esistono anche le learning curves, le quali permettono di identificare come e quanto migliorerebbero le prestazioni della rete nel caso in cui si aumentassero il numero di dati all'interno del training set. Anche queste curve vengono rappresentate, come le precedenti, all'interno di un piano cartesiano, dove sull'asse delle ascisse viene riportato il numero di dati, mentre su quella delle ordinate viene riportata la performance. L'andamento tipico di queste curve è: una rapida salita iniziale, per poi appiattirsi, questo rappresenta, graficamente, il concetto che: inizialmente avere una grande quantità di dati comporta una rapida crescita della performance, ma poi pian piano, aumentandone troppo il numero, si ottiene l'effetto opposto e la rete inizierà ad avere performance sempre più basse o simili tra loro. Quindi all'interno del training set, bisogna inserire un numero di dati pari al numero di valori che permettono di ottenere la performance massima dalla rete.

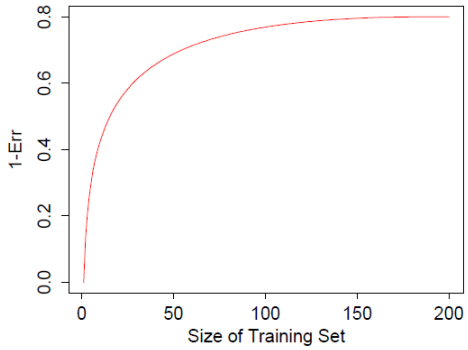


Figura 22: Learning curve
Fonte: Hastie et al. (2009)

Nella trattazione delle learning curves è stato citato il concetto di performance, ma come si va a calcolare effettivamente questo valore? Esistono diversi strumenti e metriche che permettono di calcolare questa misura, a partire dalla matrice di confusione. In particolare, questo strumento, risulta essere largamente utilizzato, soprattutto, nei

problemi di classificazione. Prendendo ad esempio un problema di classificazione binaria, la rete riceve un'istanza in input, la quale viene mappata in classi reali {p, n}, dove p sta per positivo, mentre n per negativo. Il modello, poi, produce una predizione {Y, N}, dove Y è una predizione positiva, mentre N è una predizione negativa. Si vengono, così, a delineare quattro diverse casistiche, che sono riportate nella matrice sottostante:

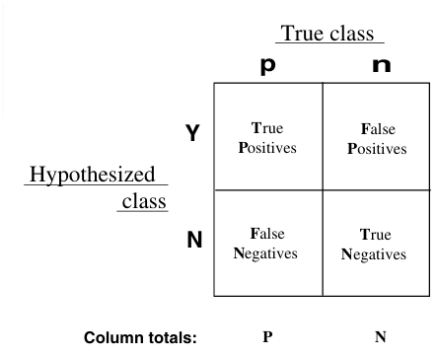


Figura 23: Matrice di confusione
Fonte: Fawcett (2006)

Le classi che si identificano sono:

- True positive, che indica quando la classe reale è positiva e la rete è riuscita a predirla come vera.
- True negative, che indica quando la classe effettiva è negativa e la rete la predice come tale.
- False positive, in questo caso la classe reale è negativa e viene classificata come positiva.
- False negative, in questo caso la classe reale è positiva e viene classificata come negativa.

In questo esempio il problema è una classificazione binaria, quindi, si definisce una matrice 2x2, ma, in base al numero di classi che la rete deve predire, la matrice si ingrandirà, quindi, un problema di classificazione con N classi, produrrà una matrice NxN.

È importante notare come le predizioni giuste, quindi quelle in cui la rete funziona correttamente, sono quelle presenti sulla diagonale principale della matrice.

A partire dalla matrice di confusione e dai suoi valori, è possibile calcolare alcune metriche, tra cui:

- Recall, ossia il true positive rate o sensitivity, quindi il numero di true positive sul totale delle classificazioni fatte correttamente dalla rete, quindi l'efficacia della rete di predire correttamente il valore positivo. Calcolato con la seguente formula:

$$Recall = \frac{TP}{TP + TN}$$

Equazione 23: Formula per il calcolo della recall

- False positive rate, che calcola il numero di campioni che sono negativi nella realtà, ma che il modello sbaglia e classifica come positivi. Similmente alla recall, si calcola:

$$fp\ rate = \frac{FP}{FP + TN}$$

Equazione 24: Formula per il calcolo del false positive rate

- Specificity, che calcola l'efficacia della rete di identificare tutti i campioni che sono effettivamente negativi e si calcola con la seguente formula:

$$specificity = \frac{TN}{FP + TN}$$

Equazione 25: Formula per il calcolo della specificity

- Precision, metrica che permette di misurare il numero di dati classificati come positivi correttamente dalla rete, sul totale di istanze classificate come positive. Essa si calcola come:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

Equazione 26: Formula per il calcolo della precision

A differenza della recall mette in luce il numero di campioni che la rete ha classificato come positivi e lo sono veramente, sul totale di casi classificati come positivi dalla rete. Quindi, questa metrica, permette di rispondere alla domanda: di tutte le istanze predette come positive, per quante la rete ci ha effettivamente azzeccato?

- Accuracy, metrica che permette di misurare il numero di volte che la rete ha predetto i valori correttamente, che siano essi positivi o negativi. Essa si calcola come:

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

Equazione 27: Formula per il calcolo dell'accuracy

- F1-score, metrica che permette di unire i concetti di Precision e Recall in un unico valore di performance, basato sulla media armonica. Tale metrica si misura come:

$$\text{F1} - \text{score} = 2 * \frac{\text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Equazione 28: Formula per il calcolo di F1-score

  A volte viene aggiunto un parametro β, il quale permette di assegnare maggiore importanza a una delle due metriche, rispetto all'altra, in base al concetto che si vuole valorizzare maggiormente.

Come si nota tutte queste formule sono strettamente collegate alla matrice di confusione e ai suoi valori, quindi, nel momento in cui un dato della matrice cambia, allora anche i valori delle metriche cambieranno mostrando una nuova performance della rete. Le formule precedenti, riguardanti le metriche di valutazione di un problema di classificazione, sono state scritte per un problema binario, ma si possono generalizzare a problemi con N classi, semplicemente aggiungendo le sommatorie.

Un altro strumento che può essere utilizzato, per la valutazione della prestazione della rete, è la funzione di costo, in quanto la cross entropy permette di misurare la distanza tra la distribuzione della predizione fatta dal modello e quella reale, permettendo di mettere in luce quanto il modello è sicuro e confidente nelle proprie decisioni e, questo, è sicuramente un grande vantaggio rispetto alle altre metriche che permettono solamente di sottolineare la correttezza, o meno, della predizione fatta.

Fino ad ora ci si è concentrati su metriche, per la valutazione delle prestazioni della rete, in problemi di classificazione, in realtà ogni tipologia di problema ha le proprie metriche, le quali permettono di verificare che la rete stia svolgendo correttamente il proprio compito.

Per esempio per problemi di regressione, vengono utilizzati:

- MAE (Mean Absolute Error), che viene calcolato attraverso la seguente formula:

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|e_i|$$

Equazione 29: Formula per il calcolo MAE

- RMSE (Root Mean Square Error)

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2}$$

Equazione 30: Formula per il calcolo RMSE

Con:

- e: che identifica l'errore commesso dalla rete, quindi la differenza tra il valore reale e quello predetto.

Entrambe le formule presentate permettono di misurare l'errore commesso dalla rete durante la regressione, solo che nel primo caso viene affidato lo stesso peso a tutti gli errori, mentre nel secondo vengono penalizzati maggiormente gli errori più grandi, rispetto a quelli più piccoli a causa dell'elevamento al quadrato. Quindi, l'RMSE, deve risultare sempre maggiore del MAE, nel caso in cui vengano calcolati entrambi sugli stessi valori.

La letteratura afferma che non c'è una metrica migliore dell'altra, quindi, non si è arrivati, ancora, a una conclusione riguardo a quale sia la misura più corretta da utilizzare per valutare i risultati ottenuti dalla fase di addestramento di una rete neurale. Solitamente, si è concordi sull'affermare che per problemi in cui gli errori seguono una distribuzione normale si preferisca RMSE, alternativamente si dovrebbe utilizzare il MAE.

Un'altra metrica fondamentale per i problemi di regressione è:

- R^2, che è il coefficiente di determinazione, calcolato come:

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

Equazione 31: Formula per il calcolo del coefficiente di determinazione

Al numeratore della formula viene inserita la differenza, al quadrato, dei residui, mentre al denominatore, la differenza, sempre al quadrato, dei dati rispetto alla loro media. Solitamente questo è un valore compreso tra 0 e 1, dove 1 indica che la rete riesce a rappresentare correttamente la differenza, mentre 0 significa che il modello non riesce ad imparare questa differenza, prevedendo semplicemente la media dei valori.

Fino ad ora si sono andate a descrivere tantissime metriche differenti e, come già affermato diverse volte, non esiste una misura che è migliore rispetto alle altre in maniera assoluta, perché tutto dipende dal problema che si sta affrontando e dalle sue caratteristiche. Per esempio, metriche quali accuracy, F1-score e precision, sono molto sensibili a un eventuale sbilanciamento presente nel dataset fornito alla rete, inoltre, l'accuracy è una misura che permette di fornire una buona indicazione sulla performance della rete, senza però essere in grado di dare una valutazione sulla singola classe che viene predetta dalla rete stessa. Anche in base a queste caratteristiche citate, si capisce, come, in base alla tipologia di problema ci si debba orientare in maniera differente per valutarne correttamente le prestazioni. Spesso si tende a calcolarle tutte per avere una visione completa delle prestazioni dell'algoritmo generato, ma bisogna sempre tenere in considerazioni del trade off tra completezza delle informazioni e costi computazionali per il calcolo di tali valori.

2.3.6 Contesti di applicazione delle reti neurali

Come analizzato anche nelle prime sezioni del capitolo le applicazioni degli algoritmi di ML sono le più svariate e questa caratteristica non risulta essere diversa per le reti neurali, esse, infatti, vengono sfruttate per risolvere diverse tipologie di problemi, anche non lineari, come: regressione, clusterizzazione e classificazione.

Alcuni contesti nei quali vengono utilizzate sono:

- Il riconoscimento di figure, persone, animali, lettere o numeri, all'interno di immagini o video.
- Il riconoscimento della lingua parlata, meccanismo molto utile per alcuni programmi o applicazioni di traduzione, o di riconoscimento della lingua in simultanea.
- Per effettuare diagnosi mediche, grazie all'imaging, alla cura e il monitoraggio di pazienti a distanza, oltre ad essere stato largamente utilizzato durante il periodo del Covid-19.
- Per analizzare il territorio e fare una valutazione dei più importanti indicatori geologici, come ad esempio prevedere il meteo, la temperatura o altre informazioni geologiche ritenute interessanti, oltre a potersi focalizzare anche sull'aspetto energetico.
- Per fare previsioni di vendita e di mercato.
- Per prevenire frodi fiscali.
- Per l'inteligent Decision Making.
- Per le previsioni e la gestione del traffico.

- Per fornire ai clienti supporto e consigli: per una migliore gestione del e-commerce e in alcune piattaforme di streaming di film, o canzoni, spesso è presente la sezione dedicata ai suggerimenti, dove l'applicazione fornisce consigli sulla base delle preferenze e degli acquisti che il cliente ha effettuato.
- Negli anni si sono sempre più digitalizzate città e case, si sta, quindi, andando sempre di più verso smart home e smart city. Oltre a Internet of things (IoT) e la cybersicurezza collegata.

Queste sono solo alcune delle applicazioni principali delle reti neurali, ma esse vengono sfruttate anche da:

- Neuroscienziati per definire il comportamento del cervello umano ed animale.
- Ingegneri per la progettazione di macchine intelligenti.
- Matematici per analizzare e studiare meglio problemi sempre più complessi.
- Inoltre vengono sfruttate anche per diversi scopi industriali all'interno delle moderne aziende.

Come si può ben capire dall'elenco appena citato le reti neurali, ma anche più in generale gli algoritmi di machine learning, possono essere utilizzati in diversi contesti, garantendo grandissimi vantaggi, come: velocizzazione del processo, automazione, digitalizzazione e sviluppo. Grazie ai numerosi aspetti positivi che hanno apportato e che continuano ad apportare, questi modelli, sono sottoposti a continui studi, ricerche e sviluppi che li portano a migliorarsi sempre di più, aumentando le proprie applicazioni e potenzialità.

CAPITOLO 3: Applicazione di un modello surrogato di machine learning per la predizione di bande di stop in metamateriali a trave piegata periodicamente

3.1 Introduzione al problema e idea per la risoluzione

Il progetto di tesi che è stato sviluppato parte proprio dal concetto di metamateriale, in particolare una trave, la quale risulta essere piegata periodicamente. Come è stato già descritto in maniera più approfondita durante il primo capitolo, la caratteristica al centro degli studi effettuati è la capacità di riuscire, a determinate frequenze, a bloccare completamente le onde elettromagnetiche grazie all'accoppiamento contra-direzionale, dovuto alla piegatura del metamateriale stesso.

Queste frequenze, che definiscono la banda entro la quale le onde riescono ad essere bloccate, vengono chiamate BG_start, ossia la frequenza dove inizia la banda di bloccaggio, e BG_end, che, invece, indica la frequenza in cui questa banda termina, da questi due valori è possibile calcolarne la dimensione e la media.

I ricercatori dell'università di Parma, hanno ideato un codice Matlab, il quale è in grado, implementando il DSM[14], di calcolare la stop band di configurazioni, di metamateriali a trave piegati periodicamente, le quali gli vengono fornite in input.

In particolare, il suddetto codice accetta quattro dati di input, per i quali sono forniti dei range di valore. Essi sono:

- r: che rappresenta il raggio della sezione della trave. Generalmente, con valori piccoli viene rappresentata una trave molto sottile e flessibile, la quale permette la generazione di bande di bloccaggio più ampie, al contrario, per valori di r maggiori, la trave sarà più rigida determinando una banda di inibizione di piccole dimensioni.
- Alpha: ossia l'angolo di piegatura del segmento della trave, rispetto alla direzione dritta. Con α uguale a zero, non si avrà il fenomeno di bloccaggio, in quanto la trave risulterà dritta, non permettendo l'accoppiamento contra-direzionale. Mentre, per valori di angolo compresi tra i 10 e i 25 gradi si avranno i valori di stop band maggiori.
- A: valore che rappresenta la lunghezza del segmento piegato.
- L: rappresenta la lunghezza di una cella ripetuta.

[14] Dynamic Stiffness Method, affrontato nel primo capitolo teorico riguardo i metamateriali.

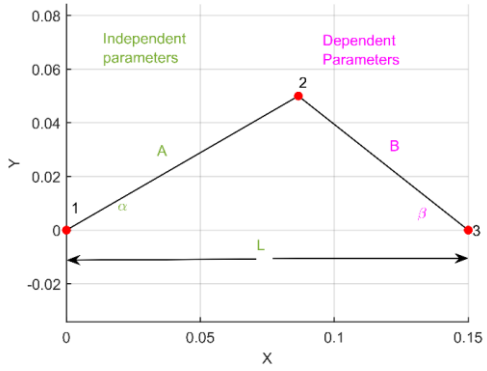


Figura 24: Esempio di trave piegata periodicamente

Oltre a questi valori, il codice Matlab accetta anche un ulteriore parametro di input, che è n. Esso rappresenta il numero di divisioni dei parametri di input, ciò significa che i range fisici dei valori di input non vengono considerati interamente, ma vengono campionati in base al valore dato a n. A tale parametro, nel corso del progetto, è stato affidato valore n = 10, così da ottenere, attraverso il prodotto cartesiano di tutti i valori, 10^4 possibili configurazioni da testare.

Per ognuna di queste configurazioni il codice svolge tre operazioni principali:

- Verifica della validità della configurazione: alcune di esse, infatti, siccome nascono direttamente dal prodotto cartesiano, potrebbero non rispettare vincoli fisici importanti, per questo, è fondamentale effettuare subito un controllo ed eventualmente evidenziare le configurazioni che non sono ammissibili.
- Successivamente, vengono costruite le curve di dispersione, che permettono di rappresentare come le onde si propagano attraverso la struttura, sottolineando le zone in cui esse riescono a propagarsi e quelle, invece, dove vengono inibite, o bloccate completamente.
- Infine, viene individuata, per ogni configurazione valida, la stop band, calcolando nello specifico BG_start, BG_end, BG_width (ossia la grandezza della banda di bloccaggio, calcolata come la differenza tra il valore di end e quello di start) e BG_mean (che, come suggerisce il nome, calcola la media tra il valore di end e quello di start).

Viene, quindi, generato un database all'interno del quale sono contenute tutte queste informazioni appena citate, quindi, sia i valori di input che gli output generati tramite la simulazione. Ad ogni simulazione viene, poi, associato, aggiungendo una nuova colonna al database, uno stato: "OK", se la configurazione è valida e genera una banda di stop, "No BG", nel caso in cui la configurazione rispetti tutti i vincoli fisici, ma non presenti bande di bloccaggio, e "Invalid" nel caso in cui la configurazione testata non rispetti i vincoli fisici e quindi sia impossibile calcolarne le frequenze di generazione della banda. Viene inserita, poi, in un ulteriore colonna, la reason, ossia la motivazione per cui è stato assegnato quel determinato valore di stato alla configurazione. Viene, poi, mostrata a schermo all'utente i parametri di input della configurazione che permette di ottenere l'ampiezza della stop band maggiore.

Oltre ai parametri di input, che possono essere modificati liberamente, sono introdotte, anche, alcune proprietà fondamentali, le quali risultano essere fisse e non modificabili:

- Il modulo di Young: che permette di misurare la rigidezza del materiale alla deformazione assiale.
- Densità.
- Coefficiente di Poisson.

Questo codice risulta essere molto efficacie ed estremamente preciso, ma, già solo da questa introduzione fatta, si può dedurre il suo più grande limite, ossia il costo computazionale, infatti, per la generazione di un dataset con 10^4 configurazioni, il codice ha richiesto un tempo di esecuzione tra gli 8 e i 10 minuti, è, quindi, facile dedurre che all'aumentare del numero di divisioni, e, di conseguenza, del numero di configurazioni che devono essere testate, il tempo impiegato da questo codice risulta essere troppo elevato e insostenibile.

Per questo motivo la soluzione che si è voluta implementare è la sostituzione di questo codice con un modello surrogato di machine learning, in particolare un algoritmo formato da due reti neurali che si occupino di prevedere gli stessi valori del codice Matlab, quindi frequenza di start e di end della banda di bloccaggio, dai quali, poi, è possibile ricavare ampiezza e media. Uno dei grandi vantaggi del machine learning è proprio la capacità di riuscire ad elaborare i dati in tempi molto brevi, eliminando completamente il problema relativo al costo computazionale del codice Matlab.

Come algoritmo di ML scelto, per la generazione di un modello surrogato che potesse svolgere lo stesso compito del codice Matlab, ma in maniera più rapida, si è deciso di utilizzare le reti neurali feed forward, in quanto, questa tipologia di algoritmo, è in grado di:

- Apprendere relazioni che non sono lineari, infatti, come si vedrà nel corso del capitolo, la relazione che intercorre tra i quattro parametri di input, è molto complessa e non lineare, di conseguenza, si ha la necessità di utilizzare algoritmi che diano la possibilità di rappresentare problemi con questa tipologia di relazione.
- Esse danno la possibilità di svolgere contemporaneamente più compiti, come nel caso in analisi, in cui deve essere svolta sia l'attività di classificazione, che di regressione. Anche questa è una caratteristica tipica delle reti neurali, mentre altri algoritmi non permettono tale possibilità.
- Le reti neurali, inoltre, garantiscono costi computazionali e tempistiche bassissime su dataset composti anche da milioni di dati, riuscendo a maneggiarli in maniera semplice e rapida.

Altri algoritmi non garantiscono i vantaggi appena citati:

- Random forest: è ottimo per problemi con pochi dati con relazioni lineari, in cui deve essere svolta una sola operazione, non risultando adatto a problemi multi-task.
- Regressione lineare e logistica: sono modelli lineari, che non riescono in alcun modo a descrivere correttamente problemi che non presentano questa caratteristica, come quello in analisi.
- KNN: anche questo algoritmo soffre del problema di curse of dimensionality, aumentando le dimensioni le distanze sono sempre meno rappresentative, portando a non riuscire più a generalizzare e finendo per memorizzare esclusivamente i dati.

Per questo motivo è stato scelto di utilizzare le reti neurali le quali, sono risultate essere, il modello che meglio permette di rappresentare il problema in analisi.

3.2 Fase 0: generazione del dataset

Il primo passo che è stato effettuato, per la creazione dell'algoritmo di machine learning, è stato la generazione di un dataset, che potesse essere utilizzato per addestrare e testare la rete. Pertanto, la prima operazione compiuta, è stata:

- Fissare i valori dei range dei parametri di input.
- Fissare il valore relativo al numero delle divisioni.

Quindi, son stati definiti i valori dei parametri di input che il codice Matlab richiedeva.

In particolare gli intervalli scelti sono:

- Per r: [0.001; 0.003] m
- Per Alpha: [0; 60] gradi
- Per A: [0.10; 0.30] m
- Per L: [0.10; 0.30] m

Mentre, per quanto riguarda le divisioni, è stato definito, come già affermato all'inizio del capitolo, un dato pari a 10, questo valore è un buon compresso tra costo computazionale per il calcolo di tutte le possibili combinazioni, dettaglio di analisi e numerosità, o dimensione, del dataset per l'addestramento delle reti neurali.

Come già indicato nella sezione precedente, l'obiettivo di questo valore è di campionare gli intervalli fisici di ogni parametro di input, ciò significa che ogni valore viene diviso in 10 sezioni equidistanti tra loro, all'interno del proprio range. Quindi, per esempio, per l'intervallo di r si sono ottenuti i seguenti valori: [0.001, 0.001222, 0.001444, …, 0.003].

Una volta definiti i valori di input, è stato eseguito il codice Matlab, che ha permesso la generazione di una griglia con tutte le possibili configurazioni, generate dal prodotto cartesiano delle divisioni dei valori di input. Tale griglia è il dataset che è stato utilizzato per addestrare le reti neurali. Esso è composto da undici colonne, contenenti: i quattro

parametri di input, i 4 valori di output, ai quali si aggiunge BG_norm che indica la misura normalizzata dell'ampiezza, lo stato e la reason.

Questo dataset grezzo è stato il punto di partenza e le fondamenta per la creazione del modello surrogato. A seguito di questa fase di generazione del set di dati, è stata eseguita una fase di analisi e di preprocessing dei valori presenti nel dataset, proprio per cercare di individuare pattern ricorrenti, caratteristiche, peculiarità ed, eventuali, problematiche. Con l'obiettivo di ottenere un set di dati pulito e adatto all'addestramento di un modello di Machine Learning, che funzionasse correttamente e avesse buoni risultati di performance.

3.3 Fase 1: analisi dei dati

Come accennato al termine della sezione precedente, generato il dataset di partenza, è stata effettuata un'analisi esplorativa dei dati contenuti al suo interno, con l'obiettivo di riuscire ad identificare:

- La struttura dei dati.
- La distribuzione delle diverse classi, con un'analisi più specifica delle motivazioni dell'invalidità di alcune configurazioni.
- Distribuzione dei diversi Status per i singoli parametri di input.
- Calcolo della distribuzione degli output e la loro relazione.
- Correlazione tra alcuni parametri di input e output.

Oltre, alle attività appena elencate, ne sono state svolte altre più specifiche, che saranno descritte nel corso di questa sezione. Come si intuisce, quindi, l'obiettivo è stato quello di analizzare, in modo approfondito, il dataset, senza voler applicare cambiamenti o preparare i dati alla fase di addestramento, ma, semplicemente, trarre alcune informazioni teoriche che sono risultate utili per capire alcuni valori ottenuti in fase di implementazione del modello.

Questa analisi, come tutte le altre fasi che saranno descritte in seguito, è stata effettuata tramite il servizio di Google Colab, in linguaggio di programmazione Python.

In particolare, per questa prima fase, sono state utilizzate le seguenti librerie Python:

- Pandas: che ha permesso di leggere e manipolare tabelle di dati.
- Numpy: che ha permesso di eseguire operazioni matematiche sui valori del dataset.
- Matplotlib: libreria che ha permesso di realizzare grafici rappresentativi, per una comprensione più facilitata delle informazioni e dei valori.
- Seaborn: altra libreria che è stata utilizzata per la realizzazione dei grafici.
- Scipy: dalla quale è stato importato il modulo Stats, che ha permesso di calcolare ed utilizzare metriche statistiche.

3.3.1 Distribuzione dei valori di input

Dopo aver importato tutte le librerie e i moduli utili per le analisi che sono state effettuate nel corso del notebook, è stata fatta una rapida analisi della struttura del dataset.

La prima operazione che è stata effettuata è il calcolo della distribuzione dei quattro parametri di input, da cui si è dedotto che hanno tutti delle distribuzioni uniformi, quindi, presentano skewness[15] nulla. Questa è stata un'osservazione fondamentale da fare per il notebook successivo, in quanto ha evidenziato che non vi sia la necessità di effettuare una trasformazione logaritmica sui valori di input, ma è sufficiente una trasformazione lineare (come Min-Max o z-score), questo aspetto sarà approfondito meglio nella prossima sottosezione dedicata al preprocessing dei dati, nella quale saranno effettuate tutte le trasformazioni necessarie per pulire e rendere utilizzabili i dati per l'addestramento delle reti.

Il passaggio successivo, siccome si ha a che fare con un dataset con numerose configurazioni, è stato quello di calcolare la distribuzione delle diverse classi, ossia la distribuzione della colonna Status. Come accennato, nella prima sezione del capitolo, questa colonna può assumere tre diversi valori:

- Invalid: nel caso in cui la configurazione non rispetti vincoli fisici fondamentali per cui non è possibile calcolare il valore delle frequenze per la definizione delle band gap.
- OK: in questo caso, invece, la configurazione è valida e vengono calcolati i valori della frequenza di partenza e di arrivo della banda di stop.
- No BG: in quest'ultimo caso, invece, la configurazione è valida, quindi tutti i vincoli fisici sono rispettati, ma non esiste una banda di stop.

Essendoci, quindi, queste tre diverse casistiche è stata calcolata la distribuzione di ognuna di esse e ciò che è emerso sono le seguenti percentuali:

- Per la classe Invalid: 7057 configurazioni su 10000, ossia 70,6%.
- Per la classe OK: 2625 configurazioni su 10000, ossia 26,2%.
- Per la classe No BG: 318 configurazioni su 10000, ossia 3,2%.

Per rappresentare meglio questi valori e far capire in maniera più evidente il forte sbilanciamento di essi, sono stati realizzati due grafici riportati di seguito:

[15] La skewness è una misura che permette di calcolare la simmetria di una distribuzione di dati

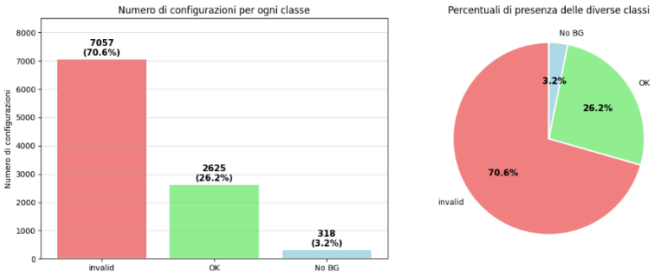


Figura 25: Grafico a barre e a torta che mostra la distribuzione delle classi

Come è evidente, è presente un forte sbilanciamento dei risultati, in quanto la maggior parte delle combinazioni di parametri di input non genera delle configurazioni valide. Questa problematica non si risolverebbe nemmeno nel caso in cui si andassero a raccogliere più dati con l'obiettivo di migliorare lo sbilanciamento ricercando più configurazioni valide.

Questo è un elemento fondamentale da tenere in considerazione in fase di addestramento del modello di ML, in quanto, questa osservazione, ha mostrato che è fondamentale assegnare correttamente i pesi alle diverse classi, in modo tale che la rete neurale non tenga conto solo di quella maggioritaria, ma riesca a prendere in considerazione anche quelle che presentano meno campioni.

Per andare ad indagare meglio i motivi dell'invalidità sono stati filtrati e contati i valori presenti all'interno della colonna Reason, dedicata proprio alla spiegazione del motivo di invalidità della configurazione. Tramite il seguente grafico è stata evidenziata la distribuzione delle motivazioni di invalidità delle configurazioni:

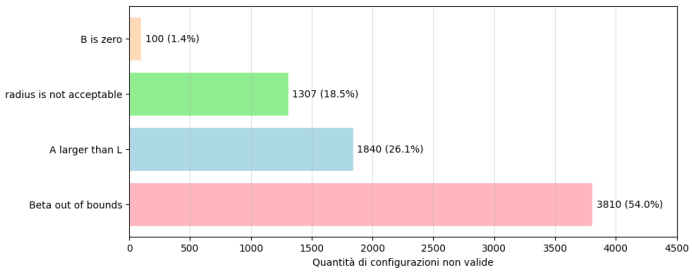


Figura 26: Grafico a barre che evidenzia la distribuzione della colonna Reason

Come si nota il dataset è formato da quattro motivi di invalidità, che il modello di machine learning ha dovuto imparare in fase di addestramento. Il più frequente è "Beta out of bounds", dove beta è l'angolo che si forma tra la trave piegata e la verticale.

Questa analisi più specifica è stata fondamentale anche per capire alcune situazioni di invalidità che si sono presentate durante l'addestramento delle reti neurali.

3.3.2 Influenza che i diversi valori hanno sulla stop band

Successivamente, per ognuno dei quattro valori di input e per ognuno dei valori in cui è stato diviso ogni parametro, è stato generato un grafico a barre per mostrare la distribuzione delle classi di stato.

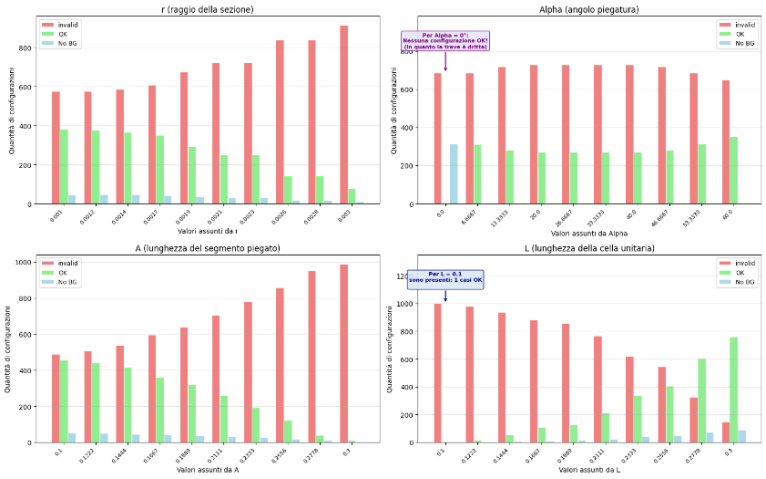


Figura 27: Grafico a barre che mostra la distribuzione degli status per i diversi parametri di input

Come è stato annotato, anche direttamente all'interno della figura, tramite questa analisi è stato possibile evidenziare alcune caratteristiche importanti dei dati, innanzitutto:

- Per Alpha: la distribuzione delle barre è pressoché identica per ogni valore di divisione del parametro. Un aspetto importante che è stato notato nel caso in cui l'angolo di piegatura sia pari a zero, è che, non esiste nessun caso valido, in quanto l'accoppiamento contra-direzionale si basa proprio sul concetto di piegatura, che, se viene a mancare, comporta l'assenza del fenomeno. Inoltre, all'aumentare dell'angolo, i casi validi aumentano, confermando la caratteristica importante che la piegatura favorisce la formazione di bande di stop.
- Per r: ciò che si è evidenziato è che all'aumentare del raggio della sezione, le configurazioni valide diminuiscono e, parallelamente, aumentano quelle invalide, confermando la caratteristica che all'aumentare della rigidezza del materiale, quindi, all'aumentare del raggio della sezione, è più rara la formazione di bande di stop.
- Per L: per questo parametro di input, insieme anche al prossimo, i casi sono strettamente collegati al vincolo fisico fondamentale di $A < L$. In figura è stato evidenziato il caso di $L = 0.1$, in quanto è presente una sola configurazione valida.
- Per A: per questo parametro, le distribuzioni hanno presentato esattamente l'andamento opposto rispetto al parametro L. Infatti, per il parametro A, all'aumentare del suo valore aumentano i casi invalidi e diminuiscono quelli validi. Questo sottolinea proprio il vincolo fondamentale di $A < L$.

Dato che dal grafico precedente risultava difficile capire l'andamento della distribuzione della colonna Status per il parametro Alpha, è stata effettuata un'analisi più specifica solo per questo valore, che ha evidenziato il seguente andamento:

- Per Alpha = 6,67° sono presenti 311 casi OK
- Per Alpha = 13,33° sono presenti 280 casi OK
- Per Alpha = 20,00° sono presenti 272 casi OK
- Per Alpha = 26,67° sono presenti 272 casi OK
- Per Alpha = 33,33° sono presenti 272 casi OK
- Per Alpha = 40,00° sono presenti 272 casi OK
- Per Alpha = 46,67° sono presenti 281 casi OK
- Per Alpha = 53,33° sono presenti 314 casi OK
- Per Alpha = 60,00° sono presenti 351 casi OK

Questa analisi ha permesso di mettere in luce, in maniera più chiara, l'andamento descritto, già in precedenza, ossia che i casi validi aumentano all'aumentare dell'angolo di piegatura, in particolare nei valori agli estremi, mentre, per quelli centrali, vi era una maggiore stabilità. Questo rappresenta perfettamente il concetto di accoppiamento contra-direzionale, descritto nel capitolo teorico relativo ai metamateriali, garantendo un accoppiamento tra modi d'onda flessurali e longitudinali. Quindi, Alpha è stato identificato come il parametro più determinante per la formazione di bande di stop.

Questa stessa analisi è stata fatta anche per il parametro r, quindi il raggio della sezione, il cui andamento riscontrato, è il seguente:

- Per r = 0,0010 sono presenti 380 casi OK
- Per r = 0,0012 sono presenti 377 casi OK
- Per r = 0,0014 sono presenti 367 casi OK
- Per r = 0,0017 sono presenti 350 casi OK
- Per r = 0,0019 sono presenti 291 casi OK
- Per r = 0,0021 sono presenti 249 casi OK
- Per r = 0,0023 sono presenti 249 casi OK
- Per r = 0,0026 sono presenti 143 casi OK
- Per r = 0,0028 sono presenti 143 casi OK
- Per r = 0,0030 sono presenti 76 casi OK

Quindi, all'aumentare del raggio della sezione, ossia all'aumentare della rigidezza del metamateriale, i casi validi scendono drasticamente, mentre per travi più sottili, e, quindi, più flessibili, sono garantiti valori di banda di stop maggiori.

Sono stati esaminati, più attentamente, solo questi due parametri perché sono quelli più rilevanti e importanti al fine della definizione delle bande di stop e, per i quali, nell'analisi generali, erano state evidenziate delle peculiarità che dovevano essere approfondite. Lo stesso approfondimento può, poi, essere svolto per i rimanenti due parametri, ma si tratta

di un'analisi che non è stata effettuata in quanto già dal grafico generale era ben nota ed evidente la distribuzione e l'andamento dei diversi casi.

Per andare ancora più a fondo, nell'analisi della correlazione tra valori di input e di output, è stato calcolato il coefficiente di correlazione di Pearson, valore compreso tra -1 e +1, che ha permesso di mettere in luce una relazione lineare, in particolare:

- +1 indica una correlazione positiva perfetta, ciò significa che: se un parametro di input aumenta, allora, di conseguenza, aumenterà anche quello di output.
- 0 indica assenza di correlazione tra i due valori.
- -1 denota una correlazione negativa perfetta, che al contrario di quella positiva, si traduce in un fenomeno per il quale: all'aumentare del parametro di input si ha una diminuzione dell'output generato.

Questo calcolo è stato effettuato solamente sulle configurazioni con status = OK e, poi, ogni coefficiente è stato rappresentato in un grafico a barre, che ha permesso di capire meglio la correlazione presente tra ogni parametro di input e i due possibili valori di output del modello:

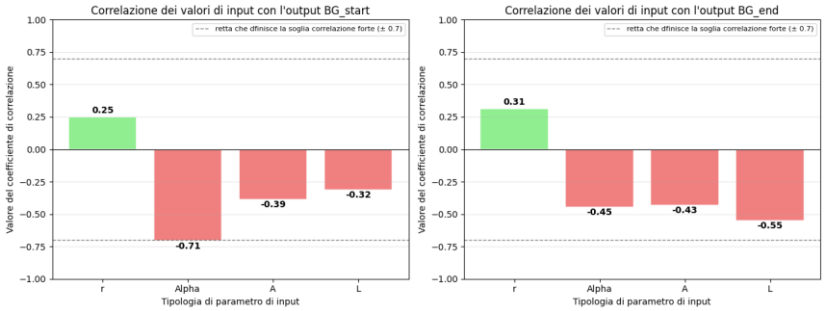


Figura 28: Grafico a barre che mostra la correlazione tra valori di input e valori di output

Ancora una volta è stato messo in luce come l'angolo di piegatura sia il parametro più determinante, il quale riesce, addirittura, a raggiungere la soglia di correlazione forte fissata a – 0,7, mentre il raggio della sezione è l'unico parametro che presenta correlazione positiva.

Da questo studio è stato possibile dedurre che il problema è non lineare e, come citato inizialmente nel capitolo, ha indirizzato la scelta della tipologia di algoritmo di machine learning, da utilizzare, verso le reti neurali.

3.3.3 Distribuzione delle frequenze di BG_start e BG_end

Successivamente, ci si è concertanti sulle distribuzioni relative agli output e non più agli input. Questi calcoli sono stati eseguiti solo sulle configurazioni classificate come OK,

quindi, quelle combinazioni di parametri per le quali esistono valori di output. Come citato a inizio capitolo, i valori di output sono:

- BG_start: valore di frequenza dove la banda di stop comincia.
- BG_end: valore di frequenza dove la banda di stop finisce.
- BG_width e BG_mean: che sono, rispettivamente, l'ampiezza e la media della frequenza delle bande di stop, essi sono calcolati direttamente dai primi due valori.

Per ognuno dei valori, appena elencati, è stato realizzato un istogramma, dove sull'asse delle ascisse è presente il valore della frequenza, in Hertz, mentre, sull'asse delle ordinate, è presente il numero di configurazioni che presentano quel determinato valore di frequenza. Ad ogni grafico sono poi state aggiunte una linea tratteggiata verde, che evidenzia la media e una linea tratteggiata blu, la quale, invece, mostra la mediana. Inoltre, per ogni dato di output è stata calcolata la Skewness, per avere un'indicazione numerica sulle distribuzioni.

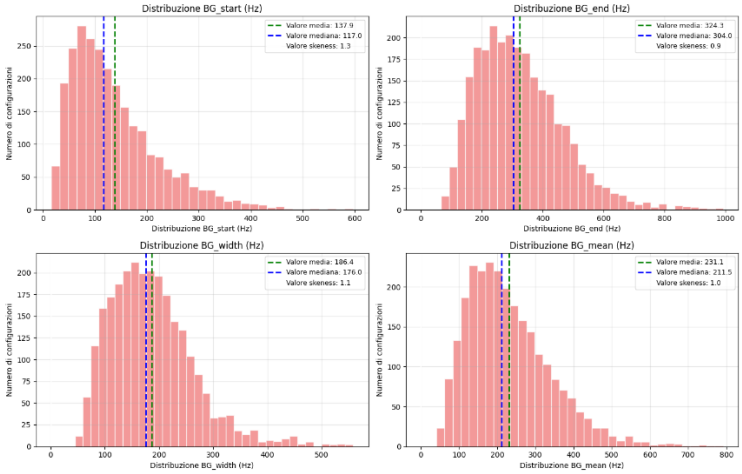


Figura 29: Grafici relativi alla distribuzione dei valori di output

Di seguito è riportata una tabella per un riassunto, più sintetico, dei dati mostrati nei quattro grafici della figura precedente:

	Min (Hz)	**Max (Hz)**	**Media (Hz)**	**Mediana (Hz)**	**Skewness**
BG_start	16	597	137,9	117,0	1,32
BG_end	66	992	324,3	304,0	0,94
BG_width	45	558	186,4	176,0	1,06
BG_mean	41	791.5	231,1	211,5	1,03

Figura 30: Tabella con valori riassuntivi delle distribuzioni dei valori di output

Dove:

- Min: indica il valore minimo di frequenza rilevato all'interno del dataset.
- Max: indica il valore più alto di frequenza rilevato.

Dai valori calcolati e mostrati nei grafici e nella tabella, si sono tratte alcune osservazioni:

- BG_start: ha una skewness, pari a 1,32, che denota un forte sbilanciamento verso sinistra e una media più alta rispetto alla mediana, fattore che evidenzia la presenza di valori alti, ma poco frequenti, quindi, una coda molto lunga verso la parte destra del grafico.
- BG_end: presenta, anch'essa, una skewness che mostra un'asimmetria simile a quella notata per la frequenza di inizio, quindi, anche in questo caso, ci sono maggiori valori accumulati verso sinistra. Un altro elemento importante, che è stato notato, è che i valori di frequenza di fine sono più alti rispetto a quelli di inizio, vincolo fisico molto importante, in quanto non ci può essere una configurazione che ha una frequenza di inizio più alta rispetto a quella di fine.

La forte asimmetria che è stata evidenziata, in entrambi i parametri di output, è stato un elemento fondamentale da osservare e da andare a sistemare, in fase di preprocessing, affinché la rete neurale riuscisse a funzionare correttamente e a ottenere buoni risultati.

Per BG_width e BG_mean, sono state fatte osservazioni simili a quelle già presentate per gli altri due valori, essendo dati che si ricavano direttamente da BG_start e BG_end.

Per mettere in luce, in maniera più evidente, la relazione tra BG_start e BG_end e, in particolare, mostrare che il valore del primo è sempre inferiore rispetto a quello del secondo, è stato realizzato uno scatterplot riportato nella seguente figura. In tale grafico è presente una linea tratteggiata rossa che mostra il caso in cui width = 0, ossia l'inizio della banda di stop è uguale alla fine, mostrando il caso degenere in cui la band gap è composta da un punto, mentre i punti blu rappresentano tutte le configurazioni che sono presenti all'interno del dataset.

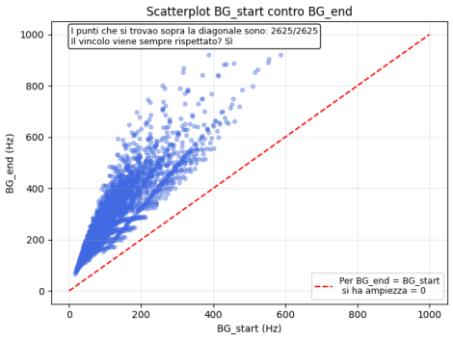


Figura 31: Scatterplot che evidenzia la relazione tra BG_start e BG_end

Come si nota tutti i punti si trovano sopra la linea tratteggiata rossa, confermando il fatto che in ogni configurazione ha BG_start < BG_end. Le configurazioni più rilevanti sono, sicuramente, quelle lontane dalla linea rossa, in quanto denotano un'ampiezza maggiore delle bande di stop, che le rende, quindi, in grado di smorzare maggiormente le onde elettromagnetiche. Dal grafico precedente è stato, anche, possibile dedurre che sono presenti numerose configurazioni che hanno un valore di BG_start compreso tra 10 Hz e 400 Hz e un valore di BG_end compreso tra 100 Hz e 600 Hz, mentre diventano più rare configurazioni con valori maggiori.

3.4 Fase 2: preprocessing del dataset

La fase precedente ha permesso di evidenziare peculiarità, correlazioni e distribuzioni all'interno del dataset grezzo generato direttamente da Matlab. Note tutte le caratteristiche, la fase successiva è stata quella di preprocessing, la quale, alla fine, ha permesso di ottenere un dataset pulito e pronto da fornire all'algoritmo di ML per la fase di addestramento.

In particolare, sono state eseguite alcune operazioni, che sono:

- Encoding dei valori della colonna "Status".
- Applicazione della trasformazione logaritmica ai valori di output per migliorarne la simmetria.
- Determinazione del dataset, per l'operazione di regressione e per quella di classificazione.
- Shuffling e divisione del dataset in training, validation e test set.
- Normalizzazione Min-Max dei parametri di input.
- Calcolo dei diversi class weights, in modo tale da riuscire a gestire il forte sbilanciamento tra le classi.

Queste operazioni saranno approfondite meglio nel corso di questa sezione. Anche per questa fase, come per la precedente, è stato utilizzato Google Colab. In particolare sono state importante le stesse librerie definite in precedenza, alle quali, però, sono state aggiunte:

- MinMaxScaler: necessaria per l'implementazione della normalizzazione Min-Max dei parametri di input.
- Compute_class_weight: libreria che permette di calcolare i pesi di ogni classe, in maniera inversamente proporzionale alla propria frequenza, fattore essenziale per gestire il forte sbilanciamento.
- Train_test_split: che permette di dividere il dataset nei 3 sotto-insiemi citati in precedenza.

3.4.1 Esecuzione dell'encoding delle variabili di input

Come definito anche nell'elenco precedente, dopo aver importato il dataset, la prima operazione che è stata eseguita è l'encoding delle classi presenti nella colonna Status. Questa è stata una trasformazione fondamentale, in quanto, le reti neurali lavorano solamente con numeri e non con stringhe. Quindi, siccome la colonna Status, che è fondamentale al fine della predizione delle bande di stop, contiene stringhe, è stata necessaria la trasformazione in valori numerici.

Di conseguenza, all'interno del dataset è stata generata una nuova colonna chiamata "Status_encoded" la quale contiene i valori numerici codificati seguendo la struttura:

- Classe OK ha codice numerico 0.
- Classe No BG ha codice numerico 1.
- Classe invalid ha codice numerico 2.

Questa trasformazione è stata possibile tramite la funzione map() di pandas, dopo aver costruito un dizionario che mappasse la struttura citata prima. Parallelamente è stato generato anche un dizionario inverso, che consentisse di eseguire l'operazione opposta, ovvero, convertire il codice numerico nel corrispondente valore stringa. Rilevandosi un elemento fondamentale durante la fase di valutazione delle prestazioni, in quanto, ha permesso di rendere i risultati più intuitivi e facilmente interpretabili. Risulta, infatti, poco immediato comprendere e ricordarsi a memoria i numeri associati a ciascuna classe, nonostante il numero sia limitato, mentre, l'utilizzo delle relative etichette in formato stringa facilita notevolmente l'interpretazione dei risultati. Per questo motivo è stato realizzato, anche, il dizionario inverso. La rete, quindi, dovrà essere in grado di predire questo valore numerico partendo dai valori di input forniti.

Per verificare che l'encoding sia avvenuto correttamente, nella parte finale della cella di codice, è stata introdotta una verifica numerica della quantità di configurazioni che ricadono all'interno di ogni classe, che ha permesso di dimostrare, che i valori e le suddivisioni si sono mantenute costanti, evidenziando la corretta applicazione dell'encoding.

3.4.2 Trasformazione logaritmica dei valori delle frequenze

Come osservato nella fase di analisi esplorativa del dataset, uno dei problemi maggiori è risultato essere la forte asimmetria nelle distribuzioni dei valori di output del modello, caratteristica che è possibile notare anche direttamente dal valore assunto dalla skewness, la quale è risultata essere fortemente positiva. Per evitare problemi in fase di addestramento delle reti neurali, è stato necessario applicare una trasformazione logaritmica in base 10 (log10), che è stata effettuata solo sui valori di BG_start e BG_end, in quanto sono gli output che la rete deve predire, mentre BG_width e BG_mean, si ricavano direttamente dai due dati precedenti.

È stata scelta una trasformazione logaritmica, in quanto, le due dimensioni dei valori di output variano su range che sono molto differenti tra loro, ma, grazie a questa trasformazione, è possibile far in modo che abbiano lo stesso peso rispettando la struttura fisica. L'MSE, quindi, viene calcolato come percentuale e non come valore assoluto, facendo in modo che un errore percentuale del 15% abbia lo stesso peso indipendentemente dal valore assoluto al quale è associato, riuscendo così a eliminare il problema relativo alla differenza di range di valori delle due distribuzioni. Inoltre, si è scelto di adottare la trasformazione logaritmica, rispetto ad altre possibili trasformazioni, in quanto la relativa operazione inversa, ovvero il passaggio dal valore ottenuto mediante log10() al corrispondente valore in Hz, risulta semplice e intuitivo. È, infatti, sufficiente applicare la seguente operazione:

$$\text{Frequenza in HZ} = 10^{\text{valore predetto dalla rete}}$$

Equazione 32: Formula per il passaggio da scala logaritmica a valore in Hz

Dopo aver effettuato la trasformazione logaritmica, sono stati realizzati alcuni istogrammi che hanno mostrato la nuova distribuzione dei valori di frequenza di inizio e di fine della banda di bloccaggio, i quali sono stati messi a confronto con i grafici realizzati in fase di analisi esplorativa, riportati nella figura di seguito. Da questa trasformazione si sono ottenuti i risultati che ci si aspettava, ossia una riduzione della skewness, con conseguente aumento della simmetria, oltre al valore della media e della mediana che sono diventati praticamente coincidenti. Osservando il grafico si nota come, dopo la trasformazione, la curva sia molto più simmetrica, non presentando più la coda verso destra, con tutti i valori alti concentrati nella parte sinistra del grafico.

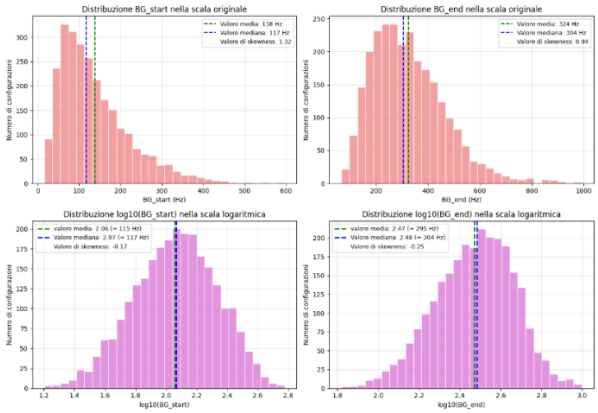


Figura 32: Istogramma rappresentante la distribuzione dei valori di output prima e dopo la trasformazione logaritmica

Andando più nello specifico, dal grafico si è osservato che la skewness:

- Per BG_start passa da 1,323 a -0,172, con una riduzione del 87%. Con una differenza tra media e media che passa da 15,2% a 0,4%.
- Per BG_end passa da 0,942 a -0,250, con una riduzione del 73%. Con una differenza tra media e media che passa da 6,2% a 0,5%.

Questi valori hanno permesso di dimostrare un miglioramento della simmetria.

Per quanto riguarda gli intervalli di valori di frequenza, invece, diventano:

- Da 1,204 a 2,776 per la scala logaritmica di BG_start, quando inizialmente era [16; 597] Hz.
- Da 1,820 a 2,997 per la scala logaritmica di BG_end, quando inizialmente era [66; 992] Hz.

Come si nota, quindi, dopo la trasformazione i valori di range, entro cui variano le frequenze, sono risultati essere molto più uniformi.

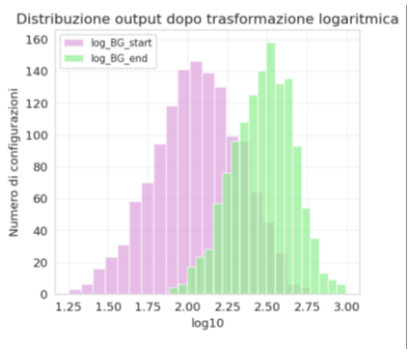


Figura 33: Distribuzione valori di output dopo la normalizzazione logaritmica

Come si nota dalla figura precedente, nella quale sono riportate, nello stesso grafico, le distribuzioni della trasformazione logaritmica di BG_start e BG_end, ora risultano essere molto più simmetriche, con i valori di BG_end più grandi rispetto a quelli di BG_start.

3.4.3 Regole per la divisione del dataset

L'algoritmo di ML che è stato generato è composto da due componenti principali, i quali hanno scopi completamente differenti:

- Classificatore: il quale utilizza l'intero il dataset e ha, come suggerisce il nome, l'obiettivo di classificare le configurazioni, quindi, deve essere in grado di riconoscere le tre diverse classi (OK, No BG e Invalid). Come si può dedurre, quindi, la rete neurale creata per lo svolgimento di questo task è stata addestrata su tutte le configurazioni, indipendentemente dalla classe di appartenenza, in

modo tale che imparasse le caratteristiche e i pattern presenti per effettuare la scelta corretta.

- Regressore: in questo caso, invece, l'obiettivo è quello di riuscire a predire i valori di BG_start e BG_end, motivo per il quale la rete neurale, che è stata generata per la realizzazione di questo compito, è stata addestrata, solamente, sulle configurazioni del dataset che presentano Status = OK, ossia quelle configurazioni per le quali esiste una banda di bloccaggio delle onde elettromagnetiche, infatti, è stato evidenziato che è inutile fornire alla rete configurazioni per le quali non esiste la banda di inibizione.

Quindi, avendo obiettivi differenti, il classificatore e il regressore sono stati addestrati su dati diversi, come citato in precedenza, motivo per il quale la fase successiva, eseguita durante il preprocessing, è proprio la divisione del set di dati per i diversi scopi.

Come prima cosa, quindi, sono stati creati quattro oggetti differenti:

- x_class: che corrisponde agli input del classificatore, ossia 4 colonne per 10.000 righe, in quanto sono state utilizzate tutte le configurazioni esistenti.
- y_class: che corrisponde agli output del classificatore, ossia una sola colonna per tutte le 10.000 righe.
- x_reg: che corrisponde agli input del regressore, che, in questo caso, sono sempre le 4 colonne, corrispondenti ai diversi parametri di input, ma, per il regressore, sono state utilizzate solo le 2625 righe con configurazioni che presentano status = OK.
- y_reg: che corrisponde agli output del regressore e, in questo caso, il sottoinsieme creato è formato da 2 colonne, che sono i due valori di output che la rete deve predire e 2625 colonne, quindi per ogni configurazione, con status = OK, la rete deve predire i valori di band gap.

Una cosa importante da sottolineare è che il set di dati di partenza, che è stato usato per definire i sottinsiemi di dati per il classificatore e per il regressore, è lo stesso, semplicemente, per il regressore sono state selezionate solo le configurazioni con Status = OK, ma non è stato generato un secondo dataset.

Nel momento in cui si va a definire un algoritmo di ML, una delle operazioni più complicate e, allo stesso tempo delicate, è la divisione del dataset in training, validation e test set. Come descritto nel secondo capitolo esistono diverse tecniche per la divisione del set di dati, che partono tutte dall'osservazione dei valori che si ha a disposizione, nello specifico, per il set di dati generato dal codice Matlab si sono notati alcuni problemi che è stato necessario gestire. Innanzitutto, i dati sono organizzati in ordine in base al valore dell'angolo di piegatura; ciò significa che le prime righe sono tutte le possibili configurazioni, nate con il prodotto scalare, con Alpha = 0, le successive hanno tutte Alpha = 6.67, e via così fino ad arrivare a Alpha = 60. Se il dataset fosse stato, semplicemente, diviso prendendo le prime righe e assegnandole al training set, le successive al validation e le rimanenti al test set, si sarebbe rischiato che la rete vedesse solo alcuni valori, e, di conseguenza, non riuscisse a generalizzare correttamente,

imparando bene solo i pochi valori che era riuscita a vedere. Inoltre, un altro problema che è stato riscontrato nel dataset, è che le configurazioni valide, che, quindi, presentano Status = OK, sono tutte organizzate in blocchi vicini tra di loro e non distribuiti uniformemente in tutto il set di valori, motivo per il quale, si rischiava, se non si fosse risolto questo problema, di creare sottoinsiemi di dati, senza, o con poche, configurazioni con Status = Ok, il che avrebbe limitato l'addestramento della rete.

Per ovviare a questi problemi è stata utilizzata una tecnica formata da due attività principali:

- shuffling: come prima cosa il dataset è stato mescolato in maniera casuale, con un seed fisso così da garantirne la riproducibilità, in caso fosse stato necessario eseguire più volte il training. Questa operazione ha fatto in modo di eliminare il primo problema che si è citato, in quanto, successivamente al mescolamento, le configurazioni erano organizzate in modo completamente random, senza seguire un pattern preciso.
- Interleaving con pattern 5-2-3: una volta eseguito il mescolamento, è stato seguito un pattern per la selezione e divisione delle configurazioni su blocchi di 10 righe consecutive. In particolare, è stato seguito il meccanismo descritto di seguito:
    - Le righe che occupano la posizione {0; 1; 2; 3; 4}, secondo %10, sono state assegnate al training set, quindi 5 righe su 10, ossia il 50%.
    - Le righe che occupano la posizione {5; 6}, secondo %10, sono state assegnate al validation set, quindi 2 righe su 10, ossia il 20%.
    - Le righe che occupano la posizione {7; 8; 9}, secondo %10, sono state assegnate al test set, quindi 5 righe su 10, ossia il 30%.

L'interleaving è risultato necessario, al posto di un semplice taglio sequenziale, per garantire che ognuno dei nuovi sottoinsiemi ricevesse la stessa distribuzione delle classi, infatti, un problema, che con lo shuffling casuale si sarebbe potuto verificare, è che tutte le configurazioni, con la stessa tipologia di classe, potessero essere raggruppate vicine, caso molto raro, ma possibile, soprattutto nel caso della classe più numerosa, ossia Invalid. L'interleaving, quindi, era l'unico metodo che garantisse di superare, eventualmente, questa problematica.

Fatta la divisione si sono ottenuti i seguenti risultati:

	Percentuali	**Classificatore**	**Regressore**
Training	50%	5.000 campioni	1.315 campioni
Validation	20%	2.000 campioni	524 campioni
Test	30%	3.000 campioni	786 campioni

Figura 34: Divisione training, validatione test set

Come si nota, il numero di campioni presenti all'interno dei set di dati per il regressore sono sempre inferiori rispetto a quelli per il classificatore, in quanto sono state selezionate solo le configurazioni che presentano una banda di stop.

Come descritto anche nella parte più teorica della tesi, dedicata al Machine Learning, il training set serve per il vero e proprio aggiornamento della rete, l'algoritmo vede questi dati che utilizza per aggiornare i pesi. Il validation set, invece, serve per il monitoraggio e per la selezione dei corretti parametri in fase di addestramento, infine, il test set è un insieme di dati che l'algoritmo vede solo alla fine, questi dati non sono mai utilizzati in fase di addestramento, ma vengono forniti alla rete solo nel momento in cui è ritenuta definitiva, per effettuare una valutazione finale oggettiva sul funzionamento e le prestazioni della rete, in caso questi dati venissero utilizzati anche in fase di addestramento si rischierebbe di ottenere una valutazione della performance positiva, ma poco rappresentativa.

Per verificare che la suddivisione fosse avvenuta in maniera corretta è stato realizzato un grafico a barre, il quale ha permesso di evidenziare che le proporzioni definite sono state mantenute anche dopo la divisione, in particolare si è notato che:

- Classe OK:
  - 50,3% training set.
  - 20,4% validation set.
  - 29,3% test set.

- Classe No BG:
  - 50,9% training set.
  - 15,7% validation set.
  - 33,3% test set.

- Classe invalid:
  - 49,8% training set.
  - 20,1% validation set.
  - 30,1% test set.

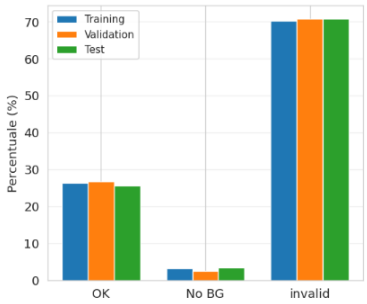


Figura 35: Istogramma rappresentate la divisione delle classi nei diversi set dati

Come si nota le percentuali, che sono state fissate, sono state rispettate, l'unica classe che presenta qualche differenza è la classe No BG, questo leggero discostamento è dovuto alla bassa numerosità di configurazioni classificate come tali. Comunque, ciò che è stato importante dimostrare, è che fossero presenti, nei tre diversi sottoinsiemi, le stesse

percentuali di configurazioni con la stessa classe, per garantire il corretto addestramento della rete neurale dedicata alla classificazione.

Un'analisi più specifica è stata, successivamente, condotta sul parametro Alpha. Come anticipato all'inizio della sottosezione, una delle principali problematiche riscontrate nel dataset di partenza, era, infatti, la sua organizzazione secondo un ordine crescente del valore dell'angolo di piegatura. Per verificare, quindi, che lo suffling e l'interleaving abbiano funzionato correttamente è stata studiata la distribuzione dei diversi valori di Alpha nei subset di dati generati. Anche in questo caso si sono andati a rappresentare i risultati ottenuti all'interno di un grafico, per far in modo che essi risultassero più intuitivi:

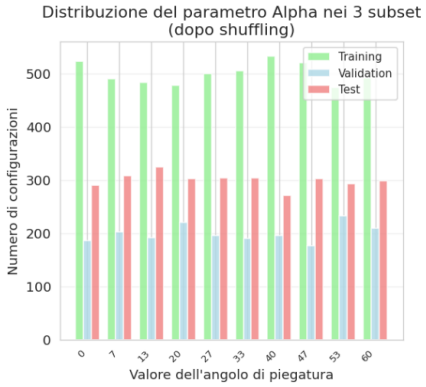


Figura 36: Grafico che rappresenta la distribuzione dei diversi valori di Alpha nei subset creati

Come si nota dalla figura, per i 10 valori di Alpha, pari al numero di divisioni fissate, le colonne, che descrivono il numero di campioni presenti nei diversi set dati generati, sono molto similari tra loro, denotando una corretta divisione delle configurazioni tra i 3 diversi set dati. Questo fa sì che la rete riesca a vedere diversi valori del parametro e non solo alcuni e, soprattutto, abbia, per ogni valore dell'angolo di piegatura, lo stesso quantitativo di configurazioni, così che non abbia la tendenza di imparare meglio alcuni valori, piuttosto che altri.

3.4.4 Normalizzazione di tipo Min-Max dei valori di input

Un altro elemento caratteristico del dataset, che è stato notato in fase esplorativa, è che i parametri di input variano su intervalli che sono molto differenti tra loro, in particolare:

- r: ha valori compresi tra 0,001 e 0,003, quindi, con un ordine di grandezza pari a 10^{-3}.
- Alpha: ha valori compresi tra 0 e 60, quindi, con un ordine di grandezza pari a 10^{1}.
- A: ha valori compresi tra 0,1 e 0,3, quindi con un ordine di grandezza pari a 10^{-1}.
- L: ha valori compresi tra 0,1 e 0,3, quindi, con un ordine di grandezza pari a 10^{-1}.

Avendo valori così differenti la problematica principale riguarda il fatto che la rete rischierebbe di concentrarsi a predire correttamente solo il parametro con l'ordine di grandezza maggiore, ignorando completamente gli altri, quindi, è stata applicata la normalizzazione Min-Max, che ha permesso di riportare tutti i parametri all'interno dell'intervallo [0; 1], per far in modo di uniformare i parametri. La formula usata è la seguente:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Equazione 33: Formula per la normalizzazione Min-Max

È stata scelta questa tipologia di normalizzazione in quanto si conoscono, dettagliatamente, i valori dei parametri di input, inoltre, tali valori, seguono una distribuzione uniforme, queste due caratteristiche sono elementi fondamentali e caratteristici per la normalizzazione Min-Max, a differenza, per esempio, della Z-score la quale preferisce distribuzioni gaussiane, dove media e deviazione standard descrivono bene i dati. Questo metodo si rivelerà, inoltre, fondamentale nella, successiva, fase di ottimizzazione inversa, durante la quale verranno generate un elevato numero di configurazioni, che dovranno essere opportunamente normalizzate, mediante il metodo Min-Max. Tale processo presenta il vantaggio di essere completamente deterministico e di consentire la normalizzazione di valori anche al di fuori dell'intervallo del training set. Al contrario, nel caso dello Z-score sarebbe necessario prestare particolare attenzione alla normalizzazione di valori non presenti nel training set.

Quando si applica questo tipo di normalizzazione, una delle problematiche principali che può essere riscontrata è, sicuramente, il data leakage. Per ovviare a questo problema è fondamentale prestare attenzione al fitting dello scaler, per questo i parametri di normalizzazione, quindi il minimo e il massimo, sono stati calcolati esclusivamente sul training set del classificatore e, poi, applicati al validation e al test set. Se si fossero calcolati i parametri sull'intero set di dati, si sarebbe rischiato di ottenere valutazioni di performance fortemente ottimistiche. I parametri sono stati calcolati direttamente sui dati del classificatore, ma, poi, applicati anche al regressore, in modo tale da ottenere una normalizzazione coerente.

Infine, la normalizzazione effettuata è stata salvata all'interno di un file .csv dedicato, così che potessero essere utilizzati i dati identici, e senza necessità di dover essere ricalcolati, anche nelle fasi successive.

3.4.5 Definizione delle class weights per il classificatore

L'ultima attività svolta nella fase di preprocessing dei dati è stata la definizione dei class weights corretti, infatti, un altro elemento emerso in fase esplorativa del dataset, e più

volte riportato nel corso della trattazione fino a questo momento, è il forte sbilanciamento tra le tre diverse classi, in cui possono essere suddivise le configurazioni. In particolare:

- 70,6% per la classe invalid.
- 26,3% per la classe OK.
- 3,2% per la classe No BG.

Come già affermato in precedenza, se non si fosse effettuata una correzione, la rete si sarebbe concentrata a predire solamente la classe invalid, ossia la più frequente, ignorando completamente le altre, senza imparare nulla.

Per correggere questa problematica è stato importante definire correttamente i class weights, che sono pesi (o penalità) che vengono moltiplicati per l'errore commesso dalla rete, per ogni campione, in fase di training, quindi, vengono utilizzati direttamente nel calcolo della loss function. In particolare, i class weights, in PyTorch, vengono passati alla loss function CrossEntropyLoss

In questa fase è risultato importante assegnare pesi maggiori alle classi meno frequenti e, al contrario, pesi inferiori alle classi meno rare. Questa strategia ha consentito di assegnare penalizzazioni maggiori agli errori che venivano commessi, dalla rete, sulle classi meno frequenti e, parallelamente, ridurre il peso degli errori su classi più numerose. Infatti, se non venissero assegnati pesi differenti, la rete tenderebbe a predire sempre la classe maggioritaria, ossia Invalid, in quanto otterrebbe una minimizzazione dell'errore, nonostante i risultati della prestazione sarebbero di bassa qualità, in quanto predirebbe sempre una sola classe, sbagliando. Per ovviare a questo problema diventa necessario pesare maggiormente gli errori commessi su classi meno frequenti, così che se la rete sbagliasse, l'errore peserebbe molto di più. La formula che è stata seguita per decidere quali pesi assegnare alle diverse classi è la seguente:

$$\text{weight[classe]} = \frac{\text{campioni totali}}{\text{numero di classi} * \text{campioni per la classe analizzata}}$$

Equazione 34: Formula per la definizione die classe weights

Dove:

- Campioni totali si intende il numero totale di configurazioni presenti all'interno del training set, che, come visto nelle sotto-sezioni precedenti, è pari a 5.000 configurazioni. Anche in questo caso, sono state usate solo le configurazioni presenti all'interno del training set, per il calcolo dei pesi, per evitare il problema, già citato in precedenza, dei risultati troppo ottimistici, ma poco significativi, che si sarebbero ottenuti se si fosse utilizzato l'intero dataset per il calcolo.
- Il numero di classi è pari a 3, ossia OK, No BG e Invalid.
- Campioni per la classe in analisi sono pari a: 1321 per classe OK, 162 per classe No BG e 3517 per la classe invalid.

Applicando la formula precedente, i pesi ottenuti, per ogni classe, sono i seguenti:

	Campioni training set	**Percentuale**	**Weight**
OK	1.321	26,4%	1,2617
No BG	162	3,2%	10,2881
invalid	3.517	70,3%	0,4739

Figura 37: tabella riassuntiva dei class weights

Come si nota sono stati assegnati pesi maggiori alle classi più rare e pesi più bassi a classi più frequenti. Nello specifico la classe No BG, per la quale è stato evidenziato un numero ridotto di configurazioni, presenta un peso che è, quasi, 22 volte superiore al peso della classe maggioritaria, ciò significa che: qualora la rete sbagliasse una predizione su questa classe, produrrebbe un errore 22 volte maggiore rispetto a quello che produrrebbe sulle altre classi, che sono più frequenti.

Per evidenziare ancora meglio la differenza, presente tra i valori dei pesi, è stato realizzato anche un grafico a barre, riportato di seguito:

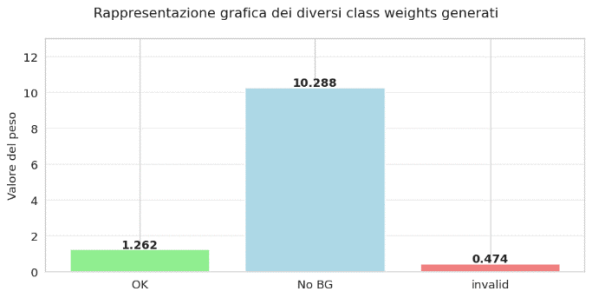


Figura 38: Grafico per la rappresentazione grafica dei classe weights delle diverse classi

In conclusione a questa fase di preprocessing tutti i dati, che sono stati lavorati e modificati, sono stati salvati in appositi file .csv, attraverso la libreria os, in modo tale da poterli utilizzare liberamente e facilmente nelle fasi successive. Sono, dunque, stati generati:

- File relativo alla normalizzazione degli input.
- File relativo alla divisione tra training, validation e test set per classificatore e regressore.
- File contenente i dati di output normalizzati tramite trasformazione logaritmica.
- File contenente i dati di input codificati.
- Salvataggio dei class weights.

3.5 Fase 3: training della rete

Le prime due fasi (esplorativa e preprocessing dei dati) sono state, sicuramente, molto corpose, ma la corretta preparazione e analisi del dataset è una delle condizioni cardine e fondamentale al fine del corretto funzionamento degli algoritmi di machine learning. Risulta, infatti, inutile addestrare una rete neurale su dei dati grezzi, non lavorati e poco significativi, in quanto si otterrebbero performance scarse, è, invece, necessario addestrare gli algoritmi su dati puliti, preparati in base alle esigenze dell'algoritmo e del problema stesso, oltre allo scopo che si vuole raggiungere.

Terminata la fase di definizione del dataset, è iniziata la fase centrale del progetto, ossia la definizione, addestramento e valutazione della performance del modello di machine learning. Ribadendo l'obiettivo, attorno al quale si sviluppa tutto l'elaborato, è la generazione di un algoritmo di ML surrogato, il quale, partendo da dei valori di input (r, Alpha, A e L) deve essere in grado, in pochissimi secondi, di predire le frequenze di partenza e di arrivo delle bande di stop, dalle quali è possibile, poi, calcolare la profondità e la media, riuscendo a superare la problematica principale del codice Matlab, ossia la lentezza computazionale.

Questa fase che verrà definita training, come le precedenti, è stata sviluppata tramite un notebook di Google Colab, il quale si è articolato in 3 tentativi successivi che hanno permesso la generazione della corretta architettura. All'interno del notebook dedicato all'addestramento delle reti neurali, oltre alle librerie citate ed utilizzate anche per le due fasi precedenti, sono state importate:

- PyTorch: libreria principale per la costruzione e addestramento delle reti neurali.
- nn: modulo di PyTorch contenente tutti i componenti necessari per la costruzione di reti neurali, come strati linear, funzioni di attivazione e di loss, ecc.
- Optim: modulo di PyTorch contenente tutti gli ottimizzatori utilizzati per le reti neurali.
- Time: libreria che permette di tenere traccia del tempo impiegato da un algoritmo per svolgere il proprio task. Libreria fondamentale per tenere monitorate le tempistiche impiegate dalla rete e verificare il miglioramento rispetto al codice Matlab.
- DataLoader: libreria dedicata alla gestione dei mini-batch di dati che saranno forniti alle reti in fase di addestramento.
- TensorDataset: permette la definizione dei tensori, ossia array di valori che sono utilizzati per manipolare e gestire i dati, che saranno utilizzati da PyTorch all'interno dell'algoritmo per la gestione del dataset.

3.5.1 Primo tentativo: architettura con backbone condiviso per classificazione e regressione

Il primo tentativo che è stato eseguito è quello di realizzare un modello multi- task, in quanto, l'algoritmo doveva essere in grado di svolgere sia l'attività di classificazione che quella di regressione, con un backbone condiviso, ciò significava realizzare un'unica rete

neurale che fosse in grado di svolgere entrambe le attività. Questa architettura puntava a sfruttare la rappresentazione interna comune, ossia che ad entrambe le operazioni servissero le stesse conoscenze fisiche di base, e, quindi, attraverso un backbone condiviso non vi fosse la necessità di costruire una doppia struttura, ma la rete doveva limitarsi a imparare le caratteristiche e pattern più importanti e, soltanto alla fine, raffinare le informazioni in base o al compito di classificazione o a quello di regressione.

L'architettura creata per questo tentativo è composta da:

- Il layer dedicato agli input, formato da 4 neuroni, che corrispondono a i 4 parametri di input del problema Alpha, r, A e L, che sono gli stessi sia per il task di regressione che per quello di classificazione.
- Per quanto riguarda gli hidden layer è stata scelta una architettura con struttura a diamante formata da [64, 128, 128, 64] nodi completamente connessa. L'architettura a diamante è stata preferita in quanto è una struttura adatta per imparare rappresentazioni sempre più astratte e complesse allargandosi, per poi riuscire a dettagliarle e sintetizzarle meglio stringendo la struttura e generando l'output finale. Quindi, all'inizio vengono imparate le caratteristiche più semplici, poi, negli strati successivi la rete si concentra ad imparare relazioni sempre più complicate, fino a sintetizzare tutti i risultati per ottenere un valore finale che deve essere passato all'ultimo strato. Questa struttura, infatti, viene spesso utilizzata per problemi non lineari, come il problema affrontato per il progetto di tesi. Il numero di strati, come citato anche nel capitolo teorico relativo al machine learning, è un numero fondamentale da definire per evitare il problema dell'overfitting, inizialmente, quindi, è stato fissato a 4 strati, i quali possono essere aumentati o diminuiti in caso venissero rilevati dei problemi.
- Ogni strato è composto da: Linear + BatchNorm1d + ReLU + Dropout(). Questa struttura è stata scelta perchè:
    - Linear: permette di calcolare l'output, attraverso la formula:

$$output = W * input + b$$

Equazione 35: Equazione strato lineare

      Con W è stata indicata la matrice contenente i pesi, mentre con b il vettore relativo ai bias che la rete impara durante la fase di addestramento.
    - BatchNorm1d: consente di normalizzare la distribuzione degli output di ogni strato, calcolandone media, che deve essere uguale a zero, e varianza prossima al valore unitario. Questo permette di favorire la convergenza dell'algoritmo in fase di addestramento e permette di utilizzare valori di learning rate più alti. Inoltre, fa sì che si possa controllare il fattore per il quale i valori crescono, o decrescono, in maniera incontrollata, man mano che l'informazione fluisce in avanti nei diversi strati. Durante l'addestramento, infatti, la distribuzione dei valori di output, di ogni strato, cambia costantemente a causa dell'aggiornamento dei pesi, rendendo la fase di addestramento molto instabile. Se non si effettuasse la normalizzazione, ogni strato dovrebbe adattarsi, costantemente, a

distribuzioni di input che sono sempre differenti, ciò instabilità e lentezza nella convergenza.

- ReLU: la scelta di questa funzione di attivazione riguarda il fatto che essa garantisce di poter introdurre il concetto di non linearità presente all'interno del problema e per far in modo di riuscire a rappresentare la relazione fisica che il problema presenta. Se non venisse utilizzata, è come se la rete neurale fosse composta da un unico strato lineare in grado di calcolare l'output. Inoltre, come descritto anche nel capitolo dedicato questa funzione, essa garantisce di evitare il problema del vanishing gradient, inoltre, risulta essere computazionalmente molto efficiente, motivo per il quale è stata selezionata tra le varie possibili funzioni di attivazione citate nel secondo capitolo del lavoro di tesi.
- Dropout(0,2): fissando il valore di dropout a 0,2, significa che vengono disattivati il 20% dei neuroni ad ogni forward pass eseguito durante la fase di training. Disattivando i neuroni, tutti i loro parametri vengono azzerati, ciò fa sì che essi non concorrano più al calcolo dei valori di output o all'aggiornamento dei pesi. Il motivo dell'introduzione di questo spegnimento dei neuroni dipende dal fatto che si vuole evitare che la rete dipenda esclusivamente da alcuni neuroni o da alcuni percorsi che i dati compiono, per questo motivo, in maniera casuale, alcuni di essi vengono disattivati, per aumentare la robustezza del modello. Questo è un parametro il cui valore deve essere modificato nel caso in cui si presenti overfitting. Durante la fase di valutazione della performance, invece, sarà disattivato completamente.

- Infine, per il livello di output, vi è una doppia testa:
  - Una dedicata alla classificazione con uno strato Linear(64, 3), in grado di produrre 3 valori di output, corrispondenti alle 3 classi in cui vengono divise le configurazioni.
  - Una dedicata alla regressione con uno strato Linear(64, 2), in grado di produrre 2 valori di output, corrispondenti alle trasformazioni logaritmiche di BG_start e BG_end.

  Sia per il classificatore che per il regressore non sono state applicate funzioni di attivazione, nel primo caso perché la CrossEntropyLoss la applica già internamente, mentre per il secondo task, deve risultare possibile predire valori continui senza imposizione di vincoli.

In questo tentativo a backbone condiviso, la funzione di loss totale viene calcolata con la seguente formula:

$$\text{Loss function totale} = \text{loss}_{\text{class}} + \text{lambda} * \text{loss}_{\text{regr}}$$

Equazione 36: Equazione per il calcolo della loss function totale nel primo tentativo

Quindi, per il classificatore viene utilizzata, come citato sopra, la CrossEntropyLoss, con i class weights differenziati per le singole classi. Questa è la funzione di loss più

largamente utilizzata per problemi multi classe e permette di calcolare, per ogni configurazione, quanto i valori predetti siano lontani dal valore reale. Come detto, essa utilizza i pesi assegnati alle varie classi in fase di preprocessing. I pesi vengono moltiplicati per l'errore commesso dalla rete, per ogni configurazione, ciò fa sì che gli errori commessi dalla rete sulle classi più rare, come per esempio No BG, siano molto più penalizzanti.

Mentre, per il task dedicato alla regressione è stata utilizzata, come loss function, MSELoss, la quale permette di calcolare l'errore quadratico medio (Mean Squared Error – MSE), tra i valori che la rete ha predetto e quelli reali. In particolare questo calcolo avviene sui valori che hanno subito la trasformazione logaritmica e non sui valori in Hertz. Questo è un dettaglio fondamentale perché la rete punta a minimizzare errori percentuali e non in valore assoluto. È stato scelto MSE in quanto tende, essendo un errore quadratico, a penalizzare maggiormente gli errori grandi.

Tramite l'utilizzo della formula riportata sopra è, quindi, stato possibile calcolare il valore della funzione di perdita totale, tenendo conto sia della loss function del regressore che di quella del classificatore.

Aggiungendo, poi, un parametro lambda pari a 0,5, è stato possibile bilanciare l'importanza di entrambi i tasks, quindi, in questa maniera, la classificazione presenta il doppio del peso rispetto alla regressione. Lambda è una costante il cui valore può essere modificato in relazione al comportamento presentato dai due task in fase di valutazione delle prestazioni.

L'ottimizzatore scelto, invece, è Adam, con un learning rate iniziale pari a 0,001, esso è un valore consolidato dalla letteratura e largamente utilizzato nella pratica. Il valore di 0,001 garantisce un rapido miglioramento nelle prime epoche, ma, al tempo stesso, è un valore sufficientemente basso da non destabilizzare troppo la fase di addestramento. Questa tipologia di ottimizzatore permette di adattare il learning rate in base alla storia dei gradienti, quindi fa sì che parametri che presentano gradienti maggiori, abbiano aggiornamenti più piccoli e, viceversa, per i parametri con gradienti più piccoli. Inoltre, l'Adam garantisce una riduzione delle oscillazioni, grazie all'utilizzo della media mobile dei gradienti passati.

A questo ottimizzatore è stato associato, anche, uno scheduler ReduceLROnPlateau, il quale permette di dimezzare (factor = 0,5) il learning rate dopo 10 epoche senza miglioramenti (patience = 10), in particolare tramite questo metodo è stato possibile utilizzare un learning rate più alto nelle prime epoche, per velocizzare la convergenza e raggiungere in maniera più rapida una zona corretta dello spazio, mentre nelle fasi finali, per una maggiore precisione, è stato utilizzato un valore più piccolo.

Inoltre, è stato introdotto anche un meccanismo di early stop, con una patience pari a 50 epoche, quindi, se la validation loss non migliora per 50 epoche consecutive, allora l'addestramento si blocca e viene salvato il miglior modello trovato fino a quel momento, questo fa in modo che si eviti di continuare ad addestrare un algoritmo che ha già raggiunto l'ottimo.

In fase di addestramento, inoltre, è stata limitata la norma del gradiente, attraverso un clipping dei gradienti stessi. Questo ha permesso di evitare il problema del gradient

exploding, il quale si presenta nel momento in cui i gradienti diventano molto grandi, destabilizzando la fase di aggiornamento dei pesi durante l'addestramento. Questo è stato un elemento fondamentale da introdurre, in quanto, in una struttura condivisa come quella creata, i gradienti delle due diverse loss function si sommavano tra loro, quindi, erano fortemente soggetti a raggiungere valori molto alti.

Un'altra particolarità implementata, di cui però si era già discusso in precedenza, riguarda l'utilizzo di due training loop diversi, in quanto il classificatore, in fase di addestramento, utilizza tutto il training set formato da 5.000 configurazioni, mentre il regressore utilizza, all'interno del training set, solo le configurazioni con Status = OK.

Nella seguente figura è rappresentata, in maniera schematica, la struttura realizzata per la rete neurale multi task a backbone condiviso, descritta fino a questo momento, realizzata per il primo tentativo:

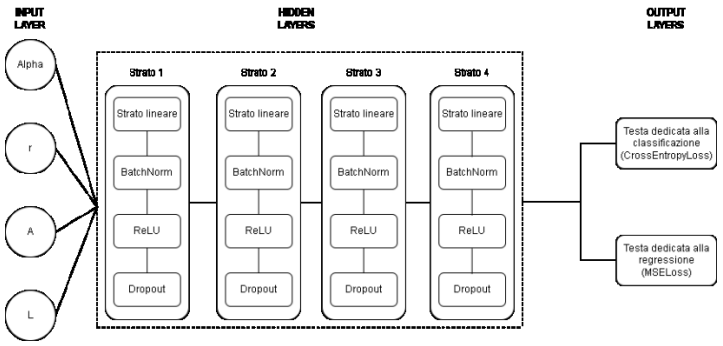


Figura 39: Rappresentazione grafica rete neurale multi task con backbone condiviso

Tutti i dati che sono serviti alla rete per l'addestramento sono stati importati direttamente dai file .csv, definiti in fase di preprocessing, così che non fosse necessario eseguire nuovamente le operazioni, ma i dati e i valori dei parametri fossero già fruibili.

È stato, poi, necessario definire i tensori, in quanto PyTorch non utilizza array Numpy, che, concettualmente sono simili, ma i primi permettono di lavorare sulla GPU, caratteristica di fondamentale importanza se si ha a che fare con dataset di grandi dimensioni e, soprattutto, permette di tenere traccia di tutte le operazioni svolte per il calcolo dei gradienti, così che non debbano essere ricalcolate costantemente. Quindi, gli input e gli output di classificatore e regressore, oltre, anche, ai class weights, sono stati convertiti. Inoltre, come ulteriore operazione, è stata utilizzata la libreria DataLoader, la quale ha permesso di dividere i dati in mini batch, sia per il classificatore che per il regressore, da sottoporre alla rete.

Questa tipologia di rete neurale è risultata concettualmente molto valida e semplice da implementare, ma ha prodotto risultati di performance molto scarsi, riportati nelle tabelle e figure di seguito:

	Precision	Recall	F1-score	Configurazioni
Classe OK	1,00	0,30	0,47	769
Classe No BG	0,55	0,41	0,47	106
Classe Invalid	0,79	1,00	0,88	2125

Figura 40: Risultati performance classificatore primo tentativo

Analizzando i risultati che sono stati ottenuti, si è notato che:

- Classe OK: è la classe che ha presentato i risultati peggiori, nonostante sia quella più importante da predire correttamente. La precision è del 100%, quindi, tutte le volte che la rete pensa e classifica una configurazione come OK, ha effettivamente ragione. Leggendo questo dato si sarebbe potuto pensare che fosse ottimo, ma affiancato alle altre metriche ha messo in luce, in realtà, grandi problematiche, prima fra tutti la recall, che è risultata pari a 0,30, quindi, la rete classificava correttamente solo il 30% delle configurazioni, ciò significa che se si prendessero 769 configurazioni, che sono effettivamente OK, solo 233 verrebbero classificate correttamente, le altre 536 verrebbero classificate, in maniera erronea, in altre classi. Portando ad avere come risultato che, indicativamente, per 2 configurazioni, che hanno Status = OK, su 3, la rete predirebbe altre classi. Il valore di precision sembra, quindi, paradossale rispetto a quanto detto per la recall, ma ciò, in realtà, significa che: la rete tendeva ad avere un comportamento conservativo, quindi, su 5.000 configurazioni del training set la rete tendeva a classificarne come OK solo 231, per le quali è sicura al 100% che fossero della classe OK. Essendo, poi, il valore di F1-score la media armonica di recall e precision, esso è calato drasticamente a causa del valore basso di recall. L'F1-score ha sottolineato, quindi, che il classificatore falliva il 53% delle volte su questa classe.
- Classe No BG: a differenza della classe precedente, in questo caso la rete presentava una recall pari al 0,41, significando che riusciva a classificare correttamente solo il 41% delle configurazioni. Tenendo conto che, all'interno del test set, sono presenti solo 106 campioni che rispettano i vincoli fisici, ma non hanno una banda di stop, è un risultato del tutto accettabile in quanto sono davvero pochi per riuscire ad addestrare bene una rete. Il problema grosso era il valore della precision, in quanto esso era pari a 0,55, quindi il 45% delle configurazioni che la rete classificava come No BG, in realtà non lo erano e avrebbero dovuto essere classificate in un'altra maniera. Nuovamente, calcolando l'F1-score, come media armonica delle altre due metriche, si otteneva un valore nella media. Una delle problematiche maggiori riscontrate per questa classe è, sicuramente, la scarsità di configurazioni, rendendo difficoltosa l'affidabilità statistica dei risultati ottenuti.
- Classe Invalid: questa è la classe per la quale si sono ottenuti i risultati migliori con una recall pari a 1,00, la quale indicava che tutte le configurazioni invalide erano classificate correttamente, e solo il 21% delle configurazioni classificate come invalide in realtà appartenevano ad altre classi, valore ricavato dal dato di

precision. Anche il valore di F1-score, di conseguenza, tendeva ad essere molto alto.

È stato calcolato anche l'F1-score macro, il quale indica un valore consuntivo come media dei F1-score di ogni classe del problema. Questo valore è risultato pari a 0,6051, denotando una performance mediocre con netto margine di miglioramento. Oltre a un'accuratezza, di tutto il modello di classificazione, pari a 0,8003.

Per la valutazione delle prestazioni del classificatore è stata realizzata anche la matrice di confusione, riportata di seguito.

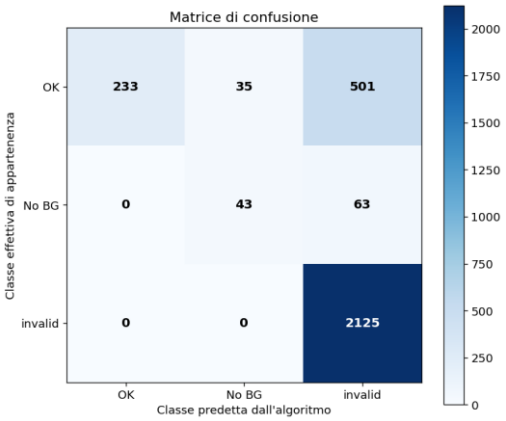


Figura 41: Matrice di confusione per la valutazione della prestazione del classificatore

La confusion matrix è un ottimo strumento, che ha permesso di valutare, in maniera rapida e intuitiva, la performance dell'algoritmo creato, ribadendo i risultati riportati nella tabella precedente. Come si nota dai dati presenti nella matrice, il modello lavorava molto bene sulla classe Invalid, per la quale classificava correttamente tutte e 2125 le configurazioni, mentre, per le altre due classi aveva prestazioni del tutto inaccettabili, soprattutto per la classe OK, che è la più importante, infatti, più del doppio delle volte classificava in maniera sbagliata la configurazione.

Valutando integralmente tutti i risultati ottenuti, le conclusioni che sono state tratte, per la testa di rete neurale dedicata alla classificazione, è che l'algoritmo aveva imparato a classificare come Invalid praticamente tutte le configurazioni. Soprattutto quando non era certa della propria scelta, tendeva ad assegnarle a questa classe anche se non era quella corretta, caratteristica che si nota molto bene anche dalla matrice di confusione. Stessa cosa valeva per la classe OK, ma all'opposto, ossia l'algoritmo tendeva a classificare le configurazioni come OK, solo se era certo che appartenessero effettivamente a questa classe. Questo problema era colpa della forte prevalenza di configurazioni non valide, nonostante l'introduzione dei class weights per cercare di bilanciare questa differenza.

Valutata la prestazione della testa dedicata alla classificazione, la concentrazione è stata spostata a valutare, anche, la performance del regressore, per il quale si sono ottenuti i seguenti risultati:

	MAE	MAPE	R^2
BG_start	73,80 Hz	46,28%	-0,4770
BG_end	214,00 Hz	63,28%	-2,2129

Figura 42: Risultati performance regressore primo tentativo

Come è possibile notare dai risultati riportati nella tabella sopra:

- Il valore di R^2 è risultato negativo per entrambi i valori di output generati dalla rete neurale. Questo era, senza dubbio, il problema maggiore del modello realizzatoo, in quanto evidenziava che la rete aveva una performance inferiore rispetto a quella che si sarebbe ottenuta decidendo di predire sempre il valor medio. Considerando che la predizione perfetta corrisponde a $R^2 = 1$, è facile capire come la rete stesse funzionando molto male e non avesse imparato nulla in fase di addestramento.
- Il MAE indica il valore dell'errore medio in termini assoluti di Hz. Anche per questa metrica si sono ottenuti dei risultati disastrosi, in quanto:
  - Per BG_start si è ottenuto un MAE pari a 73,80 Hz, su un range di valori che varia tra 16 e 600 Hz corrisponde, circa, all'12%.
  - Per BG_end si è ottenuto un MAE pari a 214 Hz, su un range di valori che varia tra 65 e 990 Hz corrisponde circa all'21%.
- Il MAPE, invece, indica l'errore percentuale medio. Per entrambi i valori di inizio e di fine della banda di stop si ottengono risultati che sono davvero pessimi, con livelli di errore molto alti e non accettabili, praticamente pari e superiori al 50%.

Per mostrare meglio la scarsa prestazione del regressore sono stati realizzati due grafici scatterplotter, uno per ogni valore di output, per mostrare graficamente gli errori, che venivano commessi dalla rete, durante la fase di predizione:

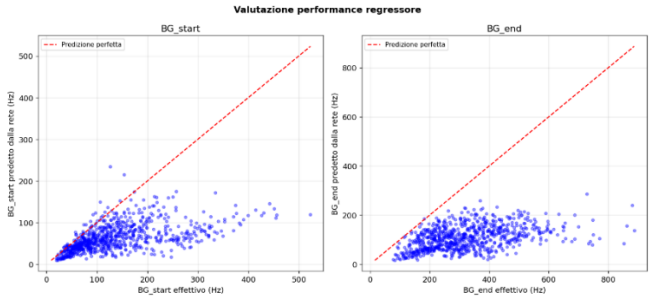


Figura 43: Grafici per la valutazione della performance del regressore per il primo tentativo

I grafici riportati nella figura sopra, hanno l'obiettivo di mettere in luce il significato della metrica R^2. La riga rossa tratteggiata denota la predizione perfetta che la testa della rete neurale dedicata alla regressione potrebbe fare, quindi il valore $R^2 = 1$, i puntini, invece, mostrano le predizioni fatte dal modello. La situazione migliore possibile sarebbe quella in cui la rete riuscisse a predire esattamente il valore corretto e, quindi, i puntini dovrebbero seguire, il più possibile, l'andamento del segmento, ma, come è ben visibile, il modello era ben lontano dall'andamento perfetto, sottolineando in maniera più chiara e intuitiva la scarsa prestazione.

Infine, per terminare e completare la valutazione di questo modello, sono state realizzate, anche, le curve di apprendimento, le quali sono riportate nell'immagine seguente:

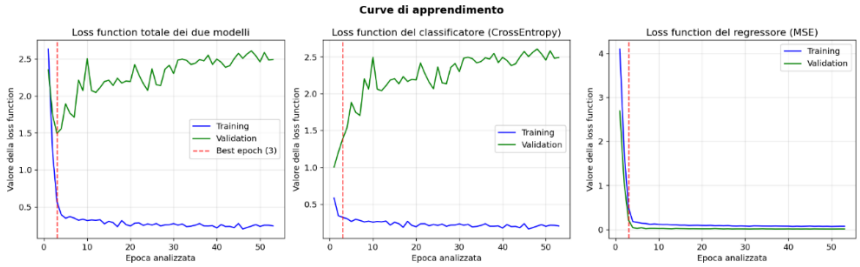


Figura 44: Curve di apprendimento primo tentativo

Le curve di apprendimento hanno permesso di avere una conferma visiva, più semplice e intuitiva, del fallimento dell'architettura realizzata come primo tentativo. In figura sono riportate le curve dedicate al classificatore e al regressore, oltre a un grafico consuntivo finale che mette insieme sia la loss function del classificatore che quella del regressore. Come descritto anche nel capitolo teorico, dedicato al ML, l'andamento di queste curve dovrebbe essere, sia per quella di training, che per quella di validation, decrescente, altrimenti si deduce sia presente una situazione di overfitting da dover risolvere. Come si può vedere, per il classificatore, o più in generale sul grafico dedicato alla funzione totale, l'andamento del training era decrescente, ma quello del validation tendeva a salire fin da subito. Andando ad indagare meglio i risultati e i grafici, si è intuito che il problema non consisteva solamente in un forte overfitting del modello, ma quest'ultimo non riusciva proprio a imparare e generalizzare le configurazioni che gli venivano fornite e il segnale più importante che ha portato a questa conclusione è stato il valore dell'epoca migliore, ossia 3, con un'interruzione, per early stop, dopo non appena 53 epoche e 26 secondi. Questo valore mette in luce come, dopo le prime iterazioni, nelle quali viene trovato il risultato ottimo, il modello smetteva di migliorare e imparare. Inoltre, a una prima lettura il grafico della funzione di loss del regressore poteva sembrare avere il corretto andamento, ma, anche in questo caso, il risultato era ingannevole, in quanto esso dipendeva dal fatto che la rete tendesse a predire sempre il valore medio, producendo una loss function bassa, ma con un valore di R^2 estremamente negativo.

È stato, infine, realizzato anche il grafico relativo all'andamento del learning rate:

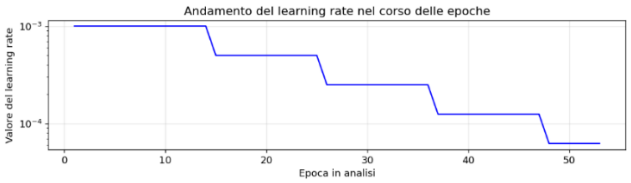


Figura 45: Andamento learning rate primo tentativo

Il learning rate ha subito, nel corso del processo, quattro dimezzamenti, i quali hanno permesso di mettere in luce come, nemmeno la riduzione di tale parametro, potesse risolvere le scarse prestazioni del modello.

Analizzando i risultati ottenuti sia per il classificatore che per il regressore è stato possibile dedurre le motivazioni della pessima performance del modello. La motivazione principale è stata riscontrata nel doppio aggiornamento dei pesi che il modello doveva realizzare: per il primo, ossia quello per la classificazione, i gradienti venivano calcolati su 5.000 configurazioni, mentre per il secondo, ossia quello dedicato alla regressione, solamente su 1315. Il problema principale che è stato riscontrato è che questi due aggiornamenti spingevano in direzioni completamente opposte i pesi, a causa dei compiti nettamente diversi che le due teste dovevano svolgere, di conseguenza, la rete non era in grado di soddisfare entrambe le richieste. In particolare, tra i due compiti, quello che tendeva a dominare l'altro, era la classificazione, in quanto aveva maggiori configurazioni su cui calcolare i gradienti, impendendo, così, al regressore di convergere. La testa dedicata alla classificazione, quindi, tentava di aggiornare i pesi in maniera tale da effettuare la divisione tra le tre diverse classi, mentre il regressore tentava di aggiornare i gradienti in modo tale da effettuare delle predizioni, ma veniva sempre sormontato dal classificatore. Nonostante, comunque, dominasse il classificatore, esso non riusciva ad ottenere buone prestazioni, in quanto subiva un'influenza da parte del regressore, facendo sì che l'intera rete neurale, a backbone condiviso, non riuscisse mai a raggiungere una situazione di stabilità. Quindi, riassumendo, la causa principale del fallimento del modello realizzato era l'interferenza che i due compiti creavano l'uno sull'altro.

3.5.2 Secondo tentativo: riduzione dell'architettura a backbone condiviso

Nonostante quanto affermato alla fine della sotto-sezione precedente si è effettuato un secondo tentativo, mantenendo la stessa struttura, ma applicando alcune modifiche per verificare che effettivamente non si trattasse di overfitting. La principale modifica che è stata applicata è la seguente:

- Aumento del dropout da 0,2 a 0,4.

Questo è una delle prime modifiche che deve essere applicata nel caso in cui si sospetti overfitting, quindi un aumento del numero di neuroni che a ogni epoca vengono spenti.

Nonostante ciò, come era possibile intuire, i risultati non hanno mostrato nessun miglioramento significativo: l'epoca migliore è risultata essere la quarta, individuata in

24 secondi, e dopo, solo, 54 epoche il processo di training si è interrotto, denotando, anche in questo caso, una prestazione molto scarsa della rete che non è riuscita ad apprendere in modo efficacie i dati.

Per quanto riguarda il classificatore i risultati ottenuti sono i seguenti:

	Precision	**Recall**	**F1-score**	**Configurazioni**
Classe OK	1,00	0,36	0,53	769
Classe No BG	0,46	0,50	0,48	106
Classe Invalid	0,82	1,00	0,90	2125

Figura 46: Risultati performance classificatore primo tentativo con aumento del dropout

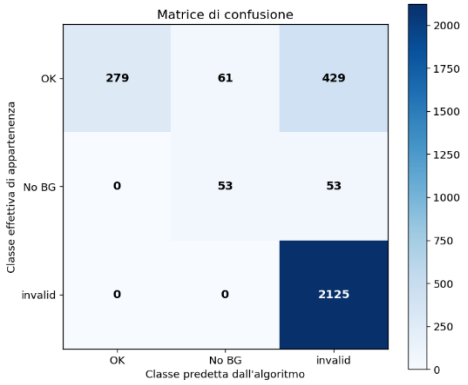


Figura 47: Matrice di confusione per la valutazione della prestazione del classificatore con dropout aumentato

Mentre per il regressore i risultati sono i seguenti:

	MAE	**MAPE**	**R^2**
BG_start	60,84 Hz	39,68 %	-0,1069
BG_end	141,00 Hz	41,79 %	-0,7476

Figura 48: Risultati performance regressore primo tentativo con dropout aumentato

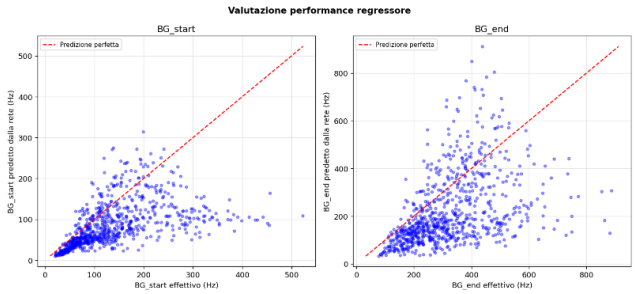


Figura 49: Grafici per la valutazione della performance del regressore per il primo tentativo con dropout aumentato

Come si nota dai due grafici precedenti, anche con un aumento del dropout, le predizioni fatte continuavano ad essere molto distanti dal valore effettivo e, addirittura, la rete commetteva errori che erano più grandi, soprattutto sul valore di fine della frequenza.

Sono poi stati realizzati anche i grafici relativi alle curve di apprendimento, che come era intuibile, non sono migliorati:



Figura 50: Curve di apprendimento primo tentativo con dropout aumentato

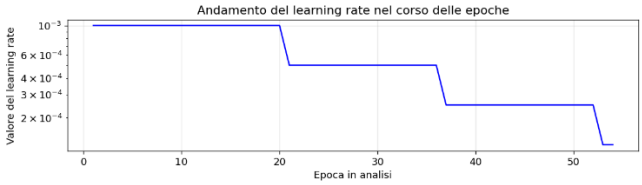


Figura 51: Andamento learning rate primo tentativo con dropout aumentato

Questa modifica effettuata ha permesso di dimostrare che, effettivamente, il problema non era l'overfitting, in quanto andando ad aumentare il dropout si sarebbe dovuto ottenere un miglioramento nelle prestazioni, anche se piccolo. Nel modello era, quindi, presente una complessità molto più profonda a livello strutturale causata da un'interferenza tra i due task che non riuscivano a coesistere nello stesso backbone, in quanto si intralciavano l'uno con l'altro.

Altre modifiche che sono state applicate, ma che non hanno portato a un miglioramento dei risultati, sono:

- Variazione del valore di lambda nel calcolo della funzione di loss totale. Esso è stato aumentato, per provare a dare maggiore importanza al regressore. In particolare, il classificatore lavora con 4 volte il numero di campioni del regressore, quindi, è stato assegnato un valore pari a quattro al coefficiente lambda, per provare a far dominare il task dedicato alla regressione. Tale valore, inoltre, è anche stato posto uguale a zero, per vedere se almeno il classificatore lavorasse correttamente.
- Variazione del numero di neuroni all'interno di ogni strato della rete. Essi potevano essere aumentati, per far in modo che la rete imparasse più configurazioni possibili e non fermasse il proprio apprendimento dopo sole tre epoche. Inoltre, è stato eseguito anche il tentativo opposto di diminuire il numero di neuroni all'interno di ogni strato, per tentare di eliminare i conflitti interni tra i due task.
- Variazione del numero di hidden layer della rete neurale. In particolare l'aumento del numero di strati nascosti è stato implementato per provare se la rete, dotata di una maggiore profodnità, fosse in grado di apprendere meglio le informazioni. Parallelamente, è stato effettuato anche il tentativo opposto, riducendo il numero di strati nascosti con l'obiettivo di limitare, per quanto possibile, il conflitto esistente.
- È stato abbassato il valore del learning rate, per avere aggiornamenti più piccoli, così da garantire maggiore equilibrio e ridurre l'interferenza tra i due compiti.
- È stato aumentato il valore di patience dello scheduler e dell'eraly stopping. Si è ipotizzato, infatti, che concedendo un intervallo di tempo maggiore alla rete, per apprendere i dati, essa sarebbe stata in grado di realizzare e imparare rappresentazioni migliori.
- È stata aumentata la dimensione dei batch che venivano forniti alla rete, così da ottenere gradienti più stabili, con una riduzione dell'interferenza tra i due compiti.

Nonostante siano state fatte tutte queste modifiche, la rete con architettura a backbone condiviso ha continuato ad avere prestazioni scarse, ottenendo miglioramenti che, però, erano davvero marginali, mantenendo, pur sempre, livelli di accuratezza, precisione ed errore non accettabili, in quanto il problema non era tanto nel valore assegnato ai parametri, quanto più nella struttura vera e propria.

3.5.3 Soluzione definitiva: definizione di due reti completamente indipendenti

Capite le motivazioni alla base delle scarse prestazioni del modello precedente, si è andata a rivalutare completamente l'architettura della rete neurale, infatti, nonostante concettualmente poter sfruttare un backbone condiviso poteva essere un'ottima idea, soprattutto per quanto riguarda il risparmio dei costi computazionali, in realtà si è dimostrato che i due task interferivano tra loro, portando a scarse prestazioni. Motivo per il quale come ulteriore, ed ultimo, tentativo sono state implementate due reti neurali

completamente indipendenti tra loro, così da eliminare qualsiasi forma di connessione o interferenza tra i due task.

Come per i tentativi precedenti, anche in questo caso, è stato realizzato un notebook di Google Colab, nel quale inizialmente sono state importate le stesse librerie viste per i tentativi precedenti, oltre ad effettuare il caricamento di tutti i dati lavorati durante la fase di preprocessing, i quali erano stati salvati in appositi file per essere riutilizzati nel corso del progetto. Infine, l'ultima attività svolta, prima della definizione delle due nuove reti neurali, è stata la conversione dei dati e dei class weights, sia per il regressore che per il classificatore, in tensori utilizzabili da PyTorch e, attraverso la libreria DataLoader, la divisione dei dati in mini-batch, la cui dimensione è stata fissata a 64, che rappresenta un ottimo trade-off tra: velocità computazionale e quantità di dati per l'addestramento delle reti.

Dopo di che, sono state definite le due reti neurali completamente indipendenti, ognuna delle quali presentava il proprio backbone, il proprio ottimizzatore e il proprio training loop:

- ClassifierNet: rete neurale dedicata alla classificazione, quindi alla divisione delle configurazioni nelle tre classi. Essa veniva addestrata sulle 5.000 configurazioni, che componevano il training set del classificatore, creato durante la fase di preprocessing dei dati.
- RegressorNet: rete dedicata alla predizione dei valori di frequenza di inizio e di fine della banda di stop. Essa, a differenza della precedente, è stata addestrata su 1315 configurazioni, ossia tutte quelle combinazioni di parametri di input, presenti all'interno del training set per il regressore, che hanno permesso di definire delle configurazioni per le quali esiste una banda di inibizione, quindi hanno Status = OK.

L'addestramento di queste due reti è avvenuto in sequenza, quindi, prima veniva addestrata la rete dedicata alla classificazione e, solo dopo la fine di questa prima fase di training, partiva da zero l'addestramento del regressore. Questa è una caratteristica completamente diversa rispetto ai due casi precedenti, nei quali l'addestramento per le due teste, avveniva in contemporanea, motivo per il quale l'aggiornamento dei pesi di un task influivano sull'aggiornamento dei valori dell'altro task. In questo modo, dividendo completamente i training delle reti neurali, si eliminava completamente il problema delle interferenze, in quanto ogni rete aggiornava in maniera autonoma i propri gradienti.

In particolare, entrambe le reti neurali nuove, utilizzavano la struttura a diamante, vista anche per i primi tentativi effettuati. È stato deciso di mantenere la stessa struttura in quanto, come più volte detto, il problema dei casi precedenti non erano tanto i parametri o valori scelti, ma quanto più la vera e propria struttura condivisa. Quindi è stato deciso di mantenere questa architettura, in modo tale da verificare, anche, che la separazione dei tasks nel backbone, con il mantenimento della stessa struttura, portasse effettivamente a un miglioramento e, quindi, non ci fossero altri problemi di fondo, ma che l'interferenza fosse il principale. Anche in questo caso, quindi, è stata mantenuta una struttura con 4 strati nascosti [64; 128; 128; 64], ognuno dei quali con: Linear, BatchNorm1d, ReLU e Dropout(), essi presentavano le stesse caratteristiche già viste per i tentativi precedenti e

ognuna delle operazioni è stata scelta per le stesse ragioni già citate e affrontate nelle sotto-sezioni precedenti.

L'unica modifica che è stata applicata è nel valore assegnato al dropout, infatti, sono stati scelti due valori diversi per il regressore e per il classificatore:

- ClassifierNet a cui è stato assegnato un valore di dropout pari a 0,3. Questo è un valore leggermente più alto rispetto a quello che è stato assegnato a RegressorNet, in quanto, la rete dedicata alla classificazione, opera su un numero di configurazioni maggiore, quindi, un valore di dropout maggiore ha contribuito a evitare il problema dell'overfitting, garantendo, parallelamente, un apprendimento più efficacie.
- RegressorNet, come specificato anche nel caso precedente, gli è stato assegnato un valore minore, in quanto ha utilizzato, per la fase di addestramento, un numero inferiore di configurazioni. In particolare si è scelto 0,2, valore scelto anche per evitare di rendere troppo instabile la fase di training, data la scarsa dimensione del training set utilizzato per questo task, infatti se si fossero spenti più neuroni, durante ogni epoca, si sarebbe rischiato che la rete non imparasse correttamente a svolgere il proprio compito.

Da notare che i primi due tentativi effettuati utilizzavano un valore ancora maggiore di dropout, pari a 0,4, per gestire il problema dell'overfitting causato dalla condivisione della struttura.

Un'altra modifica applicata, che è una diretta conseguenza della nuova struttura, riguarda la separazione del calcolo della loss function. Infatti, per il classificatore è stato scelto, nuovamente, il CrossEntropyLoss, mentre per il regressore il MSELoss, come visto anche nei casi precedenti. Ciò che è cambiato, però, è il calcolo dei valori, essi, infatti, non vengono calcolati contemporaneamente, ma ogni task eseguiva l'aggiornamento in maniera totalmente indipendente. Questa scelta rappresenta un ulteriore elemento a supporto dell'assenza di interferenza tra task presente in questo terzo tentativo effettuato. Di conseguenza, non è stato necessario introdurre la variabile lambda per il bilanciamento delle loss function, poiché le due reti lavorano in maniera indipendente, quindi ognuna punta alla minimizzazione della propria funzione di perdita.

Seguendo lo stesso ragionamento sono stati inseriti due ottimizzatori Adam completamente indipendenti, mentre nei casi precedenti veniva condiviso creando i problemi più volte citati.

Sono stati, inoltre, inseriti due scheduler separati, i quali permettono di dimezzare il learning rate quando la loss function del validation non migliora ed è stato aumentato anche il valore di patience dello scheduler, posto uguale a 15, in quanto avendo due reti, il rischio che si poteva correre era quello di finire in un plateau, prima di effettuare un miglioramento. Aumentando il valore della patience si è dato più tempo al modello per uscire da un eventuale situazione di stallo, evitando di ridurre troppo presto il learning rate, processo che avrebbe potuto portare a una convergenza più lenta. Il valore della patience per l'early stop, invece, è stato mantenuto identico.

Riassumendo la struttura realizzata per questo tentativo è riportata nella seguente figura:

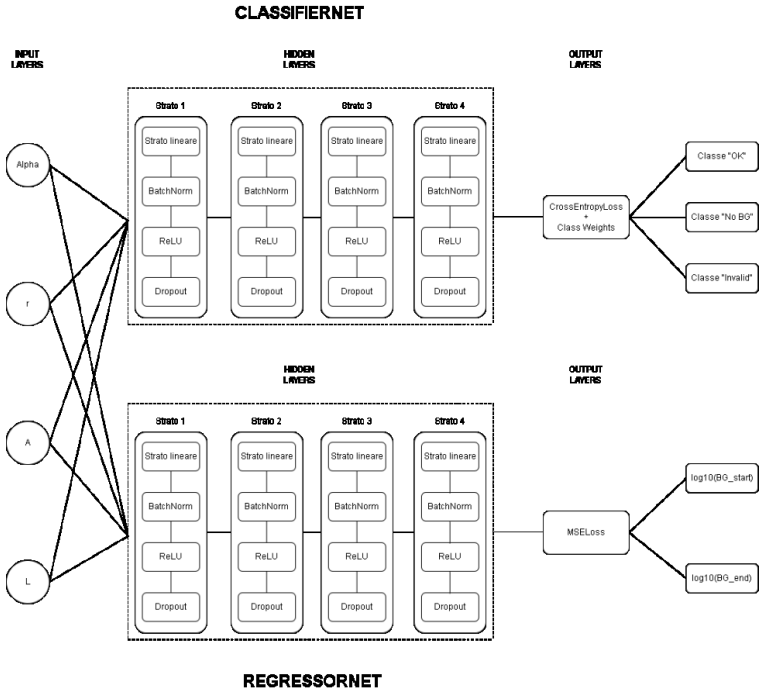


Figura 52: Rappresentazione grafica delle due reti neurali separate

Come si può osservare dalla figura, e come è stato descritto più volte nelle sotto-sezioni precedenti, per questo tentativo sono state realizzate due reti neurali feed-forward completamente indipendenti. Esse condividono solo i valori di input, mentre presentano strutture e parametri completamente differenti. Questa struttura ha permesso di eliminare completamente il problema dell'interferenza, infatti, dividendo il compito di classificazione da quello di regressione gli aggiornamenti dei gradienti non si ostacolavano, o influenzavano, gli uni con gli altri.

A differenza del caso precedente, come ci si poteva aspettare, si sono ottenuti risultati decisamente migliori, che sono riportati di seguito.

Per ClassifierNet le metriche per la valutazione della prestazione presentavano i valori riportati nella tabella seguente:

	Precision	**Recall**	**F1-score**	**Configurazioni**
Classe OK	0,95	1,00	0,97	769
Classe No BG	0,93	1,00	0,96	106
Classe Invalid	1,00	0,98	0,99	2125

Figura 53: Risultati performance ClassifierNet

La rete dedicata alla classificazione ha, inoltre, raggiunto un'accuratezza del 98,2% e un valore di F1-score macro pari a 97,42%, valori che denotano un'ottima performance svolta dalla rete dedicata alla classificazione.

Anche in questo caso è stata realizzata una matrice di confusione, che ha permesso di rappresentare, graficamente, i dati usati per il calcolo delle metriche di performance presenti nella tabella precedente.

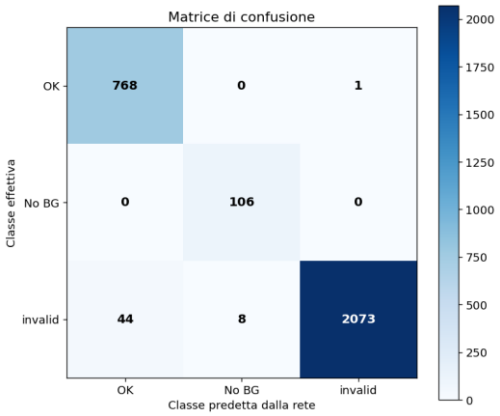


Figura 54: Matrice di confusione ClassifierNet

Come si può osservare dalla matrice di confusione, la rete dedicata alla classificazione è stata in grado di riconoscere perfettamente tutte le configurazioni appartenenti alle due classi minoritarie, ossia OK e No BG. Questo è un risultato molto importante, in quanto la rete con struttura a backbone condiviso non era stata in grado di raggiungere lo stesso livello di accuratezza. In particolare, ai fini del progetto, era fondamentale identificare e classificare correttamente tutte le configurazioni che presentano status = OK. La rete ha, poi, mantenuto un comportamento buono anche sui casi Invalid, classificando erroneamente solo 52 configurazioni su 2125, questo ha dimostrato che la rete tendeva ad avere un comportamento più conservativo, per il quale, nel caso fosse indecisa, preferiva classificare la configurazione come OK, piuttosto che rischiare e classificarla come invalida, in quanto riteneva che fosse più grave perdere delle configurazioni con status = OK, piuttosto che analizzare qualche caso Invalid in più.

Il ClassifierNet ha trovato il miglior valore della loss function, pari a 0,0468, all'epoca 227, con un successivo early stop all'epoca 277 in 88 secondi. Come si può osservare dai valori assunti dalla best epoch, in questa architettura di rete neurale, il modello ha avuto tutto il tempo necessario per poter essere addestrato ed imparare correttamente le caratteristiche dei dati, a differenza del caso precedente nel quale la migliore epoca registrata era la seconda, o terza.

Mentre per RegressorNet, i risultati trovati per le metriche sono i seguenti:

	MAE	MAPE	R^2
BG_start	8,97 Hz	6,02 %	0,9663
BG_end	20,54 Hz	6,06 %	0,9511

Figura 55: Valori metriche performance RegressorNet

Come si nota dalla tabella riportata sopra, l'errore medio assoluto commesso, per la frequenza di fine della banda di stop, è maggiore rispetto a quello della frequenza di inizio, questo è dovuto al fatto che BG_end lavora su un range più ampio, in fase di analisi esplorativa del dataset inziale, infatti, si era notato che i valori di questo dato di output erano compresi nell'intervallo [66; 992]. La rassicurazione del corretto funzionamento e delle buone prestazioni di RegressorNet è derivato, però, dal valore dell'errore percentuale, il quale si fissa su 6% per entrambi i valori di output, evidenziando un risultato molto buono per un problema non lineare come quello affrontato. Ottimi valori, poi, si sono ottenuti anche per il coefficiente R^2.

Come per i tentativi precedenti, anche in questo caso, è stato realizzato lo scatterplotter che mette in relazione la predizione effettuata dalla rete e il valore reale di inizio e di fine frequenza della banda di stop.

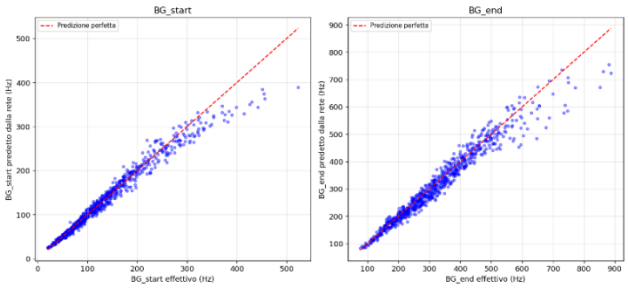


Figura 56: Grafico valutazione performance RegressorNet

Come si nota dai grafici a scatterplot riportarti in figura 46, la struttura realizzata ha permesso di ottenere risultati, evidenziati dai pallini blu, che seguono meglio la predizione perfetta, ossia la linea rossa tratteggiata. Si può notare un discostamento maggiore dalla retta per valori di frequenze più grandi, dovuto alla minor presenza di configurazioni con tali dati, motivo per il quale la rete non è riuscita ad apprendere a pieno tutte le caratteristiche per ottenere una precisione maggiore, si nota, infatti, che la maggior parte dei pallini blu sono concentrati in zone con frequenze con valori più piccoli.

Il regressore ha individuato il valore migliore di loss function, ossia 0,0013, all'epoca 268 in 27 secondi, con, successivo, early stop all'epoca 318. Come detto anche per ClassifierNet, questi valori di epoche di stop, permettono di mostrare il fatto che la rete abbia avuto sufficiente tempo per imparare le caratteristiche e peculiarità dei dati del set,

a differenza del tentativo a backbone condiviso, nel quale la migliore epoca si era registrata alla seconda, o terza, iterazione.

Inoltre sono state realizzate, per entrambe le reti, le curve di apprendimento, rappresentanti l'andamento delle loss function di training e di validation, oltre alla rappresentazione grafica dell'andamento decrescente di aggiornamento del learning rate.

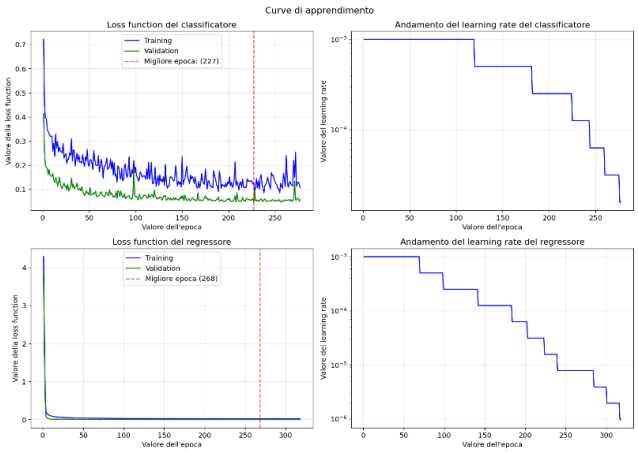


Figura 57: Curve di apprendimento ClassifierNet e RegressorNet

Come si nota dalla figura, riportata sopra, le curve di apprendimento, delle due reti neurali create, hanno, finalmente, l'andamento corretto, ossia decrescente sia per la loss function del training set, che per quella del validation set. In particolare, per RegressorNet, entrambe le curve hanno una discesa rapidissima nelle prime 20 epoche, per poi stabilizzarsi a valori molto vicini allo zero, fino quasi ad arrivare a sovrapporsi, questo è un segnale, molto importante, in quanto ha dimostrato che non fosse presente il fenomeno dell'overfitting. Nello specifico, dalla figura, si nota come la funzione di perdita del validation set, rappresentata dalla linea verde, sia per ClassifierNet che per RegressorNet, risulti essere sempre inferiore rispetto alla funzione di perdita del training set, questo è dovuto al dropout, che permette di disattivare una certa percentuale di neuroni durante la fase di training, ma non durante la fase di validazione. Questo fa sì che, nel secondo caso, la rete utilizzi tutti i neuroni a propria disposizione, avendo prestazioni ancora maggiori. Per quanto riguarda le curve dedicate all'andamento del learning rate, invece, si denota che sono presenti, per ciascuna rete, diversi dimezzamenti del valore di partenza, arrivando a valori decisamente piccoli, che dimostrano che: durante la fase finale dell'addestramento, la rete stava aggiornando i pesi con delle modifiche molto piccole, cercando di affinare sempre di più i risultati.

In ultima battuta, prima di confermare il nuovo modello creato, è stato effettuato un controllo su un vincolo fisico fondamentale, ossia quello che afferma che la frequenza di fine della banda di stop deve sempre essere maggiore rispetto a quella di inizio. Controllo che ha permesso di dimostrare che, tutte le configurazioni classificate come OK dalla rete, lo rispettano.

Un aspetto importante da sottolineare in questa struttura è che effettuando il training loop più volte, si ottengono risultati che possono differire leggermente, ciò è dovuto ad alcuni elementi di variabilità presenti all'interno delle reti, come:

- Inizializzazione casuale dei pesi, i quali vengono assegnati, selezionando dei valori da una distribuzione di probabilità, direttamente da PyTorch. Per tale distribuzione non è stata fissato un seed, di conseguenza, per ogni ciclo di addestramento, si possono ottenere risultati differenti.
- Mescolamento casuale dei batch di dati che sono stati generati. Questo è un punto fondamentale, in quanto i gradienti dei pesi vengono aggiornati dopo ogni batch, di conseguenza, se essi vengono forniti in ordine diverso, allora si avranno aggiornamenti diversi, che portano a risultati differenti.
- Neuroni di dropout casuali, come definito anche nel secondo capitolo, i neuroni che vengono disattivati, in fase di training, dal dropout sono completamente casuali.

Avendo tutti questi elementi di variabilità intrinsechi nel modello, si possono ottenere risultati che sono leggermente differenti tra loro, motivo per il quale è stato effettuato più volte il training del modello per verificare la robustezza del modello. Ciò che è scaturito è che: i risultati che si ottengono differiscono di pochissimo, per esempio: l'accuratezza varia nell'intervallo 97,7-98%, mentre il coefficiente R^2 tra 0,955 e 0,9680, lo stesso ragionamento vale anche per tutte le altre misure di performance, quindi nonostante la variabilità la rete continua a mantenere livelli di performance molto alti.

Questa variabilità può essere eliminata fissando un seed, ma siccome è stato dimostrato che i risultati che si ottengono hanno tutti la stessa qualità, precisione e accuratezza, quindi, non dipendono da determinati valori di inizializzazione più favorevoli, si è preferito mantenere la casualità del modello. Nel modello a backbone condiviso, invece, questa variabilità, seppur presente, era nascosta da un problema più grande, ossia l'interferenza tra i tasks, che faceva in modo di ottenere errori così elevati che eliminavano completamente la possibilità di raggiungere risultati differenti a causa della randomicità, per questo motivo il fenomeno non era stato rilevato, né approfondito. È importante, però, ricordare che la variabilità interna dei modelli di machine learning è una caratteristica importante da studiare e analizzare per garantire la robustezza del modello.

3.6 Fase 4: ottimizzazione inversa

Una volta addestrate correttamente le due reti neurali, dedicate alla classificazione e alla regressione, è stato sviluppato un passaggio ulteriore, ossia l'implementazione di un sistema di ottimizzazione inversa. Esso segue il processo opposto rispetto a quello adottato e descritto fino a questo momento. Infatti, se l'obiettivo del modello di machine learning era riuscire a calcolare, a partire da determinati valori di input dei parametri fisici, il valore della frequenza di inizio e di fine della banda di stop, nella fase di ottimizzazione inversa, si è cercato di rispondere alla seguente domanda: "Quali valori devono essere assegnati ai parametri di input, affinché si possa massimizzare l'ampiezza della band gap?". Questa procedura, quindi, è stata implementata per ottenere

un'ottimizzazione geometrica, con l'obiettivo di individuare configurazioni che massimizzano, o soddisfano, determinati requisiti, in termini di band gap, oppure nel rispetto di range di frequenza, direttamente fissati dall'operatore.

In particolare, nel corso del notebook è stata implementata una funzione di ottimizzazione inversa, come descritto fin ora, essa è stata, poi, particolarizzata aggiungendo la possibilità di fissare una frequenza target da rispettare.

Si è, quindi, sfruttata la velocità di calcolo delle reti neurali, ricercare il valore massimo dell'ampiezza della banda di stop, considerando solo le configurazioni che superavano una determinata soglia di affidabilità nella classificazione dei risultati.

3.6.1 Implementazione funzione di ottimizzazione

Anche per la realizzazione della funzione di ottimizzazione inversa, è stato predisposto un notebook di Google Colab, nel quale sono state importate le librerie e moduli già descritti nelle sezioni precedenti. Ai quali si è aggiunta la libreria Itertools, che permette di generare combinazioni di valori. Una volta completata l'importazione delle librerie necessarie, sono stati caricati i file contenenti i dati elaborati nelle fasi precedenti, per il classificatore, per il regressore. È stato inoltre caricato il file contenente i parametri dello scaler, per garantire che la normalizzazione avvenga sempre allo stesso modo, anche per i nuovi dati di input che verranno generati. Salvare i risultati e i valori in file è stato essenziale per non dover, ad ogni notebook, eseguire nuovamente operazioni che erano già state svolte in precedenza, velocizzando e ottimizzando i processi svolti. Inoltre, per caricare i pesi, salvati in precedenza, è stato necessario ridefinire, con gli stessi valori dei parametri, le due reti neurali che sono state costruite nel notebook precedente.

Successivamente è stata definita la funzione per l'ottimizzazione inversa. Questo processo è stato diviso in alcune fasi principali, le quali sono riportate di seguito:

- Definizione dei range e del valore delle divisioni per ogni parametro di input, che dovranno essere forniti direttamente dall'operatore in base alle necessità. Quindi, l'operatore è in grado di inserire i valori di minimo e massimo, oltre a quello delle divisioni, per ogni parametro di input, esattamente come eseguito dal codice Matlab originale.
- Viene, poi, eseguito il prodotto cartesiano, che genera tutte le possibili configurazioni con le diverse combinazioni di valori di parametri definiti al punto precedente. Dal prodotto cartesiano è possibile generare la griglia di combinazione. Anche in questo caso, come visto a inizio del capitolo, tutte le tuple di parametri sono state convertite in array Numpy, così da poter essere utilizzabili.
- Normalizzazione dei parametri di input, utilizzando gli stessi scaler usati durante la fase di preprocessing dei dati, questo fa sì che la rete riceva valori sulla stessa scala su cui è stata addestrata. Se si fossero utilizzati valori diversi si sarebbe rischiato che la rete ricevesse parametri fuori dal range sul quale era stata addestrata, producendo, di conseguenza, risultati e prestazioni basse, per sicurezza, infatti, è stato aggiunto un ulteriore passaggio per eliminare tutti i valori che risultavano fuori dal range [0;1].

- Tutti i dati sono stati convertiti in tensori PyTorch e, poi, il dataset di configurazioni è stato passato, prima al classificatore, il quale ha permesso di selezionare solo le combinazioni di valori di input che hanno Status = OK. Poi in secondo luogo, le configurazioni selezionate, sono state passate al regressore, il quale ha permesso di predire i valori di BG_start e BG_end, inizialmente in valore logaritmico e poi riconvertiti in Hz. Interessante è stato anche, tenere traccia del tempo impiegato.
- Calcolati i valori di BG_start e BG_end, è stato possibile calcolare il valore di BG_width e BG_mean.
- Successivamente sono state filtrate le configurazioni che rispettano tre vincoli fondamentali: status = OK, BG_start < BG_end e una probabilità di confidenza nei risultati della rete superiore, o pari, a 80%, valore che può essere modificato in relazione alle esigenze dell'operatore.
- Le configurazioni che rispettano i vincoli citati al punto precedente, sono state, poi, ordinate in base a valori decrescenti di BG_width, così che sia stato possibile ottenere la configurazione che consenta di massimizzare l'ampiezza della band gap.

3.6.2 Analisi e validazione dei risultati finali ottenuti dal processo di ottimizzazione inversa

Definita la funzione base è stato possibile testarla inserendo come range dei valori dei parametri e delle divisioni:

- Raggio della sezione: [0,001; 0,003].
- Angolo di piegatura: [0; 60].
- A: [0,13; 0,20].
- L: [0,14; 0,30].
- Numero di divisioni: 50.
- Soglia di confidenza pari a 80%.

È importante ricordare, però, che tutti questi valori sono totalmente personalizzabili da parte dell'operatore.

Inserendo questi dati, quindi, si sono ottenute 6.250.000 (50^4) configurazioni che sono state testate, in un tempo totale di 58.711,2 ms, pari a, circa, 9,7852 minuti, questo dato ha permesso di sottolineare, nuovamente, l'estrema rapidità di analisi ed elaborazione delle reti neurali create, il modello, infatti, impiega pochi minuti per elaborare più di 6 milioni di configurazioni diverse, cosa che per il codice Matlab sarebbe molto difficile, se non impossibile. La configurazione ottimale che è stata trovata, con i parametri di input inseriti in precedenza, è la seguente:

- Raggio della sezione: 0,001612
- Angolo di piegatura Alpha: 25,71 gradi
- A: 0,13 m^2
- L: 0,1596 m

Questi parametri hanno permesso di definire la combinazione di valori, all'interno dei range fissati, che permette di ottenere la massimizzazione della dimensione della band gap, più precisamente:

- BG_start: 283,58 Hz.
- BG_end: 661,58 Hz.
- BG_width: 378,00 Hz, ossia = 661,58 – 283,58.
- BG_mean: 472,58 Hz.

Successivamente si è deciso di aggiungere la possibilità di stampare le 15 configurazioni che permettono di massimizzare i valori di band gap, ossia quelle che compaiono per prime nella classifica decrescente creata, ottenendo i seguenti risultati:

r	Alpha	A	L	BG_start_Hz	BG_end_Hz	BG_width_Hz	Prob_OK
0.001612	25.714285	0.130000	0.159592	283.582794	661.580383	377.997589	0.800234
0.001571	25.714285	0.130000	0.159592	279.862183	657.408875	377.546692	0.829574
0.001571	25.714285	0.131429	0.159592	279.293854	656.411743	377.117889	0.809310
0.001571	24.489796	0.130000	0.159592	288.563599	665.319824	376.756226	0.831577
0.001612	24.489796	0.130000	0.159592	291.801270	668.136353	376.335083	0.801072
0.001490	24.489796	0.130000	0.156327	281.891083	658.161804	376.270721	0.823307
0.001571	24.489796	0.131429	0.159592	287.995605	664.244324	376.248718	0.812005
0.001327	18.367348	0.130000	0.149796	315.341370	691.491394	376.150024	0.838781
0.001653	26.938776	0.130000	0.162857	275.630585	651.668274	376.037689	0.804483
0.001612	25.714285	0.130000	0.162857	281.745667	657.589355	375.843689	0.852970
0.001367	18.367348	0.130000	0.149796	318.965454	694.650146	375.684692	0.802546
0.001653	25.714285	0.130000	0.162857	284.986145	660.632080	375.645935	0.809238
0.001531	24.489796	0.130000	0.159592	284.548248	660.126343	375.578094	0.851876
0.001612	25.714285	0.131429	0.162857	281.263947	656.805359	375.541412	0.835312
0.001327	19.591837	0.130000	0.149796	304.770416	680.293335	375.522919	0.826364

Figura 58: Classifica configurazioni che massimizzano la band gap

Per analizzare meglio i risultati ottenuti e capire in maniera rapida se la funzione identificata stesse funzionando correttamente, sono stati realizzati alcuni grafici riportati di seguito:

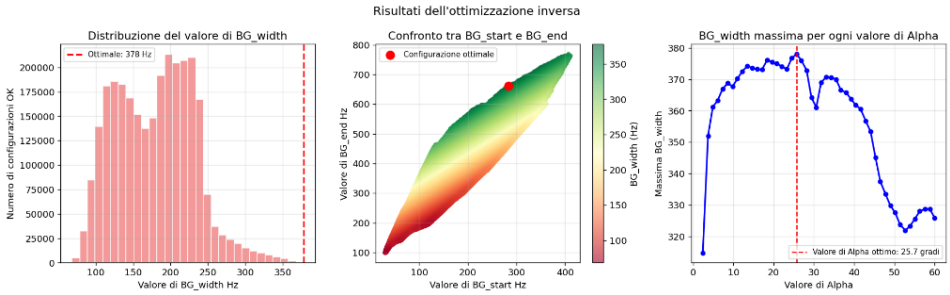


Figura 59: Grafici relativi all'ottimizzazione inversa con n=50

Il primo grafico, partendo da sinistra, mostra la distribuzione dei valori di ampiezza della band gap, in funzione del numero di configurazioni. Con la linea tratteggiata rossa, poi, è stata evidenziato il valore ottimo, ossia quello che massimizza BG_width. Il secondo grafico, invece, permette di mettere in luce il confronto tra BG_start e BG_end. La forma

che si viene a definire è composta da puntini, che rappresentano la differenza tra BG_end e BG_start di una configurazione data, più è piccolo il valore della differenza più il colore tenderà al rosso, viceversa per configurazioni che hanno differenze maggiori, quindi, di conseguenza, per valori di ampiezza delle band gap maggiori, il colore tende al verde. Con il pallino rosso, invece, è individuata la configurazione che permette di massimizzare l'ampiezza della banda di inibizione. Nell'ultimo grafico a destra, invece, è stato rappresentato l'andamento del valore di BG_width in funzione dell'ampiezza dell'angolo di piegatura. Si era, già, notato come per angoli compresi tra i 10 e i 25 gradi, si ottenessero i valori massimi di ampiezza per le band gap. Il grafico permette di evidenziare questa tendenza; come si vede, infatti, i valori maggiori di ampiezza della band gap si concentrano nell'intervallo definito, per poi diminuire per valori, di angoli di piegatura, superiori ai 30 gradi. La linea tratteggiata rossa evidenzia il valore dell'angolo di piegatura in corrispondenza del quale si ottiene il massimo valore di ampiezza per la band gap.

A titolo informativo sono riportati i risultati che si sono ottenuti, mantenendo gli stessi range di parametri, ma fissando il numero delle divisioni pari a 10 per ognuno dei parametri di input:

- Raggio della sezione: 0,001444
- Angolo di piegatura Alpha: 20,00 gradi
- A: 0,13 m^2
- L: 0,1578 m

Con valori di output pari a:

- BG_start: 307,50 Hz.
- BG_end: 675,82 Hz.
- BG_width: 368,33 Hz, ossia = 675,82 – 307,50.
- BG_mean: 491,66 Hz.

Analisi effettuata su 10.000 configurazioni, in 147,3 millisecondi.

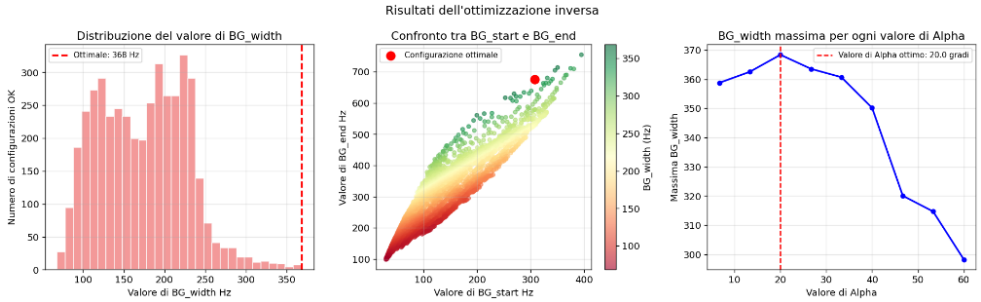


Figura 60: Grafici relativi all'ottimizzazione inversa con n=10

Con le migliori configurazioni identificate:

r	Alpha	A	L	BG_start_Hz	BG_end_Hz	BG_width_Hz	Prob_OK
0.001444	20.000000	0.130000	0.157778	307.495483	675.823853	368.328369	0.891729
0.001444	26.666666	0.130000	0.157778	252.908340	616.410156	363.501831	0.881293
0.001444	13.333333	0.130000	0.157778	350.691467	713.178162	362.486694	0.920168
0.001444	33.333332	0.130000	0.157778	199.779282	560.442566	360.663269	0.886564
0.001667	26.666666	0.130000	0.175556	269.310944	629.220154	359.909210	0.919665
0.001444	6.666667	0.130000	0.157778	395.206787	753.958130	358.751343	0.862449
0.001444	13.333333	0.137778	0.157778	345.799835	704.295593	358.495758	0.832567
0.001667	26.666666	0.137778	0.175556	262.713257	615.369141	352.655884	0.883494
0.001667	33.333332	0.130000	0.175556	213.470718	565.648376	352.177673	0.930707
0.001889	40.000000	0.130000	0.193333	193.703735	543.879211	350.175476	0.837110
0.001222	13.333333	0.130000	0.157778	331.401337	679.932617	348.531281	0.967976
0.001222	20.000000	0.130000	0.157778	273.139160	620.956970	347.817810	0.947797
0.001444	20.000000	0.130000	0.175556	287.784607	633.443970	345.659363	0.976736
0.001667	33.333332	0.137778	0.175556	205.924973	550.130981	344.205994	0.880593
0.001667	40.000000	0.130000	0.175556	173.367203	515.655823	342.288635	0.940032

Figura 61: Configurazioni che massimizzano la dimensione della band gap con n=10

3.6.3 Fase di validazione tramite codice Matlab

Come già noto, il codice Matlab è estremamente preciso nei risultati che fornisce, motivo per il quale è stato deciso di utilizzare il modello originale per verificare la correttezza dei risultati ottenuti tramite la funzione di ottimizzazione inversa, per controllare che, effettivamente, stesse lavorando correttamente e non commettesse errori. Per fare ciò sono state applicate piccole modifiche al codice Mtalab, così che potesse prendere in input alcuni valori di configurazioni e fosse in grado di eseguire la simulazione completa, oltre al calcolo della percentuale di differenza tra il valore simulato, tramite Matlab, e il valore predetto dalla funzione di ottimizzazione inversa.

L'attività di validazione è stata divisa in fasi successive durante le quali sono state testate diverse configurazioni, con parametri differenti. Per facilitare il compito è stata introdotta una cella, sul notebook di Google Colab, che stampasse direttamente i valori dei parametri delle configurazioni nella forma adatta da poter inserire direttamente all'interno del codice Matlab.

Per la fase di validazione sono stati testati dei set composti da un numero di configurazioni, determinato dal trade off tra velocità di esecuzione dei calcoli, da parte del codice Matlab, e la sicurezza statistica dei risultati. Come già evidenziato, il modello Matlab presenta tempi di calcolo elevati; pertanto, scegliere un numero maggiore di configurazioni avrebbe rallentato i tempi per eseguire l'analisi.

La fase di validazione è stato un processo iterativo, composto da set di configurazioni differenti, per testare varie aree differenti:

- Il primo set era composto da valori di input nei range già fissati in precedenza, ma con il valore di divisioni pari a 10. In particolare, sono state selezionate le prime 15 configurazioni che massimizzano l'ampiezza della band gap.

Le configurazioni che sono state testate e validate direttamente sul codice Matlab sono le seguenti:

r	Alpha	A	L	BG_start_Hz	BG_end_Hz	BG_width_Hz	Prob_OK
0.001444	20.000000	0.130000	0.157778	307.495483	675.823853	368.328369	0.891729
0.001444	26.666666	0.130000	0.157778	252.908340	616.410156	363.501831	0.881293
0.001444	13.333333	0.130000	0.157778	350.691467	713.178162	362.486694	0.920168
0.001444	33.333332	0.130000	0.157778	199.779282	560.442566	360.663269	0.886564
0.001667	26.666666	0.130000	0.175556	269.310944	629.220154	359.909210	0.919665
0.001444	6.666667	0.130000	0.157778	395.206787	753.958130	358.751343	0.862449
0.001444	13.333333	0.137778	0.157778	345.799835	704.295593	358.495758	0.832567
0.001667	26.666666	0.137778	0.175556	262.713257	615.369141	352.655884	0.883494
0.001667	33.333332	0.130000	0.175556	213.470718	565.648376	352.177673	0.930707
0.001889	40.000000	0.130000	0.193333	193.703735	543.879211	350.175476	0.837110
0.001222	13.333333	0.130000	0.157778	331.401337	679.932617	348.531281	0.967976
0.001222	20.000000	0.130000	0.157778	273.139160	620.956970	347.817810	0.947797
0.001444	20.000000	0.130000	0.175556	287.784607	633.443970	345.659363	0.976736
0.001667	33.333332	0.137778	0.175556	205.924973	550.130981	344.205994	0.880593
0.001667	40.000000	0.130000	0.175556	173.367203	515.655823	342.288635	0.940032

Figura 62: Configurazioni che massimizzano l'ampiezza della band gap con n=10

In questa fase di validazione si sono ottenuti risultati molto positivi, partendo dal fatto che sono state confermate, dal codice Matlab, come valide tutte e 15 le configurazioni riportate sopra, ottenendo, quindi, un tasso di conferma pari al 100%.

Conf	r	Alpha	A	L	BG_start	BG_end	Err%_start	Err%_end
1	0.0014	20.00	0.1300	0.1578	282.00	675.00	9.04	0.12
2	0.0014	26.67	0.1300	0.1578	224.00	610.00	12.91	1.05
3	0.0014	13.33	0.1300	0.1578	371.00	759.00	5.47	6.04
4	0.0014	33.33	0.1300	0.1578	183.00	562.00	9.17	0.28
5	0.0017	26.67	0.1300	0.1756	232.00	634.00	16.08	0.75
6	0.0014	6.67	0.1300	0.1578	516.00	852.00	23.41	11.51
7	0.0014	13.33	0.1378	0.1578	356.00	719.00	2.87	2.04
8	0.0017	26.67	0.1378	0.1756	219.00	601.00	19.96	2.39
9	0.0017	33.33	0.1300	0.1756	192.00	586.00	11.18	3.47
10	0.0019	40.00	0.1300	0.1933	170.00	546.00	13.94	0.39
11	0.0012	13.33	0.1300	0.1578	314.00	642.00	5.54	5.91
12	0.0012	20.00	0.1300	0.1578	239.00	571.00	14.28	8.75
13	0.0014	20.00	0.1300	0.1756	250.00	598.00	15.11	5.93
14	0.0017	33.33	0.1378	0.1756	181.00	554.00	13.77	0.70
15	0.0017	40.00	0.1300	0.1756	163.00	544.00	6.36	5.21

Figura 63: Risultati validazione Matlab con n=10

Per quanto riguarda i valori di band gap, quindi, si sono ottenuti i seguenti risultati:

	BG_start	**BG_end**
Errore medio	11,94%	3,64%
Errore massimo	23,41%	11,51%
Errore minimo	2,87%	0,12%

Figura 64: Risultati consuntivi validazione configurazioni n=10

Dagli errori ottenuti, per le singole configurazioni, ciò che si nota è che per Alpha = 6,67 gradi si hanno gli errori maggiori, pari a 23,41% su BG_start e 11,51% su BG_end, questo è dipeso dal fatto che, come già più volte affermato, questa è una zona poco campionata, infatti, ci sono pochissime configurazioni che hanno Status = Ok, che abbiano un angolo piccolo. Di conseguenza, avendo meno campioni su cui essere addestrata, la rete tende ad avere un errore maggiore ed ad essere meno prevedibile.

Parallelamente le due configurazioni per le quali si sono ottenuti i risultati migliori sono:

	Configurazione 3	**Configurazione 7**
Errore BG_start	2,87%	2,04%
Errore BG_end	5,47%	6,04%

Figura 65: Figura con i risultati di validazione migliori per n=10

Questi risultati, significativamente migliori rispetto a quelli ottenuti in precedenza, possono essere ricondotti al motivo opposto rispetto a quello citato precedentemente. Le configurazioni considerate, infatti, si trovano in una regione dello spazio, con una maggiore densità di valori, motivo per il quale le reti hanno avuto modo di imparare bene i valori e le configurazioni. Ciò ha permesso di ottenere prestazioni migliori. Inoltre, queste configurazioni, si collocano vicino all'ottimo identificato e descritto nelle sezioni precedenti, fornendo un ulteriore conferma del corretto funzionamento del modello.

- Il secondo set di dati è stato composto allo stesso modo del primo, ma con un numero di divisioni pari a 50.

  In questo caso le configurazioni che sono state testate direttamente sul codice Matlab sono le seguenti:

r	Alpha	A	L	BG_start_Hz	BG_end_Hz	BG_width_Hz	Prob_OK
0.001612	25.714285	0.130000	0.159592	283.582794	661.580383	377.997589	0.800234
0.001571	25.714285	0.130000	0.159592	279.862183	657.408875	377.546692	0.829574
0.001571	25.714285	0.131429	0.159592	279.293854	656.411743	377.117889	0.809310
0.001571	24.489796	0.130000	0.159592	288.563599	665.319824	376.756226	0.831577
0.001612	24.489796	0.130000	0.159592	291.801270	668.136353	376.335083	0.801072
0.001490	24.489796	0.130000	0.156327	281.891083	658.161804	376.270721	0.823307
0.001571	24.489796	0.131429	0.159592	287.995605	664.244324	376.248718	0.812005
0.001327	18.367348	0.130000	0.149796	315.341370	691.491394	376.150024	0.838781
0.001653	26.938776	0.130000	0.162857	275.630585	651.668274	376.037689	0.804483
0.001612	25.714285	0.130000	0.162857	281.745667	657.589355	375.843689	0.852970
0.001367	18.367348	0.130000	0.149796	318.965454	694.650146	375.684692	0.802546
0.001653	25.714285	0.130000	0.162857	284.986145	660.632080	375.645935	0.809238
0.001531	24.489796	0.130000	0.159592	284.548248	660.126343	375.578094	0.851876
0.001612	25.714285	0.131429	0.162857	281.263947	656.805359	375.541412	0.835312
0.001327	19.591837	0.130000	0.149796	304.770416	680.293335	375.522919	0.826364

Figura 66: Configurazioni che massimizzano l'ampiezza delle band gap con n=50

Validando le precedenti configurazioni su Matlab si sono ottenuti i seguenti risultati:

Conf	r	Alpha	A	L	BG_start	BG_end	Err%_start	Err%_end
1	0.0016	25.71	0.1300	0.1596	invalid	-	-	-
2	0.0016	25.71	0.1300	0.1596	249.00	667.00	12.39	1.44
3	0.0016	25.71	0.1314	0.1596	246.00	658.00	13.53	0.24
4	0.0016	24.49	0.1300	0.1596	259.00	678.00	11.41	1.87
5	0.0016	24.49	0.1300	0.1596	invalid	-	-	-
6	0.0015	24.49	0.1300	0.1563	251.00	655.00	12.31	0.48
7	0.0016	24.49	0.1314	0.1596	256.00	670.00	12.50	0.86
8	0.0013	18.37	0.1300	0.1498	292.00	667.00	7.99	3.67
9	0.0017	26.94	0.1300	0.1629	invalid	-	-	-
10	0.0016	25.71	0.1300	0.1629	250.00	672.00	12.70	2.14
11	0.0014	18.37	0.1300	0.1498	301.00	687.00	5.97	1.11
12	0.0017	25.71	0.1300	0.1629	invalid	-	-	-
13	0.0015	24.49	0.1300	0.1596	252.00	661.00	12.92	0.13
14	0.0016	25.71	0.1314	0.1629	247.00	664.00	13.87	1.08
15	0.0013	19.59	0.1300	0.1498	278.00	652.00	9.63	4.34

Figura 67: Configurazioni e risultati validazione Matlab con n = 50

Come si nota, delle configurazioni che il modello di machine learning aveva predetto come Status = OK, 11 sono state confermate, mentre 4 sono state classificate come Invalide. Questo dipende dal fatto che esse si trovano nella zona di transizione più problematica per il modello, ossia quello con angoli di piegatura compresi tra 25 e 27 gradi.

Sicuramente, una delle osservazioni più importanti risultante dall'analisi fatta, riguarda un andamento degli errori commessi dal modello, che è ben definito. In particolare, si ha che BG_start risulta essere sempre maggiore rispetto a BG_end.

Infatti:

	BG_start	**BG_end**
Errore medio	11,38%	1,58%
Errore massimo	13,87%	4,34%
Errore minimo	5,97%	0,13%

Figura 68: Risultati consuntivi validazione Matlab con n=50

In realtà, questa tendenza, ha una motivazione ben precisa. Come si può osservare dalle figure precedenti, tutte le configurazioni che sono state testate sono concentrate in una specifica zona di spazio (con angolo di piega tra 24 e 27 gradi, A pari a 0,13 e L compreso tra 0,1498 e 0,1629), questa è una zona in cui la rete ha dimostrato di essere molto precisa nella predizione del valore di fine della banda di stop, ma al contempo di avere maggiori difficoltà nella predizione di BG_start, in quanto è un valore più sensibile ai cambiamenti dell'angolo di

piegatura, anche una variazione di 1-2 gradi dell'angolo, possono determinare un incremento significativo dell'errore commesso.

Differentemente, le configurazioni 8 e 11, hanno un valore di Alpha = 18,37 gradi, che risulta essere più distante dalla zona di transizione, precedentemente descritta. Di conseguenza, la rete neurale riesce ad essere più accurata e precisa riuscendo ad ottenere risultati migliori.

- Per il terzo set, invece, è stato fatto un ragionamento diverso, in quanto si era notato che le configurazioni che venivano testate erano concentrate in determinate regioni dello spazio, quindi, avevano valori dei parametri di input molto simili tra loro. Per superare questo problema, si è deciso di scegliere manualmente, sempre 15 configurazioni, con valori dei parametri di input che permettessero di coprire zone diverse dello spazio, per avere una copertura più uniforme e verificare l'efficacia del codice anche in queste zone.
  Per far in modo di andare a coprire tutto lo spazio disponibile, sono state testate le configurazioni riportate nella figura seguente, le quali hanno ottenuto i seguenti risultati:

Conf	r	Alpha	A	L	BG_start	BG_end	Err%_start	Err%_end
1	0.0014	15.92	0.1300	0.1465	363.00	777.00	4.29	6.54
2	0.0010	20.00	0.1500	0.2000	132.00	316.00	5.95	6.53
3	0.0020	20.00	0.1500	0.2000	264.00	633.00	6.90	12.07
4	0.0030	20.00	0.1500	0.2000	invalid	-	-	-
5	0.0015	10.00	0.1500	0.2000	294.00	543.00	3.56	3.62
6	0.0015	20.00	0.1500	0.2000	198.00	475.00	16.74	3.79
7	0.0015	30.00	0.1500	0.2000	144.00	418.00	9.94	2.59
8	0.0015	45.00	0.1500	0.2000	100.00	353.00	0.38	9.03
9	0.0015	60.00	0.1500	0.2000	77.00	286.00	0.49	8.41
10	0.0015	20.00	0.1300	0.1800	252.00	601.00	14.34	4.76
11	0.0015	20.00	0.2000	0.2500	120.00	288.00	15.04	1.45
12	0.0015	20.00	0.1500	0.3000	114.00	241.00	2.91	7.72
13	0.0010	10.00	0.1300	0.1500	323.00	596.00	1.63	3.85
14	0.0030	40.00	0.1800	0.2500	invalid	-	-	-
15	0.0020	30.00	0.1600	0.2200	164.00	472.00	11.23	9.46

Figura 69: Configurazioni e risultati testati su codice Matlab con distribuzione su tutto lo spazio

Come è possibile notare, delle 15 configurazioni fornite al codice originale, Matlab ne ha confermate, come valide, 13. Una particolarità da osservare è che le 2 configurazioni che ha classificato come "Invalid" hanno, entrambe, valore di raggio della sezione, pari a 0.003, che risulta essere un valore al limite, rappresentante una struttura molto rigida, che spesso viola i vincoli, caratteristica che era già stata osservata durante la fase di analisi esplorativa dei dati.

Quindi, per queste configurazioni, i risultati che si sono ottenuti sono i seguenti:

	BG_start	BG_end
Errore medio	7,18%	6,14%
Errore massimo	16,74%	12,07%
Errore minimo	0,38%	1,45%

Figura 70: Risultati validazione con configurazioni su tutto lo spazio

A differenza dei casi precedenti, ciò che si è notato è che le differenze tra gli errori, soprattutto per quanto riguarda il valore medio, sono meno evidenti e i due valori non si discostano di tanto. Nei casi precedenti il pattern che si osservava era che l'errore commesso su BG_start era sempre maggiore rispetto a quello su BG_end, in questo caso invece, avendo configurazioni distribuite su tutto lo spazio, l'errore tende a distribuirsi in maniera più uniforme.

Un'osservazione molto importante che è stata fatta è che: per le configurazioni 8 e 9, si ottengono valori di BG_start davvero bassi, nonostante l'angolo di piegatura abbia valori più elevati. Questo sembra contro intuitivo rispetto quanto detto fino ad ora, ma in realtà la spiegazione è che la rete è riuscita ad imparare molto bene la relazione che caratterizza questo spazio, anche con la scarsa presenza di campioni. Al contrario, invece, BG_end tende ad assumere valori maggiori, evidenziando una maggiore difficoltà nella predizione della frequenza di fine della banda di inibizione.

Inoltre, tra le configurazioni inserite manualmente, è stata aggiunta anche la quella ottimale trovata dal modello, per la quale si sono ottenuti i seguenti risultati:

- BG_start = 363 Hz
- BG_end = 777 Hz

Di conseguenza il modello ha predetto un valore di BG_width = 777 – 363 = 414 Hz, mentre Matlab ha predetto, come valore reale, BG_width = 423,51 Hz. La differenza tra i due valori è del 2,25%, che ribadisce, nuovamente, gli ottimi risultati e l'elevata accuratezza del modello.

Un'ultima osservazione che è stata fatta è che le configurazioni 6, 10 e 11, le quali hanno tutte angolo di piegatura pari a 20 gradi, ma diversi valori per gli altri parametri, è che hanno un errore, su BG_start, che è molto simile tra loro, compreso tra 15% e 16%. Questo ha permesso di capire che essa è una zona molto complessa e poco lineare, non è una zona di transizione verso delle configurazioni invalide, quanto più una zona dove la rete ha maggiori difficoltà e avrebbe bisogno di più configurazioni per imparare meglio la relazione presente.

Successivamente, per verificare la riproducibilità dei risultati, sono state testate ulteriori 30 configurazioni, di cui 15 distribuite su tutto lo spazio, mentre le altre 15 che

massimizzavano la dimensione della banda di stop. Anche in questo caso si sono ottenuti risultati molto simili a quelli già osservati in precedenza per gli altri set di dati.

Riassumendo i risultati finali ottenuti:

	Configurazioni validate	Errore medio BG_start	Errore medio BG_end
Set configurazioni n=10	100%	11,94%	3,64%
Set configurazioni n=50	73%	11,38%	1,58%
Primo set configurazioni distribuite	87%	7,18%	6,14%
Secondo set configurazioni distribuite	87%	8,38%	5,77%
Set configurazioni BG_width massima	100%	13,95%	2,72%

Figura 71: Risultati finali validazione Matlab

La fase di validazione tramite il codice Matlab, ha permesso di dimostrare che il modello surrogato realizzato, funziona bene anche per svolgere il processo inverso rispetto a quello per il quale è stato realizzato, quindi determinare i parametri che massimizzano la band gap. Infatti l'errore medio complessivo su BG_start risulta essere 10,57%, mentre su BG_end 3,97%.

3.6.4 Ottimizzazione inversa con vincolo sulla frequenza

In questa fase, infine, è stato introdotto un ultimo elemento, ossia un vincolo sulla frequenza. L'obiettivo al quale si voleva arrivare, era di riuscire a definire i valori di inizio e di fine della massima banda di stop, con una frequenza centrata in un valore definito dall'operatore.

Il processo relativo a questo nuovo passaggio è iniziato con la definizione della frequenza target che si vuole raggiungere, che risulta essere un valore modificabile dall'operatore, in base alle proprie esigenze. È stata fissata, inoltre, una tolleranza, infatti, risulterebbe molto complesso richiedere alla rete di predire il valore preciso. L'introduzione di una certa tolleranza, invece, consente di ottenere buoni risultati con costi computazionali inferiori.

Successivamente, partendo dalle configurazioni che erano già state filtrate in fase di ottimizzazione inversa, quindi: avevano Status = OK, con una confidenza di almeno il 80% e BG_start < BG_end, si vanno a filtrare quelle combinazioni che hanno BG_mean vicino al target fissato. Fatto questo si cercano tutte quelle configurazioni che hanno BG_mean all'interno dell'intervallo [frequenza target – tolleranza; frequenza target + tolleranza]. A questo punto l'ultimo compito svolto dall'algoritmo è di selezionare, tra tutte le configurazioni rimaste, se esiste, quella che massimizza l'ampiezza della banda di stop.

Fissando i range dei parametri uguali ai test compiuti prima e un numero di divisioni pari a 10, con una frequenza target pari a 200 Hz e una tolleranza di 50 Hz, si sono ottenuti i seguenti risultati per la configurazione ottima:

- r: 0,001222
- Alpha: 60,00 gradi
- A: 0,1378 m^2
- L: 0,14 m

Con valori della band gap pari a:

- BG_start: 106,84 Hz
- BG_end: 378,32 Hz
- BG_width: 271,49 Hz
- BG_mean: 242,58 Hz

Ovviamente, come citato, tutti i valori sono stati resi personalizzabili in base alle esigenze dell'operatore. Minore sarà la tolleranza accettata, maggiore sarà la precisione verso la frequenza target, ma al tempo stesso aumenteranno i costi computazionali, oltre alla possibilità che la banda di stop con le caratteristiche richieste non esista.

Fissando gli stessi valori precedenti, ma con un numero di divisioni pari a 50, è stato compiuto un ulteriore test, che ha prodotto i seguenti risultati:

- r: 0,001245
- Alpha: 60,00 gradi
- A: 0,1314 m^2
- L: 0,1433 m

Con valori della band gap pari a:

- BG_start: 110,53 Hz
- BG_end: 389,41 Hz
- BG_width: 278,88 Hz
- BG_mean: 249,97 Hz

Come si nota la configurazione che è stata trovata, con un numero di divisioni maggiori, è leggermente diversa, rispetto a quella trovata con n = 10, questo dipende dal fatto che il modello ha a disposizione un numero maggiore di configurazioni tra le quali ricercare quella che massimizza la BG_width. Una cosa importante da notare di questa seconda configurazione trovata è che si trova quasi al limite di tolleranza fissato a 50 Hz, ciò significa che si abbassasse questo valore, sicuramente il modello troverebbe una nuova configurazione.

3.7 Introduzione del rumore

3.7.1 Motivazioni alla base dell'introduzione del rumore nei dati di input

Una volta validato il modello surrogato di machine learning, che è stato realizzato, dimostrandone efficacia e accuratezza, è stato possibile introdurre un altro elemento che fino a questo momento non era stato considerato. Infatti, le macchine manifatturiere, che lavorano i metamateriali, hanno delle tolleranze, ciò fa sì che non sia possibile ottenere dei valori perfettamente uguali a quelli fissati, ma a volte possono esserci variazioni, più o meno consistenti. È importante, quindi, capire se esse possono incidere sulla formazione della band gap o sul corretto funzionamento del modello realizzato. Motivo per il quale si è deciso di introdurre delle misure di rumore, all'interno dell'algoritmo, per verificare se continuasse a funzionare correttamente, mantenendo la propria robustezza.

Il primo passaggio effettuato è stato definire dei range sensati di rumore, che una macchina potrebbe applicare durante l'esecuzione dei propri compiti, quindi si è deciso di scegliere i seguenti valori:

- Per r ±1%.
- Per Alpha ±2 gradi.
- Per A ±2%.
- Per L ±2%.

È importante notare che solo l'angolo di piegatura ha un valore di range di tolleranza in gradi, ciò dipende dal fatto che una deviazione angolare definita, ha lo stesso significato, indipendentemente dal valore nominale, mentre, per gli altri valori di input, il valore è in percentuale, in quanto dipende dal valore nominale del parametro.

Come per ogni elemento inserito all'interno della tesi, anche in questo caso, sono stati realizzati due notebook di Google Colab, nei quali sono state importate le seguenti librerie, oltre a quelle già citate nelle sezioni precedenti:

- Modulo pd di pandas, che permette di lavorare su valori e dati raccolti in tabelle, o fogli di calcolo, come quelli Excel.
- Modulo np di numpy, che permette, anch'esso, di lavorare su dati vettoriali o matriciali.
- Accuracy_score, f1_score, classification_report, mean_absolute_error, r2_score, che sono tutte funzione che permettono la valutazione del modello che è stato realizzato.

Una volta importate tutte le librerie necessarie e definiti i percorsi ai file relativi a classificatore, regressore, scaler e ai risultati trovati, come si era già visto anche per l'ottimizzazione inversa, sono state ricostruite le reti, un'osservazione importante da fare è che tutti i valori dei parametri sono stati impostati uguali alla fase di training.

Successivamente a questi passi introduttivi sono state seguite alcune fasi principali, che saranno riportate nelle sotto-sezioni successive.

3.7.2 Fase 1: introduzione del modello di rumore gaussiano

La prima fase che è state implementata è l'introduzione del rumore all'interno del modello, in particolare nel dataset. Una cosa importante da ricordare è che il codice Matlab è molto lento nel produrre la griglia di configurazioni inziali. Motivo per cui si è scelto, inizialmente, di non generare un nuovo dataset con i valori perturbati da Matlab, ma, prima, di effettuare un controllo sulla robustezza del modello direttamente dal dataset originale, per riuscire a rispondere alla domanda: "Fornendo valori di input leggermente differenti, quindi contenenti del rumore, le predizioni e performance della rete cambiano radicalmente o subiscono piccole variazioni trascurabili?". Quindi, come prima cosa, sono stati perturbati, con le misure di rumore riportate sopra, solo i valori di input del set di configurazioni originali, mantenendo uguali i valori di output. Questa può sembrare una tecnica poco significativa, dal punto di vista fisico, in quanto, modificando i valori di input, di conseguenza bisognerebbe ottenere una modifica anche nei valori di output. In realtà, questa analisi ha permesso di dimostrare se il modello, con l'introduzione di valori di incertezza, continuasse a rimanere robusto o avesse la necessità di modifiche o un eventuale nuovo addestramento. In un secondo momento, poi, sono stati generati anche i nuovi valori di output, che si definiscono da dati di input con presenza di rumore, direttamente tramite Matlab.

Ogni misura di rumore è stata introdotta come una variabile a sé stante così che possa essere liberamente modificata, in base alle diverse tolleranze che le macchine possono avere su materiali differenti. È possibile testare il caso in cui una macchina possa non avere effetti sulle lavorazioni che effettua e, quindi, avere una misura di rumore pari a 0.

La scelta di partire da un dataset perturbato solo nei sui valori di input è stata, quindi, una scelta basata sulla convenienza a livello di costi computazionali, perturbando un set di dati già esistente è stato, così, possibile risparmiare tempo, che sarebbe stato, altrimenti, impiegato da Matlab per la generazione di nuovi risultati, che, avrebbero potuto rivelarsi poco significativi, nel caso in cui avessero mostrato criticità nel modello.

Il rumore è stato introdotto tramite una distribuzione gaussiana centrata in zero e con deviazione standard calcolata con la regola del 3-sigma, per far in modo che il 99,7% dei valori ricadesse nel range. Si è scelta questa tipologia di distribuzione in base alla definizione del teorema del limite centrale, il quale afferma che: se diverse sorgenti di errore si sommano tra loro, come nel caso di tolleranze di macchine manifatturiere, si viene a definire una distribuzione gaussiana.

Per ogni parametro, quindi, è stata utilizzata la seguente formula:

$$\text{valore di input perturbato} = \text{valore originale} + \mathrm{N}(\sigma; 0)$$

Equazione 37: Equazione valore di input perturbato

Con:

- N(σ;0): pari al valore estratto da una distribuzione gaussiana, che ha media uguale a zero e deviazione standard pari a 1.

Tramite una funzione, sono stati definiti 5 set di dati, con parametri di input perturbati. Questa scelta è stata fatta per garantire maggiore affidabilità statistica, in quanto, se si fosse generato un solo dataset, estraendo valori casuali da una gaussiana, si sarebbe rischiato: o di avere valori di perturbazioni molto alti per configurazioni al limite, che avrebbero, dunque, generato delle prestazioni molto scarse, oppure si sarebbero potuti estrarre valori di perturbazione piccolissimi, che avrebbero generato performance alte, ma poco significative. Per queste ragioni è stato deciso di generare 5 dataset perturbati, con seed differenti, così da avere 5 set di dati completamente diversi tra loro. Il valore 5 è stato scelto come trade-off, tra affidabilità statistica e velocità di calcolo. Per ognuno dei dataset generati, sono state calcolate le misure di performance, dalla media delle quali è stato possibile calcolare i valori consuntivi delle metriche di valutazione della performance del modello. Questi 5 dataset formati, quindi, uniti insieme generavano il nuovo dataset utilizzato, formato da un totale di 15.000 configurazioni, ossia 3.000 combinazioni per ognuno dei set generati.

3.7.3 Fase 2: valutazione dei risultati della rete sul dataset originale con dati perturbati

Una volta inserito correttamente il rumore all'interno dei dati, ed aver effettuato un clipping per evitare la generazione di configurazioni non possibili, si è passati a valutare le performance del classificatore e del regressore, con l'unica differenza, come già più volte citato, che per il secondo task sono state utilizzate solo le configurazioni con Status = OK.

Partendo dal classificatore, quindi, gli sono stati forniti i 5 set di dati perturbati solo negli input che sono stati generati nella prima fase e sono state calcolate le metriche di performance più note, le quali hanno generato i seguenti risultati:

	Accuracy	**F1-score**
Dataset 1	98,1%	0,9704
Dataset 2	97,9%	0,9724
Dataset 3	98,0%	0,9732
Dataset 4	97,9%	0,9690
Dataset 5	97,8%	0,9682

Figura 72: Risultati classificatore sulle 5 versioni di dataset perturbati negli input

I valori riportati nella tabella precedente sono le misure di performance di accuratezza e f1-score per le 5 versioni di test set realizzate.

Da tali valori è possibile calcolare il valore consuntivo di tali metriche, dato dalla media di esse:

- Accuratezza: 98,0%.
- F1-score macro: 0,9707.

Testando lo stesso modello sul test set originale, quindi, quello in cui non sono stati perturbati i dati, invece, si sono ottenute le seguenti misure di performance:

- Accuratezza: 98,1%.
- F1-score: 0,9738.

A questo punto è stata calcolata la differenza di performance della rete dedicata alla classificazione (ClassifierNet), calcolando il delta tra le misure di performance evidenziate dal test set originale e quelle consuntive, nate dalla media del valore delle metriche delle 5 versioni realizzate. I risultati ottenuti sono i seguenti:

- ΔAccuratezza: - 0,15%.
- ΔF1-score: - 0,0032.

Ciò significa che c'è un leggero peggioramento delle prestazioni, ma risulta essere talmente basso che è, praticamente, trascurabile. Come soglia di accettabilità, infatti, era stato fissato 2%, oltre tale valore si sarebbero dovute applicare delle modifiche nella struttura della rete.

La stessa valutazione è stata effettuata anche sul regressore, per le 5 versioni di test set perturbato, si sono ottenuti i seguenti risultati per il primo output, ossia BG_start:

	R^2	MAPE
Dataset 1	0,9655	6,07%
Dataset 2	0,9657	6,07%
Dataset 3	0,9670	6,07%
Dataset 4	0,9664	6,01%
Dataset 5	0,9660	6,08%

Figura 73: Risultati metriche di performance del regressore BG_start sulle 5 versioni di dataset perturbato

	R^2	MAPE
Dataset 1	0,9565	5,75%
Dataset 2	0,9564	5,89%
Dataset 3	0,9560	5,81%
Dataset 4	0,9574	5,74%
Dataset 5	0,9564	5,84%

Figura 74: Risultati metriche di performance del regressore BG_end sulle 5 versioni di dataset perturbato

Da tali valori è stato possibile calcolare delle misure consuntive, date dalla media dei valori delle metriche nelle 5 versioni realizzate. Per BG_start si è ottenuto:

- R^2: 0,9661.
- MAPE: 6,06%.

Mentre per BG_end:

- R^2: 0,9565.
- MAPE: 5,81%.

Per il test set originale, invece, si sono ottenuti i seguenti valori:

	BG_start	**BG_end**
R^2	0,9678	0,9571
MAPE	5,76%	5,78%

Figura 75: Valore metriche performance dataset originale

Come nel caso del classificatore è stato, poi, possibile calcolare la variazione, pari a:

- ΔMAPE: +0,30% per BG_start.
- ΔMAPE: +0,03% per BG_end.
- ΔR^2: - 0,0017 per BG_start.
- ΔR^2: - 0,0006 per BG_end.

Anche in questo caso, quindi, si è osservata una differenza davvero minimale, sotto alla soglia di accettabilità, fissata sempre al 2%, motivo per il quale si è dimostrato che anche RegressorNet, continui a funzionare correttamente nonostante l'introduzione del rumore.

Per mostrare meglio i risultati sono stati creati anche alcuni grafici riportati di seguito:

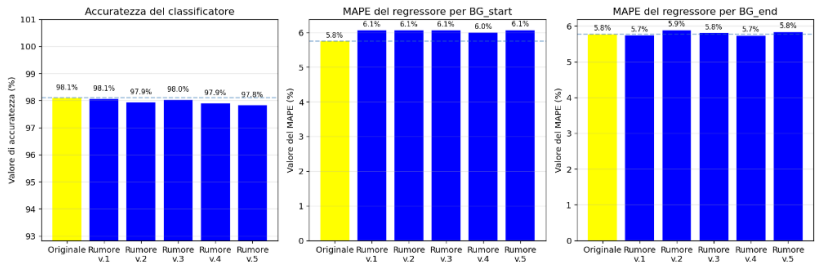


Figura 76: Confronto misure di performance tra test set originale e le 5 versioni perturbate

I tre grafici riportati nella figura precedente mostrano delle barre rappresentati i valori delle metriche di performance (in particolare si sono scelte accuratezza, per la classificazione, e MAPE per la regressione). La barra in giallo rappresenta la metrica del test set originale, mentre quelle in blu sono le 5 versioni del test set perturbato. L'idea del grafico è mostrare in maniera più chiara che i valori di performance che si ottengono sono praticamente identici, esse infatti presentono solo leggere differenze che sono del tutto trascurabili. Quindi, è stato possibile dimostrare che il modello è stabile e robusto, totalmente indipendente dalla misura di rumore introdotta, non richiedendo, quindi, un re training delle reti, dopo l'introduzione del rumore.

3.7.4 Fase 3: generazione del nuovo dataset con valori perturbati tramite il codice MATLAB

Una volta che sono state compiute tutte le analisi sui dataset con solo parametri di input perturbati e dopo aver dimostrato la validità del modello creato anche successivamente all'introduzione di rumore, è stato necessario, per questioni di validità fisica, generare un dataset che avesse sia i valori di input, che i corrispondenti valori di output perturbati, e non solo i dati di input come fatto fino ad ora. Come spiegato nelle sezioni precedenti, la scelta di implementare il rumore solo per i valori di input è dettata dal fatto di voler in primo luogo validare e dimostrare la robustezza dell'algoritmo, a costi computazionali ridotti, per poi, in un successivo minuto, rendere più accurate le analisi.

A questo punto, quindi, è stato deciso di inserire il rumore direttamente all'interno del codice Matlab, del quale si conosce l'estrema precisione e affidabilità, sempre seguendo una distribuzione gaussiana, così che, a partire dai valori di input con rumore, potesse calcolare i valori di output, che potevano essere diversi rispetto agli originali, quindi la stop band poteva avere valori di frequenza differenti. Utilizzando gli stessi range di incertezza citati nelle sotto sezioni precedenti, si sono andati a introdurre le misure di rumore all'interno del codice Matlab originale, anch'esse sono state inserite come valori parametri, così che l'operatore fosse libero di personalizzarli e modificarli a proprio piacimento, in base alle necessità.

A questo punto, definiti range di valori di input, intervalli di rumore e divisioni, il codice ha generato una griglia di valori, dati, come sempre, dal prodotto scalare delle divisioni dei parametri di input. Quello che si è formato era, quindi, il nuovo dataset, che, a differenza del precedente aveva sia valori di input che i valori di output perturbati e adattati. Una caratteristica importante da notare è che il nuovo dataset aveva più colonne, in quanto, si è deciso di mantenere nella griglia sia le colonne con i valori di input originali, che quelle con i valori perturbati.

Generato questo nuovo set di dati, il primo controllo che è stato effettuato è se le percentuali delle classi fossero cambiate. Poteva darsi, infatti, che introducendo delle leggere variazioni all'interno dei valori di input, la configurazione che si veniva a formare, cambiasse classe, soprattutto se era una combinazione ai limiti.

Le nuove percentuali che si sono definite sono:

	Dataset originale	Dataset perturbato	Variazione
Classe OK	2.625 (26.3%)	2.706 (27.1%)	+81 (+0,8%)
Classe No BG	318 (3.2%)	183 (1.8%)	-135 (-1,4%)
Classe Invalid	7.057 (70.6%)	7.111 (71.1%)	+54 (+0,5%)

Figura 77: Divisioni classi dataset perturbato

Come si nota dai dati riportati in tabella, quindi, si hanno delle modifiche nelle distribuzioni delle classi:

- La classe OK, guadagna 81 configurazioni.
- La classe No BG, perde 135 configurazioni, questo aggrava notevolmente il problema relativo alla scarsità di campioni appartenenti a questa classe.
- La classe Invalid acquisisce 54 configurazioni, che quindi passano da essere No BG, quindi essere valide fisicamente, ma senza band gap, ad essere invalide fisicamente.

3.7.5 Fase 4: confronto tra risultati ottenuti con il dataset originale e con quello perturbato

Definito il nuovo dataset sono state selezionate solo le configurazioni che sono risultate appartenenti alla classe OK, sia nel set di dati perturbato che in quello originale, nello specifico sono state selezionate 2478 configurazioni, su 2625 configurazioni OK del dataset originale e 2706 per quello con valori rumorosi.

L'idea, a questo punto, era quella di riuscire a fare un confronto tra i dati di output generati da i due dataset, per riuscire a rispondere alla domanda: "Quanto cambiano i valori di output generati da Matlab se si inserisce rumore nei dati di input?", quindi, introducendo rumore quanto cambiano i valori di output. Da questa analisi sono stati estratti i seguenti risultati:

	Variazione media assoluta	Variazione media percentuale	Variazione percentuale massima	Deviazione standard percentuale
BG_start	2,37 Hz	1,53%	15,60%	1.46%
BG_end	3,24 Hz	1,01%	5,92%	0.82%
BG_width	2,35 Hz	1,34%	14,17%	/

Figura 78: Variazioni valori di output dataset originale e perturbato

I risultati ottenuti si sono dimostrati soddisfacenti, evidenziando la robustezza del modello fisico, in quanto la variazione media, sia per BG_start che per BG_end, è inferiore a 2%. Valori più alti, si sono trovati nella variazione percentuale massima, ciò dipende dalla presenza di configurazioni in zone limite, o di transizione, come configurazioni con angoli di piegatura piccoli, o misure vicine al vincolo tra A e L. Queste configurazioni erano già state identificate come problematiche, in quanto più sensibili, soprattutto dopo l'introduzione di misure di rumore.

Come già fatto in altri casi, anche per l'introduzione del rumore è stato deciso di realizzare dei grafici per dimostrare meglio l'effetto che il rumore ha avuto sui valori di output del modello. I grafici sono molto utili per avere una rappresentazione grafica semplice e intuitiva:

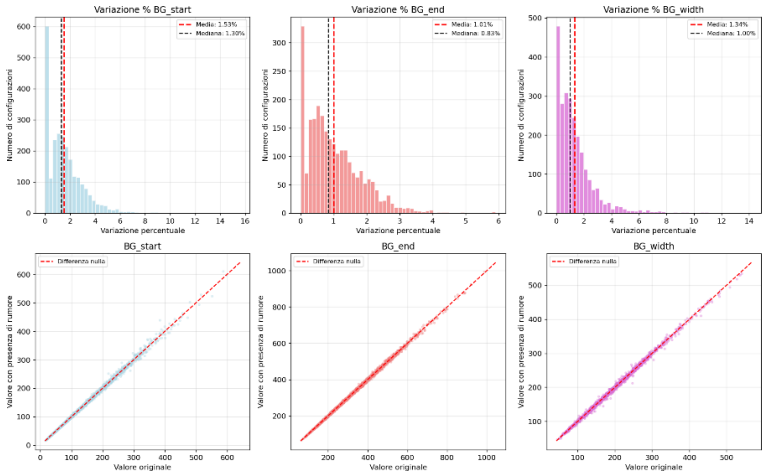


Figura 79: Effetto rumore su valori di output

Nella prima riga della figura precedente sono riportate le distribuzioni delle variazioni percentuali dei 3 più importanti valori di output del modello, quindi BG_start, BG_end e BG_width. Nei grafici, poi, sono riportate anche media e mediana. In questo caso le distribuzioni sono state calcolate su valori che non hanno subito una trasformazione logaritmica, motivo per il quale si sono riscontrate asimmetrie. Nella seconda riga, invece, sono stati realizzati 3 scatterplot, uno per ogni valore di output, che mostrano la relazione che sussiste tra il valore dell'output originale e quello generato da valori di input perturbati. Con la linea rossa tratteggiata si indica la relazione ottima, quindi, il caso di variazione nulla. Come si può notare i punti seguono perfettamente l'andamento della retta, dimostrando che l'introduzione del rumore non incide sulla performance del modello.

Dimostrata la validità e robustezza del modello fisico e dei risultati generati da Matlab, è stato svolto il confronto più importante ossia valutare le prestazioni del modello realizzato sull'attività di predizione su un dataset completamente perturbato, cosa che non era ancora stata implementata. Per fare ciò, quindi, è stato preso il dataset generato da Matlab e perturbato sia nei suoi valori di input che in quelli di output, ed è stato passato sia alla rete dedicata alla classificazione, che, successivamente, alla rete dedicata alla regressione,

le quali, nel frattempo, sono state ricostruite utilizzando gli stessi valori nei parametri, definiti in fase di training. Fornito alle reti il dataset, esse hanno prodotto i valori di output, i quali sono stati confrontati con i valori generati dal codice Matlab, facendo un confronto sulle metriche di performance ottenute e riportate nella seguente tabella:

	Dataset perturbato	**Dataset originale**
R^2 BG_start	0,9635	0,9528
MAPE BG_start	5,94%	6,88%
R^2 BG_end	0,9600	0,9582
MAPE BG_end	5,71%	5,88%

Figura 80: Tabelle confronto metriche performance dataset originale e perturbato

Come si nota anche dalle metriche, i valori che si definiscono, tra dataset originale e perturbato, sono molto simili e presentano leggerissime variazioni che sono del tutto trascurabili, in quanto tutte al di sotto del 1%. Addirittura, per quanto riguarda la metrica di MAPE, sia per BG_start che per BG_end, si nota un miglioramento del valore e, di conseguenza, una riduzione dell'errore commesso sul dataset perturbato. Questo risultato ha permesso di dimostrare che il modello è riuscito a generalizzare correttamente il dataset, nonostante la presenza di rumore, che non era stata considerata in fase di addestramento. Ciò evidenzia la precisione, efficacia e efficienza del modello.

3.7.6 Conclusioni finali sull'introduzione del rumore

Dall'introduzione del rumore all'interno dei dati è stato possibile dimostrare alcune peculiarità del modello:

- La fisica che è presente alla base del sistema è molto robusta, infatti se vengono introdotte misure di rumore nell'ordine del 1/2%, su uno o più valori di input, si ottengono delle variazioni nei valori delle stop band che sono compresi tra 0,8% e 1,5%, che risultano essere variazioni completamente trascurabili. Quindi l'introduzione del rumore non indebolisce la struttura.
- La stessa robustezza viene mantenuta anche dal modello, composto da due reti neurali, che è stato realizzato. In quanto è stato testato che le reti hanno valori delle metriche di performance che peggiorano, al massimo, di 1% rispetto al modello originale. Quindi il modello surrogato è in grado di mantenere la propria accuratezza e precisione anche dopo l'introduzione di misure di rumore nei dati.

Di conseguenza si può dedurre che le reti non abbiano bisogno di essere riaddestrate, dopo l'inserimento del rumore, ma continuino a funzionare in modo efficiente ed efficacie, senza cali drastici delle performance. Non sarà, quindi, necessario realizzare un modello di ML più complesso, nel momento in cui debbano essere introdotte misure di rumore nei dati.

CONCLUSIONI

Il progetto di tesi, che è stato sviluppato, è nato dalla necessità di creare un modello surrogato, utilizzando il machine learning, per l'ottimizzazione e predizione delle band gap in metamateriali a trave piegati periodicamente. La necessità dell'introduzione di algoritmi di machine learning, in particolare di reti neurali, è nata a causa dei costi computazionali di modelli tradizionali sviluppati tramite software come Matlab, che risulta essere molto lento e può richiedere alcuni minuti nell'analisi di una sola configurazione, rendendo, di conseguenza, impossibile l'esaminazione di centinaia, o addirittura, migliaia di combinazioni di valori. La lentezza, causata dal Dynamic Stifness Method, quindi, è la motivazione cardine per il quale si è reso necessario la definizione di un modello surrogato, composto da reti neurali feed-forward, che permettesse di analizzare un numero molto elevato di configurazioni in pochi secondi, o, al massimo, minuti, riuscendo a mantenere un livello di accuratezza, precisione e efficacia similare a quelli del codice Matlab. Dopo aver analizzato e adeguato i dati presenti all'interno del dataset, utilizzato per l'addestramento delle reti, è stato realizzato il modello, composto da due reti neurali feed-forward completamente indipendenti: ClassifierNet, dedicata alla classificazione delle configurazioni nelle classi specifiche, in grado di raggiungere un'accuratezza del 98%, e RegressorNet, dedicata alla regressione per il calcolo del valore delle frequenze di inizio e di fine delle band gap, che raggiunge un valore di R^2 superiore a 0,95. I valori delle metriche di valutazione mostrano ed evidenziano le ottime prestazioni raggiunte dalle due reti. Successivamente è stato implementato, sfruttando la struttura a reti indipendenti, un modello di ottimizzazione inversa, che permette di: calcolare i valori dei parametri fisici dei metamateriali che permettono di massimizzare l'ampiezza della banda di stop con, inoltre, la possibilità, da parte dell'operatore, di fissare una frequenza centrale come vincolo da rispettare. L'obiettivo, anche in questo caso, è poter sfruttare la forte rapidità del modello di ML, a differenza di quello creato su Matlab. Anche in questo caso i risultati ottenuti sono stati ottimi, denotando una simmetria tra i risultati tra modello surrogato ed originale, pari al 81% e un errore che rimane simile a quello riscontrato in fase di classificazione e predizione. Un ulteriore specifica introdotta è l'implementazione di misure di rumore all'interno dei dati di input, ad oggi, infatti, le macchine manifatturiere applicano delle tolleranze durante la lavorazione dei materiali, che devono essere tenute in considerazione e introdotte per una corretta, e completa, valutazione del modello a reti neurali realizzato. Motivo per il quale è stata effettuata un'analisi sull'introduzione di misure di rumore nel range dell'1-2%, tramite una gaussiana, nei valori di input. Anche in questo caso, dopo un confronto con un modello preciso, efficiente e accurato come quello di Matlab, il modello surrogato ha dimostrato di essere totalmente robusto e possedere prestazioni paragonabili, con un peggioramento minimo, e del tutto trascurabile, nelle misure di valutazione delle prestazioni, sempre al di sotto del 1,5%. Questo lavoro di tesi si basa su un argomento totalmente nuovo e mai implementato in precedenza, nonostante i metamateriali siano un campo, da anni, ampiamente analizzato e studiato, grazie alla loro estrema utilità e importanza. Sono presenti alcuni articoli scientifici riguardanti l'implementazione di modelli di machine learning nel campo dei metamateriali, ma la novità, di questo progetto, risiede nell'implementazione di tali algoritmi per strutture a trave piegate periodicamente.

Durante lo svolgimento del progetto sono state riscontrate alcune limitazioni, che è bene sottolineare, a partire dalla copertura dello spazio. Come citato nel corso della tesi, il dataset utilizzato per l'addestramento delle reti neurali, è stato generato con un numero di divisioni dei parametri pari a 10, motivo per il quale alcune zone dello spazio hanno un campionamento molto basso, come si è notato anche in fase di analisi dei risultati del regressore, infatti le zone con valori di ampiezza delle band gap maggiori, risultano essere quelle con minori configurazioni. Il problema principale che deriva da ciò, è un valore di errore maggiore per configurazioni in queste zone limite, o di confine, che, quindi, tende a far aumentare il valore consuntivo della metrica di valutazione. Da questa limitazione, deriva un ulteriore problema, ossia il fatto che le zone con ampiezza della band gap maggiore, potrebbero essere quelle campionate in maniera minore, determinando valori di errore maggiori nelle zone più importanti e interessanti a livello fisico. Un ulteriore elemento da sottolineare è che l'addestramento è stato effettuato su range di valori ben definiti per i parametri di input, motivo per il quale, se venisse fornita alla rete una configurazione con parametri di input che sono al di fuori di tali intervalli fissati, i risultati che si otterrebbero non sarebbero affidabili. Di conseguenza, nel momento in cui si volesse utilizzare la rete con range differenti di valori, bisognerebbe effettuare nuovamente l'addestramento. Inoltre, sono stati effettuati studi considerando valori di rumore fissati tra l'1/2%, in quanto rappresentativi delle tolleranze tipiche delle macchine manifatturiere. Qualora si avesse a disposizione una macchina con un range di rumore maggiore, bisognerebbe effettuare uno studio preliminare, destinato a verificare il mantenimento di elevati livelli di accuratezza e precisione da parte del modello, anche con misure di errore più importanti. Come detto il campo dei metamateriali è, da anni, largamente studiato e analizzato, grazie all'estrema utilità e potenza di tali materiali, parallelamente anche il machine learning è in continuo sviluppo, soprattutto negli ultimissimi anni. Riuscire, di conseguenza, a trovare un metodo per implementare e unire i due mondi risulta essere, estremamente innovativo e largamente promettente, per riuscire a mettere in gioco le migliori caratteristiche di ambe due i mondi, che, altrimenti, rimarrebbero completamente a sé stanti. Un operatore potrebbe essere in grado di utilizzare il modello di ML, come supporto per la realizzazione di strutture per l'isolamento vibrazionale, riuscendo a testare diverse configurazioni, scegliendo quella che massimizza la banda di stop e, al tempo stesso, potrebbe essere in grado di definire i parametri fisici dei metamateriali che permettono di rispondere ad alcuni requisiti di frequenza fissati direttamente da lui, riuscendo, in entrambi i casi, a ridurre il numero di simulazioni, ma soprattutto, il tempo necessario, rendendo più efficacie e proficuo il proprio lavoro. Essendo un campo, quello dei metamateriali, ma anche quello del machine learning, sottoposto a continui studi, il progetto può essere costantemente aggiornato in base alle necessità e alle nuove scoperte. Una delle prossime introduzioni, che potrebbe essere effettuata all'interno del modello è l'inserimento della rigidezza torsionale, con una massa concentrata nella zona di piega, la quale permette di ottenere una maggiore ampiezza e una maggiore flessibilità per le band gap. Inoltre, un ulteriore idea, potrebbe essere quella di implementare questi modelli di machine learning a strutture periodiche più complesse, rispetto alle semplici travi piegate, come metaframe reticolari, le quali, essendo strutture più complesse, troverebbero grandi vantaggi computazionali nell'utilizzo di architetture di machine learning, piuttosto che nell'utilizzo di modelli classici implementati su Matlab. Analisi più approfondite, poi, possono essere svolte su

range differenti di parametri di input, di numero di divisioni dei parametri, per ottenere dataset più corposi, o su valori differenti di rumore, permettendo di avere un quadro più preciso ed effettuare uno studio più approfondito su tutto lo spazio a disposizione.

BIBLIOGRAFIA e SITOGRAFIA

Atzrodt, H., Maniam, A., Droste, M., Rieß, S., & Hülsebrock, M. (2023). A power-based framework for quantifying parameter uncertainties in finite vibroacoustic metamaterial plates. Materials, 16(14), 5139. https://doi.org/10.3390/ma16145139

Bengio, Y. (2012). Practical recommendations for gradient-based training of deep architectures. In G. Montavon, G. B. Orr, & K.-R. Müller (Eds.), Neural Networks: Tricks of the Trade (2nd ed., pp. 437–478). Springer. https://doi.org/10.1007/978-3-642-35289-8_26

Braconi, L. (2019). Definizione e implementazione di attività di machine learning su dati relativi a macchine self-service dotate di stampanti e dispensatori di monete [Tesi di laurea magistrale, Università Politecnica delle Marche].

Chai, T., & Draxler, R. R. (2014). Root mean square error (RMSE) or mean absolute error (MAE)? – Arguments against avoiding RMSE in the literature. Geoscientific Model Development, 7(3), 1247–1250. https://doi.org/10.5194/gmd-7-1247-2014

Chen, C., Gu, J., & Li, J. (2022). Machine learning and deep learning in phononic crystals and metamaterials: A review. Materials Today Communications. https://doi.org/10.1016/j.mtcomm.2022.104606

Choi, R. Y., Coyner, A. S., Kalpathy-Cramer, J., Chiang, M. F., & Campbell, J. P. (2020). Introduction to machine learning, neural networks, and deep learning. Translational Vision Science & Technology, 9(2), Article 14. https://doi.org/10.1167/tvst.9.2.14

Failla, G., Marzani, A., Palermo, A., Russillo, A. F., & Colquitt, D. (2024). Current developments in elastic and acoustic metamaterials science. Philosophical Transactions of the Royal Society A, 382(2278), 20230369. https://doi.org/10.1098/rsta.2023.0369

Fawcett, T. (2006). An introduction to ROC analysis. Pattern Recognition Letters, 27(8), 861–874. https://doi.org/10.1016/j.patrec.2005.10.010

Gastaldi, G. (2023). Manifattura e caratterizzazione di metamateriali per il controllo delle vibrazioni di strutture leggere [Tesi di laurea magistrale, Politecnico di Torino].

Glorot, X., & Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh & M. Titterington (Eds.), Proceedings of the

Thirteenth International Conference on Artificial Intelligence and Statistics (Vol. 9, pp. 249–256). PMLR. https://proceedings.mlr.press/v9/glorot10a.html

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. https://www.deeplearningbook.org/

Grossi, E., & Buscema, M. (2007). Introduction to artificial neural networks. European Journal of Gastroenterology & Hepatology, 19(12), 1046–1054. https://doi.org/10.1097/MEG.0b013e3282f198a0

Gurney, K. (1997). An introduction to neural networks. UCL Press.

Haykin, S. (2009). Neural networks and learning machines (3rd ed.). Pearson Education.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: Data mining, inference, and prediction (2nd ed.). Springer.

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In F. Bach & D. Blei (Eds.), Proceedings of the 32nd International Conference on Machine Learning (Vol. 37, pp. 448–456). PMLR. https://proceedings.mlr.press/v37/ioffe15.html

Jin, Y., He, L., Wen, Z., Mortazavi, B., Guo, H., Torrent, D., Djafari-Rouhani, B., Rabczuk, T., Zhuang, X., & Li, Y. (2022). Intelligent on-demand design of phononic metamaterials. Nanophotonics, 11(3), 439–460. https://doi.org/10.1515/nanoph-2021-0639

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. 3rd International Conference on Learning Representations (ICLR 2015), Conference Track Proceedings. https://arxiv.org/abs/1412.6980

Kumar, R., Kumar, M., Chohan, J. S., & Kumar, S. (2022). Overview on metamaterial: History, types and applications. Materials Today: Proceedings, 56, 3016–3024. https://doi.org/10.1016/j.matpr.2021.11.423

Laudenzi, G. (2020). Machine learning and deep learning techniques. Università Politecnica delle Marche.

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436–444. https://doi.org/10.1038/nature14539

Liu, T.-W., Chan, C.-T., & Wu, R.-T. (2023). Deep-learning-based acoustic metamaterial design for attenuating structure-borne noise in auditory frequency bands. Materials, 16(5), 1879. https://doi.org/10.3390/ma16051879

Manconi, E., Dall'Olio, F., Riboli, M., Zippo, A., & Tatar, F. (2026). Contradirectional coupling and complete band gap formation in periodically folded beams and plates. Journal of Sound and Vibration, 636, Article 119799. https://doi.org/10.1016/j.jsv.2026.119799

Manconi, E., Hvatov, A., & Sorokin, S. V. (2023). Numerical analysis of vibration attenuation and bandgaps in radially periodic plates. Journal of Vibration Engineering & Technologies, 11(6), 2593–2603. https://doi.org/10.1007/s42417-023-00967-9

Manconi, E., Riboli, M., Contini, A., & Collini, L. (2025). Wave propagation and complete band gap formation in Miura-ori metabeams [Conference paper]. 15th International Conference on Computational Methods in Structural Dynamics and Earthquake Engineering (COMPDYN/UNCECOMP 2025).

Manconi, E., Sorokin, S. V., Garziera, R., & Quartaroli, M. M. (2021). Free and forced wave motion in a two-dimensional plate with radial periodicity. Applied Sciences, 11(22), 10948. https://doi.org/10.3390/app112210948

Mead, D. J. (1996). Wave propagation in continuous periodic structures: Research contributions from Southampton, 1964–1995. Journal of Sound and Vibration, 190(3), 495–524. https://doi.org/10.1006/jsvi.1996.0076

Mendhe, S. E., & Kosta, Y. P. (2011). Metamaterial properties and applications. International Journal of Information Technology and Knowledge Management, 4(1), 85–89.

Mienye, I. D., & Swart, T. G. (2024). A comprehensive review of deep learning: Architectures, recent advances, and applications. Information, 15(12), Article 755. https://doi.org/10.3390/info15120755

Nielsen, M. A. (2015). Neural networks and deep learning. Determination Press. http://neuralnetworksanddeeplearning.com

Nilsen, V. (2018). An introduction to machine learning [Dispensa didattica]. https://bafflerbach.github.io/test_software_carpentry/files/Module%201%20Lecture%20Notes.pdf

Ranghiasci, A. (2020). Progettazione di un metamateriale per il disaccoppiamento meccanico di vibrazioni [Tesi di laurea magistrale, Università Politecnica delle Marche].

Sarker, I. H. (2021). Machine learning: Algorithms, real-world applications and research directions. SN Computer Science, 2(3), Article 160. https://doi.org/10.1007/s42979-021-00592-x

Shaiakhmetov, R., Pianini, D., Filaseta, A., D'Angelo, G., & Venusti, V. (2026). First-order optimization algorithms: State of the art, classification, and performance: A practitioner's guide. Neural Computing and Applications, 38(7), Article 233. https://doi.org/10.1007/s00521-026-12014-1

Sharma, S., Sharma, S., & Athaiya, A. (2020). Activation functions in neural networks. International Journal of Engineering Applied Sciences and Technology, 4(12), 310–316.

Sokolova, M., & Lapalme, G. (2009). A systematic analysis of performance measures for classification tasks. Information Processing & Management, 45(4), 427–437. https://doi.org/10.1016/j.ipm.2009.03.002

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), 1929–1958.

Tian, C., Cheng, T., Peng, Z., Zuo, W., Tian, Y., Zhang, Q., Wang, F.-Y., & Zhang, D. (2025). A survey on deep learning fundamentals. Artificial Intelligence Review, 58(12), Article 381. https://doi.org/10.1007/s10462-025-11368-7

Tran, T. V., Nanthakumar, S. S., & Zhuang, X. (2025). Deep learning-based framework for the on-demand inverse design of metamaterials with arbitrary target band gap. npj Artificial Intelligence. https://doi.org/10.1038/s44387-025-00001-1

Viering, T., & Loog, M. (2023). The shape of learning curves: A review. IEEE Transactions on Pattern Analysis and Machine Intelligence, 45(6), 7799–7819. https://doi.org/10.1109/TPAMI.2022.3220744

Wang, K., Zhou, J., Tan, D., Li, Z., Lin, Q., & Xu, D. (2022). A brief review of metamaterials for opening low-frequency band gaps. Applied Mathematics and Mechanics, 43(7), 1125–1144. https://doi.org/10.1007/s10483-022-2870-9

Zilouchian, A. (2001). Fundamentals of neural networks. In A. Zilouchian & M. Jamshidi (Eds.), Intelligent control systems using soft computing methodologies (pp. 21–41). CRC Press.