**UNIVERSITÀ DEGLI STUDI DI NAPOLI "FEDERICO II"**

**DIPARTIMENTO DI SCIENZE ECONOMICHE E STATISTICHE**

CORSO DI LAUREA MAGISTRALE IN ECONOMIA E COMMERCIO

TESI DI LAUREA
IN
METODI STATISTICI PER IL DATA MINING

# KOLMOGOROV-ARNOLD NETWORK VS. MLP: ANALISI DI UN NUOVO MODELLO DI RETI NEURALI

# KOLMOGOROV-ARNOLD NETWORK VS. MLP: ANALYSIS OF A NEW NEURAL NETWORK MODEL

Relatore
Prof.ssa
Rosaria Romano

Candidato
Luca Ruocco
Matr.N28002171

Anno Accademico 2023/2024

*[...] non puoi collegare i punti guardando in avanti, puoi collegarli solamente guardando indietro; perciò devi fidarti che i punti si collegheranno in qualche modo nel tuo futuro. Devi fidarti di qualcosa, che sia il tuo istinto, il destino, la vita, il karma, qualunque cosa, perché credere che i punti si collegheranno lungo la strada ti darà la sicurezza di seguire il tuo cuore, anche quando questo ti porta fuori dal sentiero battuto. Farà tutta la differenza.*

*Steve Jobs – University of Stanford*

# INDICE

# INTRODUZIONE

SOMMARIO: Le reti KAN: un nuovo approccio nel mondo del deep learning.

# Le reti KAN: un nuovo approccio nel mondo del deep learning

Negli ultimi anni, le reti neurali artificiali hanno rivoluzionato diversi settori scientifici e tecnologici, grazie alla loro straordinaria capacità di risolvere problemi complessi che spaziano dal riconoscimento di immagini all'elaborazione del linguaggio naturale, fino alla previsione di eventi finanziari. L'ampio successo delle reti neurali è dovuto in gran parte alla loro flessibilità e alla loro capacità di apprendere modelli direttamente dai dati, spesso superando le prestazioni degli algoritmi di machine learning tradizionali.

Tra le molte architetture di reti neurali sviluppate nel corso degli anni, le Multi-Layer Perceptrons (MLP) rappresentano una delle prime e più diffuse tipologie. Le MLP sono reti *feedforward* completamente connesse, caratterizzate dalla presenza di uno o più strati nascosti di neuroni. Grazie alla loro semplicità e versatilità, le MLP sono state largamente utilizzate per risolvere numerosi problemi, sia di classificazione che di regressione. Tuttavia, nonostante la loro efficacia in molti contesti, le MLP presentano delle limitazioni intrinseche, specialmente quando vengono applicate a problemi con un'elevata complessità dimensionale o quando si richiede una rappresentazione particolarmente sofisticata dei dati. Queste limitazioni hanno spinto i ricercatori a esplorare nuove architetture e approcci per migliorare le prestazioni delle reti neurali.

Una delle direzioni più promettenti in questo ambito è rappresentata dalle Kolmogorov-Arnold Networks (KAN). Queste reti si basano sul teorema di rappresentazione universale di Kolmogorov, il quale dimostra che ogni funzione continua può essere approssimata con una combinazione di funzioni unidimensionali. Partendo da questo principio teorico, le KAN offrono una nuova modalità per rappresentare funzioni complesse e multidimensionali, con potenziali vantaggi in termini di efficienza e accuratezza rispetto alle MLP tradizionali. In

particolare, una delle principali promesse delle KAN risiede nella loro capacità di ridurre il numero di parametri necessari per modellare un sistema, preservando al contempo una rappresentazione accurata della funzione di interesse.

L'obiettivo di questa tesi è condurre un'analisi comparativa tra le Multi-Layer Perceptrons e le Kolmogorov-Arnold Networks, al fine di valutare le differenze in termini di accuratezza, efficienza computazionale e capacità di generalizzazione. Per fare ciò, entrambe le architetture sono implementate e applicate su una serie di dataset eterogenei, selezionati per coprire diverse tipologie di problemi, con vari gradi di complessità. Ogni rete è addestrata e valutata utilizzando gli stessi criteri e parametri, permettendo così un confronto rigoroso e oggettivo tra le due tecnologie.

In questo contesto, uno degli aspetti centrali dell'analisi è la capacità di ciascuna rete di adattarsi a diversi domini applicativi. Le MLP, pur essendo una tecnologia consolidata, tendono a richiedere un numero considerevole di parametri e possono soffrire di problemi legati al sovradattamento (overfitting) o alla difficoltà di generalizzare su dati nuovi, specialmente quando il modello viene addestrato su dataset di grandi dimensioni. Al contrario, le KAN, grazie alla loro struttura matematica unica, potrebbero offrire un'alternativa più efficiente, riducendo il rischio di sovradattamento e migliorando le prestazioni su compiti complessi. Inoltre, un altro elemento di interesse sarà l'analisi dell'efficienza computazionale: con la crescita esponenziale delle dimensioni dei dataset moderni, è fondamentale sviluppare modelli che possano essere addestrati e valutati in tempi ragionevoli, senza compromettere la qualità dei risultati.

La motivazione che spinge questo studio risiede quindi nella crescente necessità di identificare architetture di reti neurali che non solo siano accurate, ma che possano anche scalare in modo efficiente in presenza di dati complessi e di grandi dimensioni. L'analisi comparativa proposta in questa tesi mira a fornire una risposta

a questa domanda, esplorando i limiti e le potenzialità delle Kolmogorov-Arnold Networks rispetto alle Multi-Layer Perceptrons, con l'obiettivo di contribuire al progresso delle tecnologie di apprendimento automatico e offrire nuove prospettive per lo sviluppo di modelli più avanzati e performanti.

Il lavoro si articola in una prima parte dedicata ai fondamenti teorici, dove è presentato un inquadramento generale sui modelli supervisionati e sulle reti neurali artificiali, con un approfondimento specifico sulle MLP e le KAN. Successivamente, viene illustrata la metodologia adottata per l'esecuzione degli esperimenti, che includerà la descrizione dei dataset utilizzati, i criteri di valutazione delle prestazioni e i dettagli relativi all'implementazione delle reti. Seguono i risultati sperimentali, con una discussione approfondita delle differenze osservate tra le due architetture. Infine, vengono tratte le conclusioni, con una riflessione sui risultati ottenuti e suggerimenti per futuri sviluppi.

# CAPITOLO I

# IL DEEP LEARNING

SOMMARIO: 1. La scienza del Deep Learning– 2. La storia del Deep Learning– 2.1. Le prime idee: dagli anni '40 agli anni '60 – 2.2. Declino e rinascita: dagli anni '70 agli anni '90 – 2.3. L'avvento del deep learning: dagli anni 2000 in poi – 2.4. Verso le Multi-Layer Perceptrons (MLP) – 3. Le Sfide Attuali del Deep Learning: Interpretabilità, Dipendenza dai Dati e Sostenibilità Computazionale – 4. Tipologie di tecniche.

# 1. La scienza del Deep Learning

L'analisi dei dati consiste nello studio di matrici di informazione composte da osservazioni che registrano determinate caratteristiche numeriche, qualitative o miste. Generalmente, l'obiettivo è estrarre diverse tipologie di relazioni dai dati, utili in molti contesti della quotidianità. Ad esempio, attraverso l'analisi statistica è possibile prevedere l'andamento del tasso di insolvenza dei clienti di una banca, stimare l'ammontare delle vendite di un'azienda, oppure è possibile raggruppare individui simili tra loro per il settore del marketing, nonché valutare l'efficienza di un prodotto e così via.

Da sempre, l'estrazione di relazioni all'interno dei dati è un'operazione sfidante, sia per l'ammontare di informazione sia per la presenza di molte interazioni tra variabili, non immediatamente percepibili dall'essere umano.

Pertanto, col passare degli anni, la scienza dei dati ha definito un nuovo approccio di studio che sfrutta la capacità dei nuovi computer sviluppati di eseguire operazioni complesse e ricorsive in maniera istantanea per analizzare grandi quantità di dati. Si tratta della disciplina del Deep Learning.

Il Deep Learning rappresenta una branca avanzata del Machine Learning che si concentra sull'uso di reti neurali profonde per affrontare problemi complessi attraverso la modellazione di dati strutturati e non strutturati[1]. A differenza dei metodi tradizionali di apprendimento automatico, il Deep Learning si distingue per la capacità di apprendere autonomamente rappresentazioni complesse dei dati, senza la necessità di una significativa pre-elaborazione manuale delle *feature*. Le reti neurali profonde, che costituiscono il nucleo del Deep Learning, sono composte da più strati di neuroni artificiali, dove ciascun neurone riceve un input, lo elabora

[1] Kelleher, J. D. (2019). *Deep learning*. MIT press.

attraverso una funzione di attivazione non lineare e invia l'output ai neuroni dello strato successivo.

Questo approccio permette al modello di apprendere diverse astrazioni dai dati, il che si traduce in una capacità senza precedenti di riconoscere pattern e di fare previsioni accurate in contesti come la visione artificiale, il riconoscimento vocale e l'elaborazione del linguaggio naturale. Le reti neurali profonde funzionano tramite un processo di addestramento basato su retropropagazione dell'errore, in cui la differenza tra la previsione del modello e il risultato atteso viene propagata all'indietro attraverso la rete, permettendo all'algoritmo di aggiornare i pesi in modo iterativo fino a ridurre l'errore complessivo.

Questo processo permette alle reti di apprendere in modo progressivo, migliorando le prestazioni su problemi complessi con grandi quantità di dati. L’applicazione del Deep Learning è ormai estesa a numerosi settori: dalle auto a guida autonoma, dove consente il riconoscimento di oggetti e ostacoli in tempo reale, all’ambito medico, dove ha rivoluzionato l’analisi delle immagini diagnostiche permettendo di rilevare patologie con precisione sempre maggiore, fino all’elaborazione del linguaggio naturale, in cui modelli basati su Transformer, come GPT o BERT, hanno trasformato la traduzione automatica e l'analisi testuale.

Tuttavia, il Deep Learning non è esente da sfide: tra queste, una delle più significative è la necessità di grandi quantità di dati etichettati per l'addestramento, il che può rappresentare un limite in contesti in cui tali dati non sono disponibili. Inoltre, le reti neurali profonde, pur avendo ottime capacità predittive, possono essere considerate delle "scatole nere" a causa della difficoltà nell'interpretare le loro decisioni, rendendo complicato comprendere esattamente quali fattori influenzino i risultati finali.

Nonostante questi limiti, il Deep Learning ha dimostrato di essere uno strumento straordinariamente efficace e flessibile, soprattutto in ambienti in rapida evoluzione o in presenza di grandi dataset, e la sua capacità di adattarsi dinamicamente a nuovi pattern lo rende una tecnologia sempre più centrale nel panorama dell'intelligenza artificiale moderna.

# 2. La storia del Deep Learning

La storia del Deep Learning è strettamente legata all'evoluzione delle reti neurali artificiali e al più ampio campo dell'intelligenza artificiale (AI). Le sue radici risalgono agli anni '40, quando i primi ricercatori cominciarono a esplorare modelli computazionali ispirati al funzionamento del cervello umano, ma solo negli ultimi decenni si è trasformato in una tecnologia potente e largamente adottata.

## 2.1. Le prime idee: dagli anni '40 agli anni '60

Il concetto di reti neurali artificiali affonda le sue radici nel 1943, quando lo psicologo Warren McCulloch e il matematico Walter Pitts proposero un modello matematico semplificato dei neuroni biologici. Questo modello, noto come neurone artificiale di McCulloch-Pitts[2], fu la prima rappresentazione formale di un sistema di calcolo ispirato al cervello. L'idea era che un insieme di neuroni artificiali potesse, attraverso la combinazione di segnali in ingresso, produrre risposte binarie che riflettevano il comportamento di un singolo neurone biologico.

Nel 1958, Frank Rosenblatt sviluppò il perceptron, un tipo di rete neurale con un singolo strato, capace di apprendere dai dati attraverso un meccanismo di

[2] Wang, H., & Raj, B. (2017). On the origin of deep learning. *arXiv preprint arXiv:1702.07800*.

aggiornamento dei pesi basato sugli errori delle previsioni[3]. Il *perceptron* fu uno dei primi algoritmi di apprendimento automatico a guadagnare popolarità, poiché dimostrava come una macchina potesse imparare a classificare input in categorie diverse. Tuttavia, il perceptron presentava una limitazione significativa: era in grado di risolvere solo problemi linearmente separabili, il che ne ridusse notevolmente l'efficacia.

## 2.2. Declino e rinascita: dagli anni '70 agli anni '90

Negli anni '70, il campo delle reti neurali artificiali subì una battuta d'arresto a causa dei limiti teorici e pratici delle reti a strato singolo. Nel 1969, Marvin Minsky e Seymour Papert, due importanti figure nell'ambito dell'intelligenza artificiale, pubblicarono un libro intitolato Perceptrons, in cui evidenziarono le carenze dei perceptron, in particolare la loro incapacità di risolvere problemi non lineari come l'esclusivo OR (XOR). Questa critica bloccò la ricerca sulle reti neurali per circa un decennio, e l'attenzione della comunità scientifica si spostò verso altri approcci di intelligenza artificiale basati su regole simboliche.

La rinascita delle reti neurali iniziò negli anni '80, grazie a due sviluppi fondamentali. Il primo fu l'introduzione dell'algoritmo di retropropagazione, reso popolare da Geoffrey Hinton, David Rumelhart e Ronald Williams.

La retropropagazione è un metodo che permette di addestrare reti neurali a più strati (le cosiddette reti neurali profonde) regolando i pesi dei neuroni in modo da minimizzare l'errore complessivo. Questo algoritmo consentì alle reti neurali di affrontare problemi più complessi rispetto al perceptron a strato singolo[4].

---

[3] Rosenblatt, F. (2021). The Perceptron: A Probabilistic Model for Information Storage and Organization (1958).

[4] Cilimkovic, M. (2015). Neural networks and back propagation algorithm. *Institute of Technology Blanchardstown, Blanchardstown Road North Dublin*, *15*(1).

Il secondo sviluppo chiave fu il progresso delle capacità computazionali e l'aumento della potenza dei calcolatori. Questi progressi resero possibile l'addestramento di reti neurali più grandi e profonde, benché i limiti tecnologici dell'epoca fossero ancora un ostacolo significativo.

## 2.3. L'avvento del deep learning: dagli anni 2000 in poi

Negli anni 2000, il campo delle reti neurali profonde cominciò a prendere slancio grazie a una serie di scoperte chiave. Yann LeCun, un pioniere del Deep Learning, sviluppò le reti neurali convoluzionali (CNN) per il riconoscimento di immagini. Le CNN si dimostrarono particolarmente efficaci nella visione artificiale, utilizzando tecniche come la convoluzione e il pooling per estrarre automaticamente feature dai dati visivi, rendendo superflua la progettazione manuale delle caratteristiche, che era invece richiesta negli approcci di machine learning tradizionali.

Un momento decisivo per il Deep Learning avvenne nel 2006, quando Geoffrey Hinton e i suoi colleghi dimostrarono che era possibile addestrare reti neurali profonde usando tecniche di pre-addestramento non supervisionato. Questo approccio consentiva di superare alcuni dei principali problemi che affliggevano le reti neurali profonde, come la dissipazione del gradiente (un problema che rendeva difficile l'addestramento efficace dei modelli più profondi)[5]. Questo risultato riaccese l'interesse della comunità scientifica verso il Deep Learning, spianando la strada a una rapida crescita del campo.

Nel corso degli anni 2010, la crescita esponenziale della capacità di calcolo (soprattutto con l'avvento delle GPU, utilizzate per accelerare il processo di addestramento delle reti neurali), insieme alla disponibilità di grandi quantità di

[5] Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). Deep Machine Learning-A New Frontier in Artificial Intelligence Research. *A fast learning algorithm for deep belief nets. Neural Computation*, *18*(7), 1527-1554.

dati (big data), ha permesso al Deep Learning di raggiungere risultati eccezionali. In particolare, il successo di AlexNet nel 2012, un'architettura di rete convoluzionale addestrata su grandi dataset, portò il Deep Learning alla ribalta internazionale. AlexNet dominò la competizione ImageNet, un prestigioso concorso di visione artificiale, ottenendo un miglioramento senza precedenti nelle prestazioni.

A partire dagli anni 2010, il Deep Learning ha iniziato a essere applicato in una vasta gamma di settori, portando a innovazioni dirompenti. Nella visione artificiale, le reti neurali convoluzionali hanno superato le prestazioni umane nel riconoscimento di immagini e oggetti. Nell'elaborazione del linguaggio naturale, tecniche basate su reti neurali ricorrenti (RNN) e più recentemente su architetture Transformer, come BERT e GPT, hanno rivoluzionato la traduzione automatica, il completamento del testo e la generazione di linguaggio naturale. Anche nelle auto a guida autonoma, il Deep Learning è al centro dello sviluppo dei sistemi di percezione, contribuendo all'identificazione di ostacoli e alla navigazione autonoma.

In campo medico, le reti neurali profonde hanno dimostrato di poter diagnosticare malattie analizzando immagini mediche con una precisione comparabile, e talvolta superiore, a quella degli specialisti umani. Allo stesso modo, in ambiti come la finanza e il marketing, il Deep Learning viene utilizzato per analizzare enormi quantità di dati, rilevare frodi, personalizzare esperienze degli utenti e fare previsioni precise su mercati o comportamenti dei consumatori.

## 2.4. Verso le Multi-Layer Perceptrons (MLP)

Mentre le architetture più complesse, come le CNN, hanno ottenuto risultati impressionanti, è importante riconoscere il ruolo fondamentale delle Multi-Layer Perceptrons (MLP) nello sviluppo delle reti neurali moderne. Le MLP

rappresentano un'evoluzione naturale del perceptron a strato singolo, aggiungendo uno o più strati nascosti tra l'input e l'output. Questi strati nascosti permettono alla rete di apprendere relazioni non lineari e di elaborare in modo più efficace informazioni complesse.

Le MLP sono reti *feedforward* completamente connesse, in cui ogni neurone di uno strato è collegato a tutti i neuroni dello strato successivo[6]. Durante l'addestramento, l'algoritmo di retropropagazione viene utilizzato per aggiornare i pesi di queste connessioni in modo da ridurre l'errore globale. Questa architettura, pur essendo più semplice rispetto alle CNN o ad altri modelli più avanzati, è stata cruciale nel gettare le basi per il deep learning moderno, dimostrando come le reti neurali possano apprendere pattern complessi da grandi quantità di dati.

Le MLP sono state utilizzate con successo in numerosi contesti, dalla classificazione di immagini e testo alla previsione di serie temporali, dimostrando la loro flessibilità e adattabilità a una vasta gamma di problemi. Nonostante l'emergere di architetture più avanzate, le MLP continuano a essere un pilastro nel campo del deep learning, e rappresentano uno dei primi modelli in grado di affrontare efficacemente i problemi di apprendimento supervisionato.

[6] Fine, T. L. (2006). *Feedforward neural network methodology*. Springer Science & Business Media.

# 3. Le Sfide Attuali del Deep Learning: Interpretabilità, Dipendenza dai Dati e Sostenibilità Computazionale

Il Deep Learning, pur rappresentando una delle tecnologie più rivoluzionarie degli ultimi anni, presenta ancora una serie di sfide e limitazioni che devono essere affrontate per garantirne un utilizzo più efficace e sostenibile.

Tra i problemi più significativi c'è la dipendenza dai dati etichettati. I modelli di deep learning richiedono enormi quantità di dati etichettati per poter essere addestrati con un buon grado di precisione[7]. La raccolta e l'etichettatura manuale di questi dati può essere un processo estremamente dispendioso in termini di tempo e risorse, specialmente in settori come la medicina o la ricerca scientifica, dove ottenere grandi set di dati di qualità è complicato.

Questo problema ha portato all'esplorazione di metodi alternativi, come l'apprendimento auto-supervisionato e l'apprendimento semi-supervisionato, che mirano a ridurre la dipendenza da dati etichettati utilizzando in modo più efficace i dati non etichettati.

Un altro aspetto problematico è legato alla complessità e opacità delle reti neurali profonde. Questi modelli, spesso descritti come "black box", sono difficili da interpretare, e in molti contesti l'interpretabilità è cruciale.

Ad esempio, nel campo della medicina, un sistema di intelligenza artificiale che suggerisce una diagnosi o un trattamento deve essere in grado di spiegare chiaramente il motivo per cui ha preso una determinata decisione. Una mancanza di trasparenza può sollevare questioni etiche e di responsabilità, questa mancanza di trasparenza nei modelli di deep learning ha portato a una crescente necessità di sviluppare sistemi più interpretabili, ovvero modelli che non solo possano fare

[7] https://www.ibm.com/it-it/topics/neural-networks

previsioni accurate, ma che siano anche in grado di spiegare il processo decisionale che li guida. In contesti come la medicina o la finanza, dove le decisioni hanno un impatto diretto sulla vita delle persone, è essenziale che i modelli possano fornire spiegazioni chiare e comprensibili su come un certo risultato sia stato raggiunto. La trasparenza dei modelli, quindi, diventa cruciale per garantire fiducia e sicurezza nell'uso dell'intelligenza artificiale in questi settori.

In risposta a questa esigenza, sono stati sviluppati nuovi approcci che mirano a migliorare l'interpretabilità dei modelli, pur mantenendo la loro capacità di apprendere dai dati complessi. Tra questi, le Kolmogorov-Arnold Networks (KAN) si distinguono per offrire una struttura più semplice e interpretabile rispetto alle reti neurali profonde tradizionali. Senza entrare nei dettagli tecnici, le KAN propongono un modo diverso di rappresentare i dati, che permette di seguire più facilmente il processo che porta alle decisioni finali del modello.

Questi sforzi per creare modelli più interpretabili sono essenziali in un contesto in cui l'intelligenza artificiale è sempre più utilizzata in settori critici, dove la fiducia nei sistemi automatizzati è fondamentale. La capacità di spiegare come i dati in ingresso influenzano l'output finale non solo migliora la fiducia negli strumenti basati su reti neurali, ma garantisce anche una maggiore trasparenza e responsabilità nelle applicazioni in cui sono coinvolti.

In aggiunta, con l’aumento della profondità e della complessità dei modelli di deep learning, emergono questioni importanti legate al costo computazionale e all’efficienza energetica. Anche le Multi-Layer Perceptrons (MLP), nonostante la loro struttura relativamente lineare, richiedono una notevole potenza di calcolo e risorse computazionali considerevoli, soprattutto quando si lavora con reti più ampie o su dataset di grandi dimensioni. Ogni strato aggiuntivo nella rete comporta un maggiore carico computazionale, rendendo l'addestramento delle MLP un processo che può essere dispendioso in termini di tempo e di energia. Questo

aspetto limita l'accesso a tali tecnologie alle sole organizzazioni o istituzioni che dispongono delle risorse necessarie per gestire tali modelli su larga scala. Oltre ai limiti pratici, l'utilizzo intensivo di risorse computazionali solleva anche preoccupazioni sull'impatto ambientale, dal momento che l'elevata richiesta di energia pone nuove sfide in termini di sostenibilità.

Gli hardware specializzati come le GPU (Graphics Processing Units) e le TPU (Tensor Processing Units) sono progettati per accelerare i processi di calcolo, ma il loro utilizzo su larga scala richiede un elevato consumo di energia. La questione della sostenibilità sta diventando sempre più centrale, spingendo i ricercatori a sviluppare soluzioni più efficienti.

Nonostante queste sfide, il campo del deep learning è in continua evoluzione, e la ricerca si sta concentrando su diversi fronti per superare tali limitazioni. Da un lato, si sta cercando di rendere i modelli più efficienti. Dall'altro, si sta lavorando sulla generalizzazione dei modelli, affinché siano in grado di operare con meno dati etichettati e di adattarsi meglio a nuovi contesti senza dover essere riaddestrati da zero. Uno dei problemi principali nei modelli di deep learning tradizionali è quello dell'oblio catastrofico, un fenomeno per cui, quando un modello apprende nuove informazioni, tende a "dimenticare" le conoscenze precedentemente acquisite. Questo comporta spesso la necessità di riaddestrare il modello interamente da zero ogni volta che vengono aggiunti nuovi dati o compiti, aumentando il carico computazionale e riducendo l'efficienza del processo di apprendimento.

Le Kolmogorov-Arnold Networks (KAN) si propongono come una soluzione promettente per superare questo limite. Grazie alla loro struttura basata su principi matematici solidi, le KAN offrono un approccio che permette di preservare meglio le informazioni apprese in precedenza, riducendo il problema dell'oblio catastrofico. Ciò significa che queste reti possono adattarsi a nuovi contesti o compiti senza dover essere riaddestrate da zero, mantenendo la conoscenza

acquisita e migliorando significativamente la loro capacità di generalizzazione[8]. Questo aspetto non solo migliora l'efficienza dei modelli, ma riduce anche la necessità di enormi quantità di dati etichettati per ogni nuovo compito, aprendo la strada a un utilizzo più versatile e meno dispendioso delle risorse computazionali.

In conclusione, il Deep Learning, pur avendo ottenuto successi straordinari in numerosi campi, continua a presentare sfide significative che devono essere risolte per garantire il suo sviluppo futuro.

# 4. Tipologie di tecniche

Il processo di apprendimento automatico consiste in un'ottimizzazione iterativa, cioè una serie di comandi indirizzati alla macchina che sono ripetuti fino al raggiungimento di un risultato *good enough*. Queste tecniche sono suddivise in due grandi categorie: metodi di apprendimento supervisionato e non supervisionato.

I primi sono caratterizzati da un approccio confermativo dell'analisi con l'obiettivo di effettuare una previsione, cioè ipotizzano l'esistenza di una relazione di dipendenza della variabile di risposta nei confronti di una serie di predittori, in modo da ottenere degli output confrontabili con la conoscenza che si ha a priori delle osservazioni, definendo la performance del modello.

Pertanto, si sfrutta l'informazione già posseduta del fenomeno per generare uno strumento in grado di captare la relazione di associazione asimmetrica nei dati, con lo scopo di generalizzare delle previsioni a nuove unità statistiche.

[8] Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Soljačić, M., ... & Tegmark, M. (2024). Kan: Kolmogorov-arnold networks. *arXiv preprint arXiv:2404.19756*.

Le tecniche di apprendimento supervisionato si suddividono in due categorie principali: regressione e classificazione. Nella regressione, l'obiettivo è quello di prevedere un valore numerico continuo e vengono addestrati su un insieme di dati di training con etichette numeriche, con lo scopo di individuare la relazione tra le variabili indipendenti e quella dipendente. Nella classificazione, invece, l'obiettivo è quello di assegnare un'etichetta discreta a un'istanza e vengono addestrati su un insieme di unità statistiche classificate a priori.

Nei metodi non supervisionati, invece, l'approccio è di tipo esplorativo. L'obiettivo è scoprire dei pattern nei dati di cui non si ha alcuna conoscenza, e sono prevalentemente caratterizzati da tecniche di riduzione della dimensionalità, di associazione o di *clustering*, in cui tutte le variabili assumono un medesimo ruolo.

È importante notare che classificazione e clustering sono termini tecnici simili ma non sovrapponibili di significato. Entrambi gli approcci raggruppano gli individui, solo che la classificazione lo fa in termini della variabile target per approssimare una relazione di dipendenza tra la stessa e una serie di predittori, mentre il clustering compone dei gruppi sulla base delle caratteristiche che gli individui presentano, minimizzando l'eterogeneità al loro interno[9].

Inoltre, esistono anche i modelli semi-supervisionati, in cui si combinano osservazioni già etichettate con quelle di cui non si ha alcuna informazione, e possono presentare anche un incremento significativo dell'accuratezza dell'apprendimento.

Questa tesi, in particolare, porrà attenzione ai metodi supervisionati.

[9] Rajoub, B. (2020). Supervised and unsupervised learning. In *Biomedical signal processing and artificial intelligence in healthcare* (pp. 51-89). Academic Press.

# CAPITOLO II

# LE RETI NEURALI ARTIFICIALI: FUNZIONAMENTO E STRUTTURA

SOMMARIO: Introduzione – 1. I fondamenti delle reti neurali artificiali – 1.1. Ispirazione biologica – 1.1.1. Neuroni biologici – 1.1.2. Funzionamento del neurone biologico – 2. Modello del Neurone Artificiale – 2.1. Componenti del neurone artificiale – 2.2. Funzionamento del neurone artificiale – 3. Funzioni di attivazione delle reti neurali – 3.1. Il ruolo delle funzioni di attivazione – 3.2. Il funzionamento delle funzioni di attivazione – 3.3. Principali funzioni di attivazione utilizzate nelle reti neurali – 4. Struttura delle reti neurali: tipologie e architettura – 4.1. Tipologie di reti neurali – 4.2. Evoluzione dei modelli di rete – 4.3. Tecniche per la progettazione delle reti neurali – 5. Addestramento delle reti neurali artificiali – 6. Metriche di valutazione della performance – 7. Reti Multi-Layer Perceptron (MLP) – 7.1. Il Teorema di Approssimazione Universale – 7.2. Vantaggi e svantaggi delle MLP.

# Introduzione

Questo capitolo presenterà il mondo delle reti neurali artificiali, con un particolare focus sulle Multi-Layer Perceptrons (MLP), una delle architetture più fondamentali e diffuse nel campo del deep learning. Le reti neurali, ispirate alla struttura del cervello umano, rappresentano oggi uno strumento potente per risolvere problemi complessi, come la classificazione, la regressione e la previsione di dati. Si inizierà con una panoramica generale sulle reti neurali, descrivendo i principi di base che ne regolano il funzionamento, dal processo di apprendimento alla struttura a strati.

Successivamente, verrà approfondita l'architettura delle MLP, descrivendo in dettaglio come queste reti siano composte da uno o più strati nascosti che permettono di apprendere modelli sempre più complessi. Si presenterà il processo di propagazione in avanti e il modo in cui consente alla rete di elaborare i dati e come, invece, l'algoritmo di retropropagazione viene impiegato per ottimizzare i pesi interni della rete durante l'addestramento. L'analisi si concentrerà sulle basi matematiche che governano queste operazioni, con particolare attenzione alle funzioni di attivazione e alla loro importanza nel conferire al modello la capacità di apprendere relazioni non lineari.

Infine, si esamineranno anche alcune tecniche avanzate che possono essere applicate per migliorare le prestazioni delle MLP, ottimizzando l'accuratezza del modello e la sua capacità di generalizzare su dati non visti. Verranno discusse le principali sfide che si incontrano nell'addestramento di queste reti, come la dissipazione del gradiente e l'*overfitting*, e verranno presentate alcune soluzioni per affrontarle, come la regolarizzazione e l'uso di funzioni di attivazione avanzate. Questo capitolo fornirà dunque una comprensione completa delle MLP, illustrandone l'importanza e il ruolo nel contesto del deep learning e delle reti neurali artificiali.

# 1. I fondamenti delle reti neurali artificiali

Le reti neurali artificiali sono una delle tecnologie più affascinanti e potenti nel campo dell'intelligenza artificiale e del machine learning. Esse traggono ispirazione diretta dal funzionamento del cervello umano, simulando, attraverso modelli matematici, il comportamento dei neuroni biologici che formano la base del sistema nervoso. Prima di entrare nel dettaglio del modello artificiale, è utile comprendere come i neuroni biologici operano e come queste operazioni siano state astratte nei modelli computazionali.

## 1.1. Ispirazione biologica

Il cervello umano è un organo altamente complesso, costituito da circa 86 miliardi di neuroni, ognuno dei quali può formare migliaia di connessioni con altri neuroni, generando una vasta rete di interazioni. Questo sistema nervoso elabora informazioni tramite impulsi elettrici e chimici che viaggiano lungo le connessioni tra neuroni, permettendo al cervello di svolgere una gamma infinita di funzioni cognitive, motorie e sensoriali.

### 1.1.1. Neuroni biologici

Il neurone biologico è l'unità fondamentale del sistema nervoso. È responsabile della ricezione e della trasmissione dei segnali elettrici e chimici che permettono il funzionamento del cervello. Ogni neurone è composto da diverse parti che lavorano insieme per elaborare le informazioni:

- Dendriti: Sono le ramificazioni del neurone che ricevono i segnali da altri neuroni. Questi segnali possono essere eccitatori o inibitori e rappresentano i messaggi inviati dai neuroni vicini.
- Soma (Corpo Cellulare): Il corpo cellulare è il nucleo del neurone. Qui avviene l'elaborazione centrale delle informazioni ricevute dai dendriti. Il

soma integra tutti i segnali in ingresso e decide se generarne uno in uscita, basandosi sul loro valore complessivo.

- Assone: Se i segnali ricevuti dai dendriti e integrati nel soma superano una certa soglia, viene generato un potenziale d'azione, che si propaga lungo l'assone. L'assone è una lunga estensione del neurone che trasporta il segnale fino ai neuroni successivi.
- Sinapsi: Le sinapsi sono le giunzioni attraverso cui un neurone comunica con un altro. Il segnale elettrico che viaggia lungo l'assone viene trasformato in un segnale chimico (tramite i neurotrasmettitori) alla sinapsi, permettendo al messaggio di attraversare lo spazio tra neuroni e di continuare il suo percorso.

### 1.1.2. Funzionamento del neurone biologico

Il processo di elaborazione in un neurone biologico segue una sequenza precisa. I segnali elettrici vengono ricevuti attraverso i dendriti e trasmessi al soma. Qui, il neurone integra i segnali ricevuti, sommando le eccitazioni e inibizioni. Se la somma totale supera una certa soglia di attivazione, il neurone genera un potenziale d'azione, che si propaga lungo l'assone e raggiunge la sinapsi, dove stimola altri neuroni.

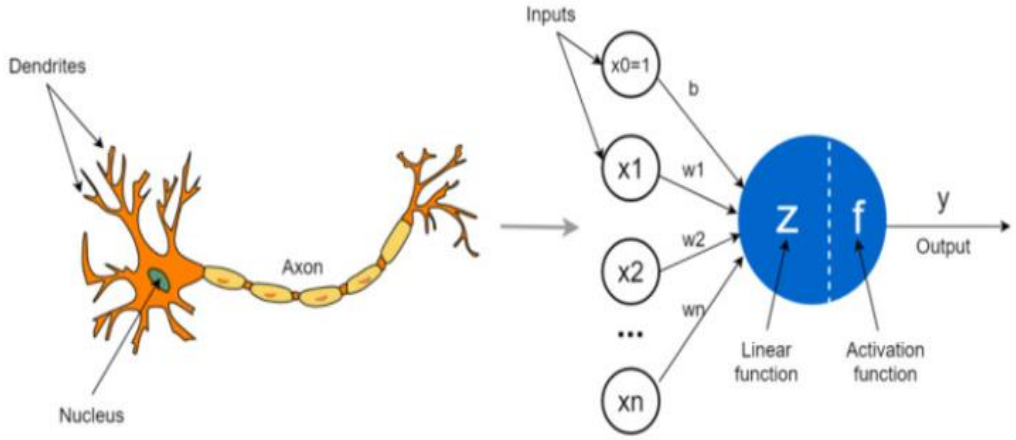



**Figura 2.1: Parallelismo tra la biologia e il deep learning nel mondo delle reti neurali artificiali.**

Il sistema nervoso è una rete complessa di milioni di neuroni che comunicano continuamente tra loro. È proprio questa capacità di elaborare e trasmettere segnali in parallelo che ha ispirato la creazione delle reti neurali artificiali, come rappresentato in Figura (2.1), che cercano di replicare l'efficacia e la potenza computazionale del cervello biologico nel risolvere problemi complessi.

# 2. Modello del Neurone Artificiale

L'idea di simulare il comportamento dei neuroni biologici in un contesto computazionale portò, alla fine degli anni '50, allo sviluppo del neurone artificiale. Il primo modello formale di neurone artificiale venne proposto da Frank Rosenblatt nel 1958, sotto forma di Perceptron. Questo modello costituisce la base di molte delle architetture di reti neurali moderne.

Come rappresentato in Figura (2.2), il neurone artificiale è un'astrazione matematica che imita il funzionamento del neurone biologico, elaborando segnali in ingresso (input), pesandoli in base alla loro importanza, e producendo un'uscita (output) che dipende da una funzione di attivazione[10].

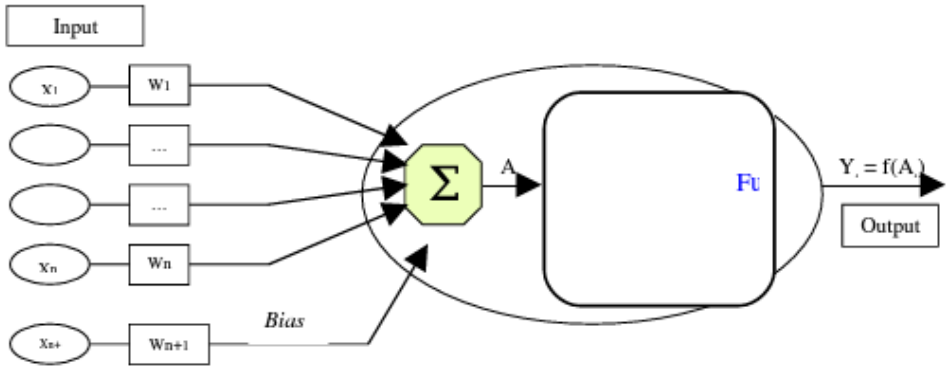



**Figura 2.2: Il neurone artificiale come astrazione matematica.**

## 2.1. Componenti del neurone artificiale

Il neurone artificiale è composto da una serie di elementi che replicano il comportamento dei neuroni biologici:

[10] Krenker, A., Bešter, J., & Kos, A. (2011). Introduction to the artificial neural networks. *Artificial Neural Networks: Methodological Advances and Biomedical Applications. InTech*, 1-18.

- Input ($x_i$): Gli input rappresentano i segnali in ingresso al neurone artificiale, che possono provenire dall'ambiente esterno o da altri neuroni all'interno della rete. Ogni input rappresenta una variabile del problema che si sta cercando di risolvere, ad esempio i pixel di un'immagine o le caratteristiche di un dataset numerico.

- Pesi Sinaptici ($w_i$): A ciascun input è associato un peso sinaptico, che rappresenta l'importanza relativa di quell'input per il neurone. I pesi sono numeri reali che vengono ottimizzati durante l'addestramento della rete neurale, e determinano l'influenza che ciascun input ha sull'output finale. Più alto è il peso, maggiore sarà l'influenza di quell'input sulla decisione del neurone.

- Bias ($b$): Il bias è un termine costante aggiunto alla combinazione lineare degli input pesati. Esso permette di regolare il livello di attivazione del neurone, e viene utilizzato per rendere il modello più flessibile, spostando la soglia di attivazione del neurone e garantendo che la funzione di attivazione possa operare in maniera più efficace.

- Potenziale Netto ($z$): Il potenziale netto è il risultato della combinazione lineare degli input pesati e del bias. Matematicamente, il potenziale netto è espresso come[11]:

$$z = \sum_{i=1}^{n} w_i x_i + b$$

[11] Wu, Y. C., & Feng, J. W. (2018). Development and application of artificial neural network. *Wireless Personal Communications*, *102*, 1645-1656.

Qui, ogni input $x_i$ viene moltiplicato per il rispettivo peso $w_i$, e la somma di questi prodotti è aggiunta al bias $b$. Il risultato è il potenziale netto che verrà elaborato dalla funzione di attivazione.

- Funzione di Attivazione ($\phi(z)$): La funzione di attivazione è un componente essenziale del neurone artificiale. Essa trasforma il potenziale netto in un output non lineare. A differenza di un semplice modello lineare, la funzione di attivazione introduce la non linearità, permettendo alla rete di apprendere relazioni complesse tra gli input. Le funzioni di attivazione più comuni includono la sigmoide, la tangente iperbolica e la ReLU (Rectified Linear Unit). Ogni funzione di attivazione ha proprietà diverse e viene utilizzata in base alla natura del problema.
- Output ($y$): Infine, il neurone produce un output basato sul risultato della funzione di attivazione applicata al potenziale netto. Questo output può essere il segnale finale della rete (nel caso dell'output layer) o può essere trasmesso ai neuroni successivi negli strati nascosti.

$$y = \phi(z)$$

## 2.2. Funzionamento del neurone artificiale

Il funzionamento di un neurone artificiale segue un processo semplice ma efficace. Ogni input ricevuto dal neurone viene ponderato attraverso i rispettivi pesi sinaptici. Questi valori vengono poi sommati tra loro insieme al bias, ottenendo il potenziale netto. A questo punto, il potenziale netto è elaborato da una funzione di attivazione che determina se il neurone debba "attivarsi" o meno, producendo così l'output finale. Se il neurone si trova in uno strato intermedio (strato nascosto), l'output viene trasmesso ai neuroni successivi. Se invece il neurone fa parte dello

strato di output, il valore generato rappresenta il risultato finale del modello, come una previsione o una classificazione.

## 3. Funzioni di attivazione delle reti neurali

Le funzioni di attivazione costituiscono un elemento cruciale nel funzionamento delle reti neurali artificiali, poiché conferiscono al modello la capacità di apprendere e rappresentare relazioni non lineari presenti nei dati. Senza l'integrazione di queste funzioni, una rete neurale si ridurrebbe a una combinazione lineare degli input, anche in presenza di numerosi strati e neuroni. Ciò limiterebbe drasticamente la sua efficacia nel risolvere problemi complessi che, nel mondo reale, sono tipicamente di natura non lineare[12].

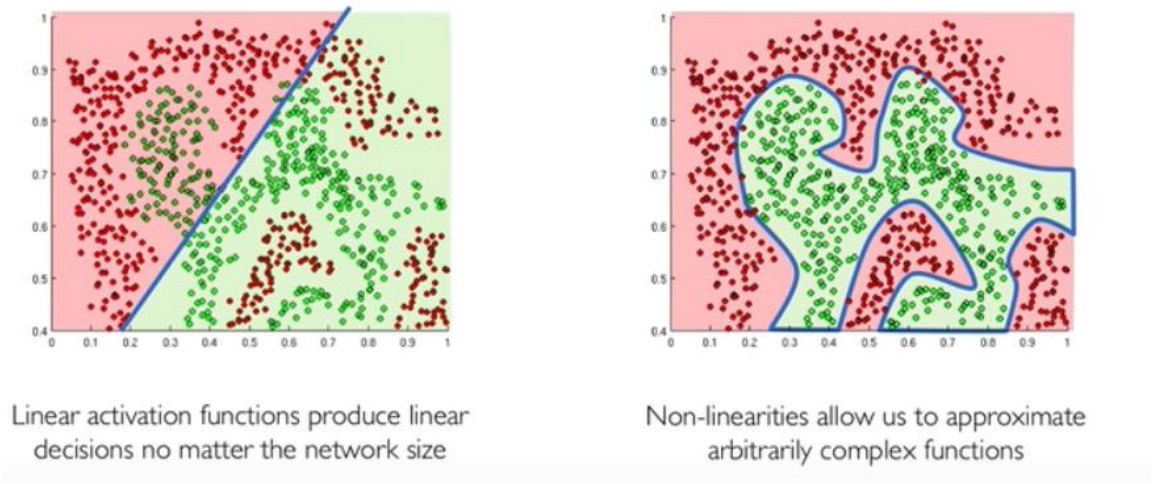



**Figura (2.3): Importanza della non linearità.**

[12] Sharma, S., Sharma, S., & Athaiya, A. (2017). Activation functions in neural networks. *Towards Data Sci*, *6*(12), 310-316.

## 3.1. Il ruolo delle funzioni di attivazione

In termini tecnici, una funzione di attivazione è una funzione matematica applicata all'output di un neurone, dopo che questo ha elaborato gli input ricevuti. Ogni neurone, all'interno della rete, riceve una combinazione lineare degli input — ovvero, la somma degli input moltiplicati per i rispettivi pesi, cui si aggiunge un termine di bias — e applica una funzione di attivazione per determinare l'output finale. Questo output, che può essere di natura non lineare rispetto agli input originali, viene poi trasmesso ai neuroni dello strato successivo o costituisce l'output finale della rete neurale.

Le funzioni di attivazione rivestono un ruolo fondamentale nel consentire alle reti neurali di modellare relazioni complesse e risolvere problemi reali. La loro necessità deriva da tre principali fattori:

1. Introduzione della Non Linearità: Senza l'applicazione di funzioni di attivazione non lineari, l'intera rete neurale, indipendentemente dalla sua profondità o complessità, si ridurrebbe a una trasformazione lineare degli input. Ciò limiterebbe significativamente le capacità del modello, poiché molti fenomeni nel mondo reale seguono relazioni non lineari (Figura 2.3). Le funzioni di attivazione, quindi, consentono alla rete di apprendere e modellare queste relazioni complesse.

2. Capacità di Apprendimento: Senza non linearità, una rete neurale sarebbe equivalente a un singolo neurone, poiché la composizione di funzioni lineari produce ancora una funzione lineare. È solo grazie all'introduzione di funzioni non lineari che la rete è in grado di sfruttare a pieno la sua architettura multi-strato, combinando le informazioni in modi complessi e articolati, con conseguenti miglioramenti nelle capacità di apprendimento.

3. Apprendimento tramite Retropropagazione: La maggior parte delle funzioni di attivazione utilizzate è differenziabile, una proprietà fondamentale per l'applicazione di algoritmi di addestramento come la retropropagazione dell'errore. La differenziabilità delle funzioni di attivazione consente di calcolare i gradienti, che sono essenziali per aggiornare i pesi della rete e migliorare le prestazioni del modello durante l'apprendimento.

## 3.2. Il funzionamento delle funzioni di attivazione

Il processo che coinvolge l'applicazione delle funzioni di attivazione all'interno di una rete neurale può essere descritto attraverso i seguenti passaggi:

1. Calcolo dell'Input Netto: Ogni neurone riceve un insieme di input dai neuroni dello strato precedente. Questi input vengono ponderati in base ai rispettivi pesi e sommati, generando una combinazione lineare nota come potenziale netto. A questo valore viene spesso aggiunto un termine di bias.

2. Applicazione della Funzione di Attivazione: Il potenziale netto viene quindi passato attraverso una funzione di attivazione, che trasforma questo valore in un output non lineare. Questa trasformazione introduce la non linearità necessaria per permettere alla rete di apprendere relazioni complesse[13].

3. Propagazione dell'Output: L'output del neurone, una volta trasformato dalla funzione di attivazione, viene trasmesso ai neuroni dello strato successivo o utilizzato come output finale della rete neurale, a seconda della posizione del neurone all'interno della rete.

[13] Parhi, R., & Nowak, R. D. (2020). The role of neural network activation functions. *IEEE Signal Processing Letters*, *27*, 1779-1783.

## 3.3. Principali funzioni di attivazione utilizzate nelle reti neurali[14]

Nel corso degli anni, diverse funzioni di attivazione sono state proposte e adottate, ciascuna con caratteristiche specifiche che le rendono più o meno adatte a particolari tipi di reti neurali o problemi.

Nel 1943 McCulloch e Pitts proposero il primo modello matematico riferito ad un neurone, rappresentato da una funzione binaria, con ingresso ed uscita binari, ed una fissata soglia di attivazione che veniva mantenuta fuori dal calcolo del potenziale di attivazione.

La risposta del neurone era data da una funzione "a gradino",

$$F(A) = \begin{cases} 1 & seA > \vartheta \\ 0 & seA \leq \vartheta \end{cases}$$

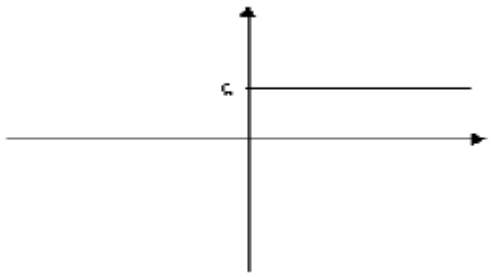


**Figura 2.4: Funzione a gradino.**

La funzione a gradino, presentata in Figura (2.4), può anche presentare un output bipolare, con F(A)= -1 per $A \leq \vartheta$ e $F(A) = 1$ per $A > \vartheta$. In entrambi i casi il neurone può assumere solo due stati, attivarsi o non attivarsi, a seconda del livello di soglia $\vartheta$ .

[14] Mercioni, M. A., & Holban, S. (2020, May). The most used activation functions: Classic versus current. In *2020 International Conference on Development and Application Systems (DAS)* (pp. 141-145). IEEE.

È possibile avere maggiori informazioni con l'utilizzo di una funzione *lineare continua*, come in Figura (2.5), che permette al neurone di trasmettere risposte di diversa intensità a seconda della gradazione di segnale ricevuto.

$F(A) = kA$ , dove $k$ è una costante

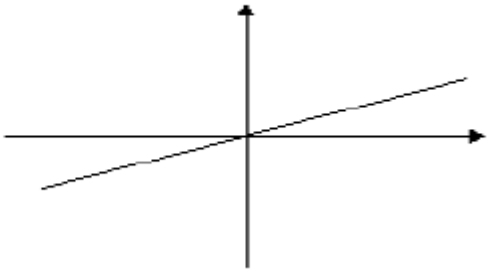


**Figura 2.5: Funzione lineare continua.**

Esistono infine una famiglia di funzioni continue non lineari, frequentemente utilizzate:

1. Funzione Sigmoide, rappresentata in Figura (2.6), è quella che approssima in modo migliore il funzionamento dei neuroni reali. Tale funzione tende asintoticamente ai livelli di saturazione rappresentati dai suoi valori di massimo e di minimo ed è descritta matematicamente dalla formula:

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

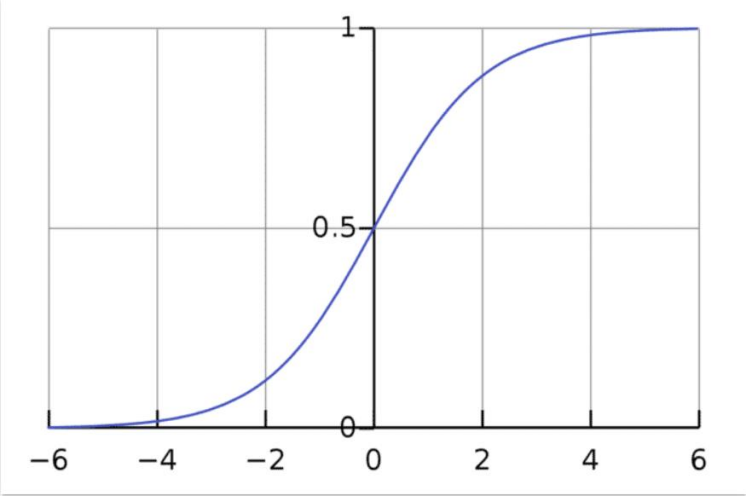



**Figura 2.6: Funzione Sigmoide.**

- Caratteristiche:
    - Mappa qualsiasi valore reale in un intervallo compreso tra 0 e 1.
    - È stata ampiamente utilizzata nelle prime reti neurali, soprattutto per problemi di classificazione binaria.
    - Un problema significativo associato alla funzione sigmoide è il vanishing gradient: per valori di input estremi, il gradiente diventa molto piccolo, rallentando drasticamente l'addestramento delle reti profonde.

2. Tangente Iperbolica (tanh), in Figura (2.7), che è definita come:

$$f(x) = \tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

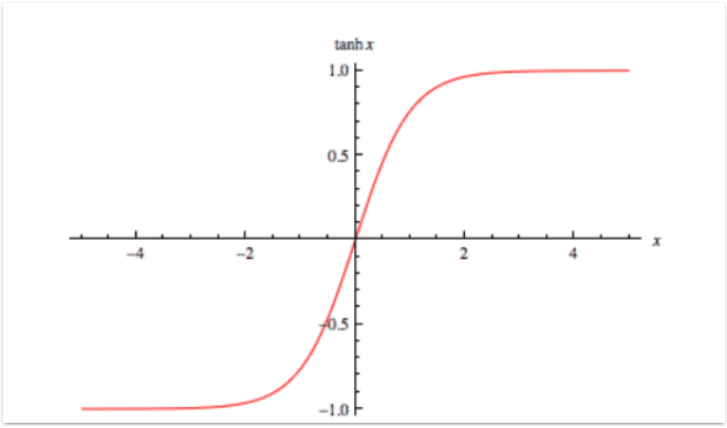



**Figura 2.7: Funzione Tangente Iperbolica.**

- Caratteristiche:
    - Mappa i valori reali in un intervallo compreso tra -1 e 1, rendendola una funzione zero-centrata.
    - Essendo centrata su zero, può migliorare la convergenza durante l'addestramento.
    - Tuttavia, soffre anch'essa del problema del vanishing gradient, in modo simile alla sigmoide.

3. ReLU (Rectified Linear Unit), rappresentata in Figura (2.8), è definita come:

$$f(x) = \max(0, x)$$

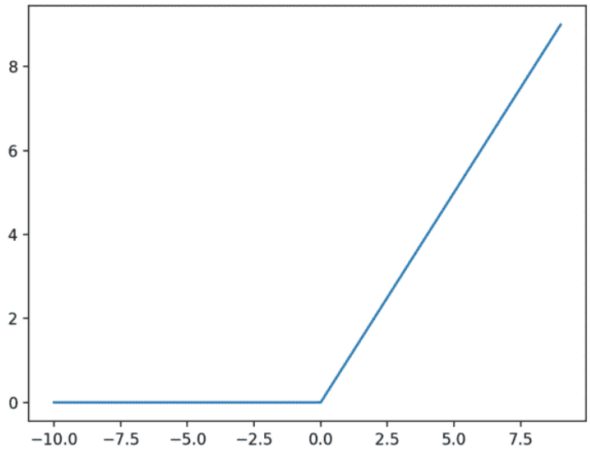



**Figura (2.8): Funzione ReLU.**

- Caratteristiche:
    - Ritorna zero per tutti gli input negativi e l'input stesso per i valori positivi.
    - La ReLU è computazionalmente molto efficiente, poiché non richiede operazioni complesse.
    - Mitiga il problema del vanishing gradient, favorendo l'apprendimento in reti profonde.
    - Tuttavia, può soffrire del problema dei neuroni morti: se un neurone riceve input negativi per tutta la durata dell'addestramento, il suo output rimarrà sempre zero, limitando il suo contributo all'apprendimento.

4. Leaky ReLU, in Figura (2.9), è una variante della ReLU e viene definita come:

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{if } x \leq 0 \end{cases},$$

Dove α è un piccolo valore positivo, tipicamente 0.01.

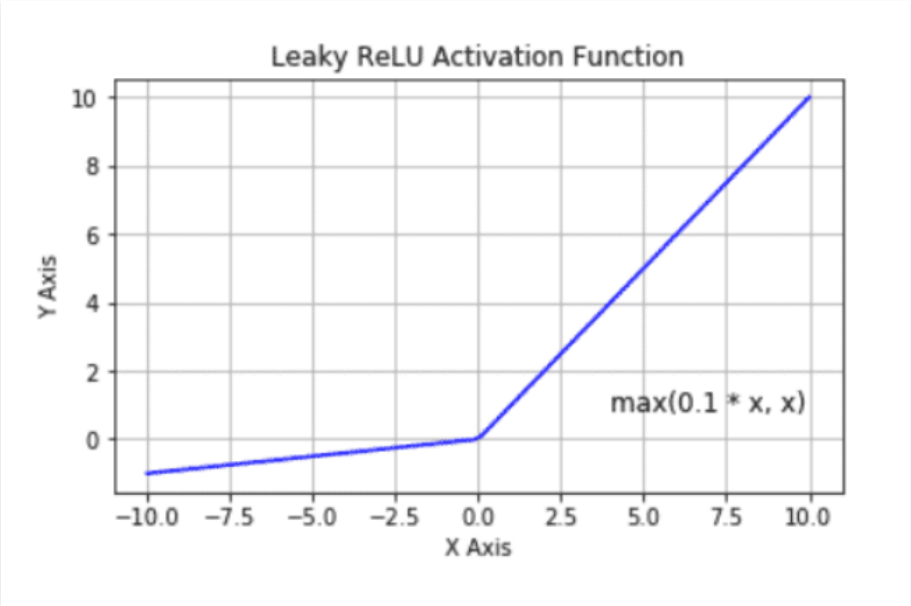



**Figura 2.9: Funzione Leaky ReLU.**

- Caratteristiche:
    - Introduce un piccolo gradiente anche per i valori negativi, risolvendo in parte il problema dei neuroni morti.
    - Questa variante permette ai neuroni di continuare ad apprendere anche quando ricevono input negativi.

L'utilizzo di funzioni sigmoidali e della tangente iperbolica, continue e derivabili in ogni punto, semplificano in maniera significativa l'implementazione *dell'algoritmo di backpropagation,* di cui si parlerà in seguito.

Le funzioni di attivazione, quindi, rappresentano un elemento cruciale nel determinare le capacità di apprendimento di una rete neurale. La loro scelta incide profondamente sulle prestazioni del modello, influenzando non solo la velocità di convergenza durante l'addestramento, ma anche la capacità della rete di generalizzare su nuovi dati. Per selezionare la funzione di attivazione più adatta, è necessario considerare diversi fattori, come la natura del problema da affrontare,

l'architettura della rete e le specifiche esigenze computazionali. Un'adeguata comprensione delle caratteristiche e delle applicazioni delle varie funzioni di attivazione è quindi fondamentale per progettare modelli di rete neurale efficaci, in grado di gestire e risolvere con successo le sfide complesse tipiche dei contesti reali.

# 4. Struttura delle reti neurali: tipologie e architettura

L'architettura di una rete neurale artificiale definisce l'organizzazione e le connessioni tra i neuroni che la compongono. Questa struttura, conosciuta anche come topologia della rete, è determinante per il flusso dell'informazione attraverso i vari strati della rete e per le modalità con cui i neuroni interagiscono tra loro. La configurazione della rete incide direttamente sulla sua capacità di apprendere e risolvere problemi complessi.

## 4.1. Tipologie di reti neurali

Esistono diverse categorie di reti neurali, tra le quali spiccano le reti autoassociative e le reti eteroassociative[15], che si distinguono per la relazione tra lo strato di input e quello di output.

- Reti Autoassociative: In queste reti, lo strato di input coincide con lo strato di output. I neuroni sono interamente connessi e la rete ha lo scopo di riprodurre in output gli stessi dati ricevuti in input. Questa struttura si presta particolarmente a compiti di compressione dei dati, riduzione del rumore

[15] Agliari, E., Alessandrelli, A., Barra, A., Centonze, M. S., & Ricci-Tersenghi, F. (2024). Generalized hetero-associative neural networks. *arXiv preprint arXiv:2409.08151*.

(denoising) e rilevamento di anomalie. Un esempio tipico di rete autoassociativa è l'autoencoder.

- Reti Eteroassociative: In queste reti, lo strato di input è distinto da quello di output. La rete apprende a mappare un insieme di input verso un diverso insieme di output, come avviene nei problemi di classificazione o regressione. Le reti eteroassociative possono essere classificate ulteriormente in base al numero di strati e alla loro organizzazione interna.

Inoltre, le connessioni tra i neuroni all'interno di una rete possono variare notevolmente e influenzano la capacità della rete di apprendere relazioni complesse. Si possono costruire:

- Connessioni Intrastrato: Si riferiscono ai collegamenti tra neuroni all'interno dello stesso strato. Queste connessioni favoriscono la comunicazione laterale tra i neuroni di uno stesso livello, permettendo una più ricca interazione interna.
- Reti Totalmente Connesse: Una rete è detta totalmente connessa quando ciascun neurone è collegato a tutti gli altri neuroni sia all'interno dello stesso strato, sia tra strati diversi. Questa configurazione aumenta la capacità della rete di catturare relazioni complesse tra i dati, ma può portare al rischio di overfitting, poiché la rete potrebbe adattarsi eccessivamente ai dati di addestramento.
- Reti Interconnesse: In queste reti, i neuroni di uno strato sono collegati esclusivamente ai neuroni degli strati precedenti o successivi, senza connessioni interne allo stesso strato. Questo tipo di organizzazione riduce la complessità del modello, semplificando il processo di apprendimento.

## 4.2. Evoluzione dei modelli di rete

Il Perceptron di Rosenblatt rappresenta il modello base delle reti neurali. È costituito da due soli strati: uno strato di input e uno strato di output, collegati da una serie di pesi. Questo modello può risolvere problemi di classificazione lineare, ma presenta limitazioni quando si affrontano problemi più complessi.

Le reti neurali moderne, come le Multilayer Perceptron (MLP), hanno superato queste limitazioni. Le MLP includono più strati di neuroni, suddivisi in uno strato di input, uno o più strati nascosti e uno strato di output. Gli strati nascosti, che non interagiscono direttamente con l'esterno, permettono alla rete di apprendere le complessità dei dati e di modellare relazioni intricate. Il numero di strati nascosti e di neuroni per strato può variare significativamente in base alla complessità del problema da risolvere[16].

Un'altra importante distinzione tra le reti neurali è il modo in cui l'informazione fluisce attraverso la rete:

- Reti Feedforward: In queste reti, le connessioni tra i neuroni sono unidirezionali: l'informazione scorre dagli input verso gli output passando attraverso gli strati nascosti. Non esistono cicli o retroazioni, il che rende queste reti adatte a compiti come la classificazione e la predizione. Le MLP sono un classico esempio di rete feedforward.
- Reti Feedback o Ricorrenti (RNN): In questo tipo di rete, le connessioni tra neuroni possono essere bidirezionali. L'output di un neurone può essere reintrodotto come input allo stesso livello o a livelli precedenti, creando così

[16] Popescu, M. C., Balas, V. E., Perescu-Popescu, L., & Mastorakis, N. (2009). Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, *8*(7), 579-588.

una memoria temporale. Questa caratteristica rende le reti ricorrenti particolarmente efficaci nell'elaborazione di dati sequenziali o temporali.

## 4.3. Tecniche per la progettazione delle reti neurali

La progettazione di una rete neurale richiede la definizione accurata della sua architettura. Esistono diverse tecniche per ottimizzare la struttura della rete, tra cui il trial and error, il cascade correlation e i metodi di pruning[17].

- Trial and Error: Questo approccio empirico consiste nel provare diverse configurazioni di rete (numero di strati, neuroni per strato, funzioni di attivazione, ecc.) per trovare quella che fornisce le migliori prestazioni. Sebbene sia flessibile e adattabile, può risultare computazionalmente costoso e non garantisce di raggiungere la configurazione ottimale.
- Cascade Correlation: Questo algoritmo costruisce progressivamente la rete durante l'addestramento, partendo da una rete minima e aggiungendo nuovi neuroni solo quando necessario. Il processo consente di ottimizzare la rete in modo incrementale, riducendo il tempo di addestramento, ma può risultare complesso e portare a reti molto grandi con numerosi neuroni e connessioni.
- Metodi di Pruning: Questi metodi mirano a ridurre la complessità della rete eliminando pesi o neuroni che hanno un impatto marginale sulle prestazioni complessive. Il pruning migliora l'efficienza computazionale della rete e può ridurre il rischio di overfitting, migliorando la capacità di generalizzazione del modello.

[17] Basheer, I. A., & Hajmeer, M. (2000). Artificial neural networks: fundamentals, computing, design, and application. *Journal of microbiological methods*, *43*(1), 3-31.

# 5. Addestramento delle reti neurali artificiali

L'addestramento di una rete neurale artificiale rappresenta un processo cruciale attraverso il quale il modello apprende dai dati e migliora progressivamente le proprie prestazioni. Questo apprendimento avviene mediante l'aggiornamento iterativo dei pesi sinaptici e dei bias associati ai neuroni, con l'obiettivo di minimizzare l'errore tra le predizioni generate dalla rete e i valori attesi.

Il processo prende avvio con la propagazione in avanti (forward propagation), durante la quale gli input vengono introdotti nello strato iniziale della rete e attraversano successivamente i vari strati nascosti fino a raggiungere lo strato di output. In ogni neurone, viene calcolata una combinazione lineare degli input ponderati, a cui viene sommato un termine di bias, seguendo la formula:

$$z_i = \sum_j w_{ij}\, x_j + b_i$$

dove $w_{ij}$ rappresenta il peso della connessione tra l'input $x_j$ e il neurone $i$, e $b_i$ è il bias associato al neurone $i$.

Una volta ottenuto questo valore $z_i$, viene applicata una funzione di attivazione non lineare $\phi$, come la sigmoide, la ReLU o la tangente iperbolica, che trasforma il potenziale netto in un output attivato:

$$a_i = \phi(z_i)$$

La funzione di attivazione introduce non linearità nel modello, un aspetto fondamentale per permettere alla rete di apprendere relazioni complesse tra input e output. Questo processo si ripete per ciascun neurone di ogni strato, fino a generare l'output finale $\hat{y}$.

Dopo aver ottenuto l'output predetto, viene calcolata la funzione di costo (o loss function), che misura la discrepanza tra l'output della rete $\hat{y}$ e il valore atteso y. La funzione di costo varia a seconda del tipo di problema affrontato. Ad esempio, per problemi di regressione si utilizza comunemente l'Errore Quadratico Medio (MSE):

$$J(w,b) = \frac{1}{N}\sum_{i=1}^{N}(\hat{y_i} - y_i)^2$$

Mentre per problemi di classificazione, si preferisce impiegare la Cross-Entropia:

$$J(w,b) = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(\hat{y_i}) + (1 - y_i)\log(1 - \hat{y_i})]$$

Successivamente, si passa alla fase di retropropagazione dell'errore (backpropagation), che consente di calcolare il gradiente della funzione di costo rispetto ai pesi e ai bias della rete. Questo calcolo avviene tramite l'applicazione della regola della catena, che permette di propagare l'errore indietro attraverso la rete, partendo dall'output fino a raggiungere gli strati iniziali. I gradienti ottenuti forniscono le informazioni necessarie per aggiornare i pesi e minimizzare l'errore.

Per aggiornare i pesi e i bias, si utilizzano algoritmi di ottimizzazione basati sulla discesa del gradiente. Il peso $w_{ij}$ viene aggiornato secondo la seguente regola:

$$w_{ij} \leftarrow w_{ij} - \eta\frac{\partial J}{\partial w_{ij}}$$

dove $\eta$ è il tasso di apprendimento (learning rate), un iperparametro che controlla la dimensione dei passi compiuti nell'aggiornamento dei parametri[18].

Un'estensione della discesa del gradiente standard è l'introduzione del momentum, una tecnica che accelera la convergenza e riduce il rischio di oscillazioni o blocchi in minimi locali. Il momentum aggiunge una frazione della variazione precedente dei pesi all'aggiornamento corrente, simulando l'effetto dell'inerzia:

$$v_{ij} = \gamma v_{ij} - \eta \frac{\partial J}{\partial w_{ij}}$$

$$w_{ij} \leftarrow w_{ij} + v_{ij}$$

dove $v_{ij}$ è la velocità associata al peso $w_{ij}$ e $\gamma$ è il coefficiente di momentum, solitamente compreso tra 0 e 1.

L'addestramento procede iterativamente, con l'intero dataset attraversato più volte durante cicli detti epoche. Un'attenzione particolare va posta nel monitorare le prestazioni della rete, per evitare il problema dell'overfitting, ovvero quando la rete si adatta troppo ai dati di addestramento, perdendo capacità di generalizzazione su dati non visti.

Per mitigare l'overfitting, si utilizzano diverse tecniche, tra cui:

- Regolarizzazione: Aggiunge un termine alla funzione di costo che penalizza i pesi di grande magnitudine, incoraggiando la rete a mantenere i pesi piccoli e a ridurre la complessità del modello. Ad esempio, nella regolarizzazione L2, la funzione di costo diventa:

$$J_{reg} = J + \lambda \sum_{i,j} w_{ij}^2$$

[18] Günther, F., & Fritsch, S. (2010). Neuralnet: training of neural networks. *R J.*, *2*(1), 30.

dove λ è il coefficiente di regolarizzazione.

- Dropout: Durante l'addestramento, si disattivano casualmente alcuni neuroni con una certa probabilità, impedendo ai neuroni di diventare troppo dipendenti dagli altri e migliorando la capacità della rete di generalizzare.
- Data Augmentation: Questa tecnica aumenta la varietà dei dati di addestramento, applicando trasformazioni ai dati esistenti, come rotazioni o scalature nel caso di immagini, incrementando così la diversità del dataset.

Per valutare quando interrompere l'addestramento, si utilizza un set di validazione, un sottoinsieme di dati non utilizzato per l'addestramento. Monitorando le prestazioni su questo set, è possibile applicare tecniche come l'early stopping, che prevede di arrestare l'addestramento quando le prestazioni sul set di validazione non migliorano più per un certo numero di epoche consecutive, prevenendo così il sovradattamento ai dati di addestramento.

L'addestramento può anche essere interrotto al raggiungimento di un numero massimo di epoche, quando la funzione di costo scende sotto una soglia specifica, o quando l'accuratezza raggiunge un valore prestabilito.

In sintesi, l'addestramento di una rete neurale artificiale è un processo iterativo che coinvolge la propagazione in avanti degli input, il calcolo della funzione di costo, la retropropagazione dell'errore e l'aggiornamento dei pesi. Questi passaggi, combinati con tecniche di ottimizzazione e prevenzione dell'overfitting, permettono alla rete di apprendere dai dati e migliorare le proprie prestazioni nel tempo.

# 6. Metriche di valutazione della performance

La scelta delle metriche di valutazione dipende strettamente dal tipo di problema affrontato, che può essere classificazione, regressione o altri. Per i problemi di classificazione si usa come strumento principale la matrice di confusione che riassume le prestazioni del modello classificando i veri positivi, i falsi positivi i veri negativi e i falsi negativi. Alcune delle metriche più utilizzate includono:

- Accuratezza (Accuracy): Misura la percentuale di unità correttamente classificate rispetto al totale.

$$\text{Accuratezza} = \frac{\text{Numero di predizioni corrette}}{\text{Numero totale di osservazioni}}$$

- Precisione, Richiamo e F1-Score: Forniscono ulteriori approfondimenti sulle prestazioni della rete, tenendo conto del bilanciamento tra falsi positivi e falsi negativi. Ad esempio:
    - Precisione è data da:

$$\text{Precisione} = \frac{\text{Veri Positivi}}{\text{Veri Positivi} + \text{Falsi Positivi}}$$

    - Richiamo (Recall) misura la capacità di individuare correttamente le istanze positive:

$$\text{Richiamo} = \frac{\text{Veri Positivi}}{\text{Veri Positivi} + \text{Falsi Negativi}}$$

    - F1-Score rappresenta la media armonica di precisione e richiamo:

$$F1 = 2 \times \frac{\text{Precisione} \times \text{Richiamo}}{\text{Precisione} + \text{Richiamo}}$$

- AUC-ROC (Area Under the Curve - Receiver Operating Characteristic): traccia la curva dei tassi di veri positivi contro i falsi positivi. Attraverso la curva ROC è possibile identificare il best cut-off, cioè il valore soglia che massimizza la differenza tra veri positivi e falsi positivi.

Per i problemi di regressione, le metriche più comuni includono:

- Errore Quadratico Medio (MSE), che misura la media degli errori quadrati tra i valori predetti e quelli reali:

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

- Errore Assoluto Medio (MAE), che misura la media degli errori assoluti:

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

- Coefficiente di Determinazione ($R^2$), che indica la proporzione della varianza nei dati spiegata dal modello:

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

Dove $\bar{y}$ è il valore medio dei dati osservati.

# 7. Reti Multi-Layer Perceptron (MLP)

Con quanto argomentato finora, è stato delineato il funzionamento generale delle reti neurali artificiali, descrivendo la propagazione dell'informazione attraverso gli strati e il meccanismo di aggiornamento dei pesi e dei bias. Tra le diverse architetture disponibili, le Reti Multi-Layer Perceptron (MLP) sono tra le più ampiamente utilizzate in virtù della loro flessibilità e capacità di risolvere problemi complessi. Le MLP, grazie alla loro struttura multistrato, sono in grado di modellare relazioni non lineari tra input e output, rendendole particolarmente adatte a una vasta gamma di applicazioni, dalla classificazione alla regressione[19].

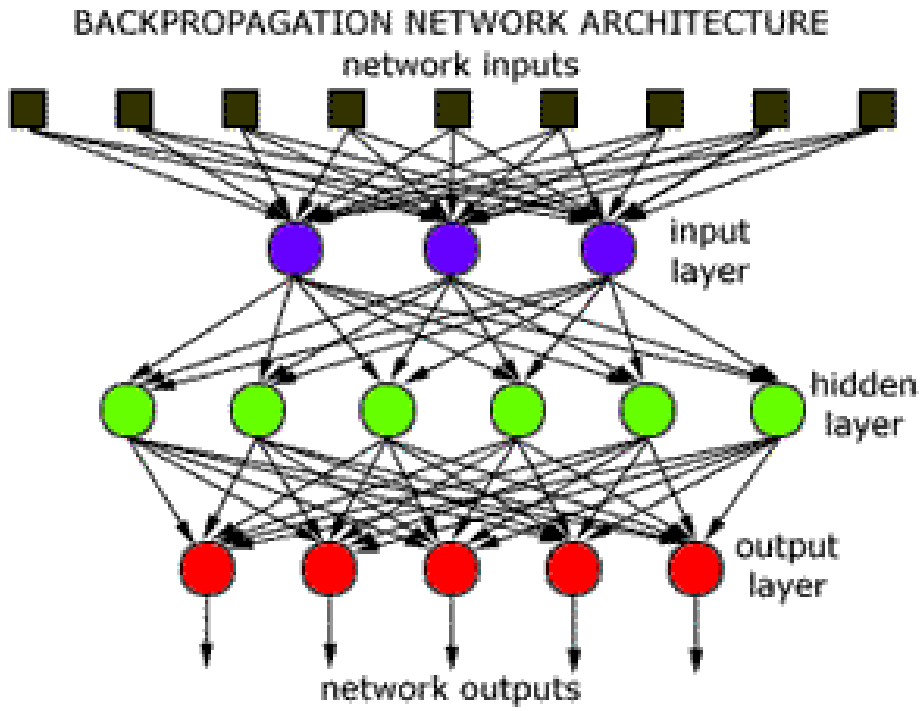



**Figura 2.10: Rete Multi-Layer Perceptron.**
**(https://polilabhome.polito.it/2021/03/25/uno-sguardo-sulle-reti-neurali/)**

[19] Popescu, M. C., Balas, V. E., Perescu-Popescu, L., & Mastorakis, N. (2009). Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, *8*(7), 579-588.

Le MLP estendono il modello del Perceptron semplice di Rosenblatt, superando le limitazioni imposte dalla separabilità lineare. Ciò è reso possibile dalla presenza di uno o più strati nascosti, come rappresentato in Figura (2.10), che consentono alla rete di apprendere rappresentazioni complesse dei dati. Questa capacità trova fondamento teorico nel teorema dell'approssimazione universale, uno dei risultati più significativi nel campo delle reti neurali artificiali.

## 7.1. Il Teorema dell'Approssimazione Universale[20]

Il teorema dell'approssimazione universale, formulato da George Cybenko nel 1989 e successivamente generalizzato da Kurt Hornik, dimostra che una rete neurale feedforward con un singolo strato nascosto e una funzione di attivazione non lineare è in grado di approssimare qualsiasi funzione continua definita su uno spazio compatto, con un livello di precisione arbitrario, purché la rete disponga di un numero sufficiente di neuroni. In altre parole, una rete MLP con un solo strato nascosto ha la capacità teorica di rappresentare qualsiasi funzione, indipendentemente dalla complessità della relazione tra input e output.

Questo risultato teorico è di fondamentale importanza poiché garantisce che, almeno in linea di principio, una MLP possa affrontare e risolvere una vasta gamma di problemi, anche quelli caratterizzati da relazioni non lineari estremamente complesse. La condizione essenziale è la presenza di un numero adeguato di neuroni nello strato nascosto, i quali fungono da "blocchi di costruzione" che permettono alla rete di approssimare con precisione qualsiasi funzione target.

Tuttavia, è importante sottolineare che, pur fornendo una garanzia di potenziale teorico, il teorema non specifica né il numero minimo di neuroni né la struttura ottimale della rete per una data applicazione. Questo implica che, nella pratica, il

[20] Nishijima, T. (2021). Universal approximation theorem for neural networks. *arXiv preprint arXiv:2102.10993*.

successo delle MLP dipende dalla capacità di progettare una rete con una configurazione di iperparametri adeguata al problema specifico.

## 7.2. Vantaggi e svantaggi delle MLP

Uno dei principali vantaggi delle MLP è la loro versatilità. La capacità di apprendere e rappresentare relazioni non lineari le rende ideali per risolvere problemi complessi in numerosi contesti applicativi, tra cui la classificazione multi-classe, la regressione e il riconoscimento di pattern. Grazie alla loro architettura multistrato, le MLP sono in grado di catturare strutture intricate nei dati, consentendo loro di generalizzare su dati non visti e di essere impiegate in una vasta gamma di domini, dai sistemi di raccomandazione all'analisi di immagini e dati temporali.

Nonostante il loro potenziale teorico, le MLP presentano alcune limitazioni pratiche. Una delle sfide principali riguarda la scalabilità. Con l'aumento del numero di strati e neuroni, il costo computazionale dell'addestramento cresce esponenzialmente, rendendo difficile la gestione di reti molto complesse senza l'uso di hardware specializzato, come le GPU. Questo aspetto può risultare particolarmente problematico quando si lavora con dataset di grandi dimensioni o con problemi che richiedono reti molto profonde per rappresentare accuratamente le relazioni nei dati.

Un'altra criticità risiede nella selezione degli iperparametri. Determinare il numero di neuroni e strati necessari per risolvere un problema specifico, così come scegliere la funzione di attivazione più adatta, richiede spesso un processo empirico e iterativo, il che può essere costoso in termini di tempo e risorse[21].

[21] https://www.researchgate.net/figure/Multilayer-Perceptron-Advantages-and-Disadvantages_tbl4_338950098

Infine, sebbene le MLP siano efficaci in molte applicazioni, presentano limitazioni quando si tratta di gestire strutture dati particolarmente complesse o dinamiche, dove i modelli più avanzati, come le Kolmogorov-Arnold Networks (KAN), si sono dimostrati più promettenti. Le KAN cercano di superare alcune delle difficoltà tipiche delle MLP, come la capacità di gestire funzioni più complesse e l'ottimizzazione dei modelli per migliorare l'efficienza computazionale e la capacità di generalizzazione. Il loro sviluppo rappresenta un'evoluzione naturale delle MLP, cercando di affrontare problemi legati alla scalabilità e alla rappresentazione accurata di funzioni più complesse.

Questo avanzamento teorico e pratico verso le KAN offre una nuova prospettiva sulle reti neurali, evidenziando come il campo dell'intelligenza artificiale stia progredendo verso soluzioni sempre più sofisticate, capaci di risolvere problemi che tradizionalmente risultavano difficili da affrontare con le architetture MLP convenzionali.

# CAPITOLO III

# KOLMOGOROV-ARNOLD NETWORK (KAN): ALTERNATIVA PROMETTENTE

SOMMARIO: 1. Reti Kolmogorov-Arnold (KAN): obiettivi e meccanismo di funzionamento – 2. Il Teorema Kolmogorov-Arnold – 3. Le B-Spline: costruzione proprietà e uso nelle KAN – 3.1. Costruzione delle B-Spline – 3.2. Proprietà delle B-Spline – 3.3. Uso nelle reti KAN – 5. Tecnica della Grid Extension – 6. Implementazione delle reti KAN – 6.1. Parametrizzazione delle Funzioni Univariate tramite B-Spline – 6.2. Addestramento delle KAN – 7. Tecniche di Semplificazione e Interpretabilità: Sparsificazione, Visualizzazione, Pruning e Symbolification – 8. Legge di Scaling nelle reti KAN – 9. Il processo del Continual Learning nelle Reti KAN – 10. Risoluzione di Equazioni Differenziali Parziali (PDE) nelle reti KAN – 11. Vantaggi e svantaggi delle reti KAN.

# 1. Reti Kolmogorov-Arnold (KAN): obiettivi e meccanismo di funzionamento

Le Reti Kolmogorov-Arnold (KAN) sono un'innovativa architettura di reti neurali che si distinguono per la loro capacità di risolvere problemi complessi di approssimazione di funzioni multivariate[22]. L'idea alla base delle KAN si fonda sul teorema di rappresentazione di Kolmogorov-Arnold, che garantisce che qualsiasi funzione multivariata continua definita su un dominio limitato può essere scomposta in una somma finita di funzioni univariate. Questo teorema fornisce un approccio rivoluzionario per affrontare il problema della maledizione della dimensionalità (curse of dimensionality), comune nelle tradizionali reti neurali come le Multi-Layer Perceptron (MLP).

Le reti MLP, pur essendo versatili e molto utilizzate, incontrano notevoli difficoltà nell'affrontare problemi che comportano la gestione di dati ad alta dimensionalità. Questi problemi si manifestano sotto forma di una crescita esponenziale del numero di parametri richiesti per rappresentare correttamente le funzioni non lineari complesse. Le KAN, grazie alla loro struttura, riescono a ridurre significativamente il numero di parametri, permettendo di risolvere problemi che per le MLP sarebbero proibitivi in termini di costo computazionale.

Nelle KAN, invece di aumentare la complessità attraverso l'aggiunta di nuovi nodi e strati come nelle MLP, si rappresentano le funzioni complesse attraverso una decomposizione in componenti univariate. Questa scomposizione non solo riduce il numero di parametri necessari, ma permette anche di ottenere un modello più interpretabile, poiché ogni funzione univariata può essere esaminata individualmente.

[22] Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Soljačić, M., ... & Tegmark, M. (2024). Kan: Kolmogorov-arnold networks. *arXiv preprint arXiv:2404.19756*.

# 2. Il Teorema di Kolmogorov-Arnold

Il teorema di Kolmogorov-Arnold è il fondamento matematico su cui si basano le reti KAN. Introdotto da Andrey Kolmogorov e Vladimir Arnold, questo teorema ha cambiato il modo in cui è possibile rappresentare le funzioni multivariate, offrendo una nuova via per risolvere problemi di apprendimento e di approssimazione di funzioni complesse.

Il teorema di Kolmogorov-Arnold originariamente stabilisce che qualsiasi funzione multivariata può essere rappresentata come somma e composizione di funzioni univariate. Questo porta a una rete neurale con due strati, in cui il primo strato contiene funzioni univariate che mappano le variabili di input, e il secondo strato contiene funzioni che combinano questi output in una singola funzione finale. In termini di rete neurale, questa è una rete a due strati, uno strato nascosto e uno strato di output[23].

Matematicamente, è rappresentato come:

$$f(x) = \sum_{q=1}^{2n+1} \Phi_q \left( \sum_{p=1}^{n} \varphi_{q,p}(x_p) \right)$$

Tuttavia, questa struttura è troppo semplice per approssimare accuratamente funzioni multivariate complesse, specialmente quando tali funzioni richiedono una rappresentazione più sofisticata. Questo perché con un solo livello nascosto, anche se il numero di neuroni è grande, non è sufficiente per catturare la complessità di funzioni reali, soprattutto quando si vogliono gestire dipendenze non lineari complesse tra le variabili.

[23] Schmidt-Hieber, J. (2021). The Kolmogorov–Arnold representation theorem revisited. *Neural networks*, *137*, 119-126.

## 2.1. La necessità di reti più profonde

Per superare queste limitazioni, diventa necessario estendere la struttura delle KAN verso reti più profonde, simili a quanto avviene con le MLP (Multi-Layer Perceptron). Nelle MLP, infatti, si aggiungono strati ulteriori, "impilando" livelli uno sopra l'altro, per gestire in modo più efficace le complesse dipendenze non lineari.

La sfida nel contesto delle KAN, tuttavia, è che il teorema di Kolmogorov-Arnold, nella sua forma originaria, riguarda esclusivamente una rappresentazione a due strati. Questo richiede quindi una generalizzazione del teorema, che permetta di considerare più strati, ciascuno definito da funzioni univariate.

## 2.2. Definizione di un "KAN layer"

Per creare reti KAN più profonde, è essenziale definire correttamente cosa rappresenti un layer all'interno di una rete KAN. La soluzione proposta è che ogni strato di una KAN sia rappresentato da una matrice di funzioni univariate.

Matematicamente, un KAN layer con $n_{in}$ ingressi e $n_{out}$ uscite è descritto come una matrice di funzioni univariate:

$$\Phi = \{\phi_{q,p}\}, \quad p = 1,2,\dots,n_{in}, \quad q = 1,2,\dots,n_{out}.$$

Questo significa che ogni connessione tra un nodo del livello di input (il nodo p-esimo) e un nodo del livello di output (il nodo q-esimo) è rappresentata da una funzione univariata $\phi_{q,p}$, che viene appresa durante il processo di addestramento.

## 2.3. Struttura di una KAN profonda

Una volta compreso come rappresentare un singolo strato come matrice di funzioni univariate, è possibile costruire KAN più profonde (Figura 3.1) semplicemente impilando più strati, come avviene nelle MLP.

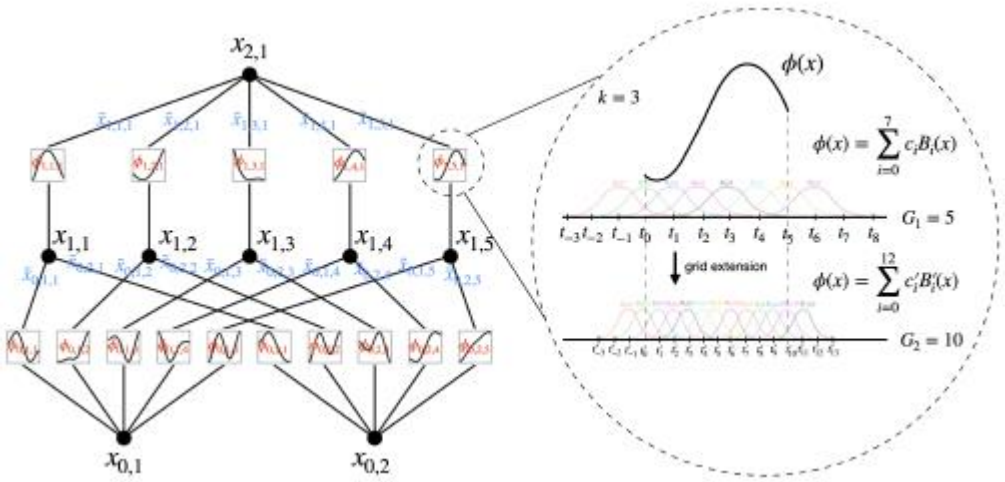



**Figura 3.1 – Rete KAN con 1 layer intermedio**

La differenza sta nel fatto che, nelle KAN, ogni strato è composto da funzioni univariate, invece che da pesi lineari.

L'output finale della rete KAN è quindi il risultato della composizione delle matrici di funzioni corrispondenti ai vari strati. Per una rete KAN con L strati, l'output f(x) è espresso come:

$$f(x) = (\Phi_{L-1} \circ \Phi_{L-2} \circ \cdots \circ \Phi_0)(x_0)$$

dove $\Phi_l$ rappresenta la matrice di funzioni univariate del livello l-esimo.

La rappresentazione matriciale delle reti KAN rende il modello altamente flessibile e modulare. Ogni trasformazione tra strati può essere trattata separatamente, il che facilita sia l'implementazione che la comprensione del comportamento della rete.

# 3. Le B-Spline: costruzione, proprietà, e uso nelle KAN

Le B-spline sono un elemento essenziale nelle reti Kolmogorov-Arnold, in quanto vengono utilizzate per rappresentare le funzioni univariate $\varphi_{q,p}(x_p)$ che compongono il modello. Le B-spline sono particolarmente adatte a questo scopo per le loro proprietà di località e flessibilità nella rappresentazione di funzioni complesse.

## 3.1. Costruzione delle B-Spline[24]

Una B-spline è una funzione spline composta da pezzi polinomiali di grado k, definiti su un intervallo limitato del dominio. Le B-spline sono costruite su una griglia di nodi $t_0, t_1, \dots, t_n$, che definiscono gli intervalli in cui la funzione è attiva. La costruzione delle B-spline avviene in maniera ricorsiva:

- B-spline di grado 0: È una funzione indicatrice su un intervallo:

$$B_{i,0}(x) = \begin{cases} 1 & se\ t_i \leq x < t_{i+1}, \\ 0, altrimenti \end{cases}$$

Questa è una funzione a tratti costante, attiva solo su un intervallo tra due nodi consecutivi.

- B-spline di grado k: Per k>0, la B-spline è definita come una combinazione lineare di B-spline di grado inferiore:

$$B_{i,k}(x) = \frac{x - t_i}{t_{i+k} - t_i} B_{i,k-1}(x) + \frac{t_{i+k+1} - x}{t_{i+k+1} - t_{i+1}} B_{i+1,k-1}(x)$$

Questa combinazione permette di costruire una funzione continua e liscia, a seconda del grado k.

[24] Prautzsch, H. (2002). Bézier and B-spline techniques.

## 3.2. Proprietà delle B-Spline

Le B-spline presentano una serie di proprietà che le rendono particolarmente adatte alla rappresentazione delle funzioni univariate nelle reti KAN:

1. Supporto Locale: Ogni B-spline è attiva solo su un intervallo limitato del dominio, il che significa che modificare un coefficiente influenza solo una piccola parte della funzione. Questo riduce la complessità computazionale dell'ottimizzazione.

2. Linearità: Le B-spline formano una base lineare, il che significa che ogni funzione univariata può essere espressa come una combinazione lineare di B-spline. Questo rende facile ottimizzare i coefficienti delle B-spline durante l'addestramento della rete.

3. Smoothness: Il grado k della B-spline determina il livello di smoothness della funzione. Una B-spline di grado k ha continuità $C^{k-1}$ , ovvero è continua fino alla derivata k−1. Questo permette di rappresentare funzioni con diversi livelli di liscezza.

## 3.3. Uso nelle reti KAN

Nelle KAN, le B-spline sono utilizzate per rappresentare le funzioni univariate $\varphi_{q,p}(x_p)$ , che vengono apprese durante il processo di addestramento. Ogni funzione univariata è parametrizzata da una serie di coefficienti, che definiscono la combinazione lineare di B-spline. L'apprendimento di questi coefficienti avviene attraverso la backpropagation, un processo di ottimizzazione che aggiorna i coefficienti per minimizzare l'errore tra l'output previsto e quello target.

L'uso delle B-spline permette di rappresentare funzioni non lineari complesse in modo flessibile, mantenendo al contempo una struttura locale e computazionalmente efficiente. Questo è particolarmente vantaggioso nelle reti

KAN, dove la capacità di adattarsi a diverse regioni del dominio senza influenzare l'intera funzione è cruciale per ottenere un buon compromesso tra precisione e complessità.

# 4. Tecnica della Grid Extension

Una delle tecniche più importanti utilizzate nelle reti KAN è la Grid Extension, che consente di aumentare la risoluzione della griglia su cui sono definite le B-spline per migliorare l'accuratezza dell'approssimazione delle funzioni target.

All'inizio dell'addestramento, le funzioni univariate nelle KAN sono definite su una griglia iniziale con un numero limitato di nodi. Questa griglia rappresenta i punti in cui vengono calcolate le B-spline. Tuttavia, una griglia troppo grossolana potrebbe non catturare accuratamente le variazioni rapide della funzione target, specialmente in aree in cui la funzione varia velocemente.

La Grid Extension risolve questo problema aggiungendo nuovi nodi alla griglia durante l'addestramento. Questo processo avviene progressivamente: la rete KAN parte con una griglia di bassa risoluzione e, man mano che l'addestramento procede, la griglia viene estesa aggiungendo nuovi punti di controllo (vedi Figura 3.2). In questo modo, la rappresentazione della funzione viene raffinata, soprattutto nelle aree critiche dove la funzione presenta oscillazioni o picchi.

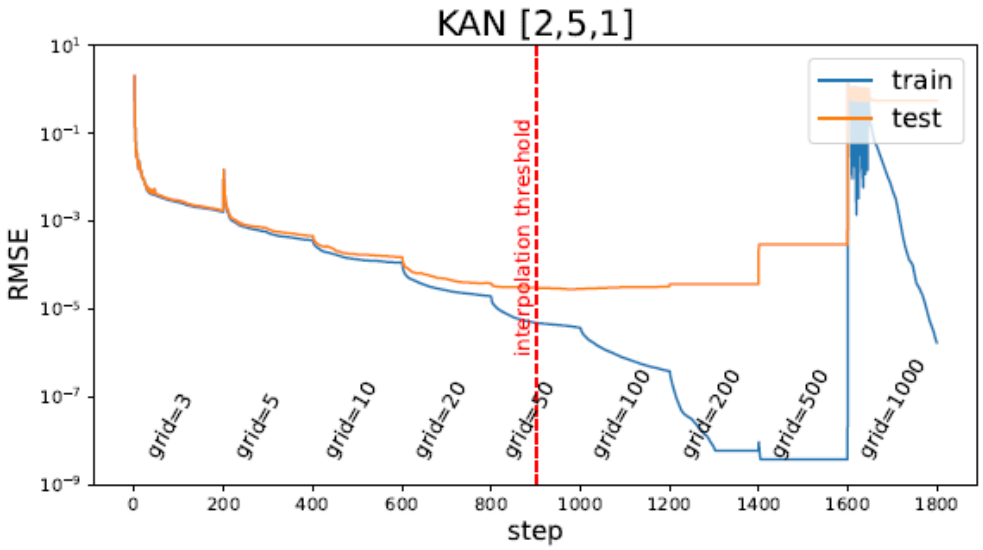



**Figura 3.2 – Espansione della griglia**

Vi sono una serie di vantaggi derivanti da tale tecnica:

1. Maggiore Accuratezza: L'estensione della griglia permette di migliorare l'accuratezza della rappresentazione delle funzioni, poiché la rete può adattarsi meglio alle variazioni locali della funzione target.

2. Efficienza Computazionale: Un vantaggio cruciale della Grid Extension è che la rete non deve essere riaddestrata completamente quando si estende la griglia. I nuovi nodi possono essere aggiunti senza ripartire da zero, rendendo questo processo molto efficiente in termini di costo computazionale.

3. Adattabilità: La Grid Extension consente alla rete KAN di adattarsi a funzioni con variazioni locali molto rapide. Ad esempio, se una funzione presenta un picco in una specifica regione del dominio, l'aggiunta di nodi in quella regione permette alla rete di rappresentare con maggiore precisione quella porzione della funzione senza dover aumentare la risoluzione su tutto il dominio.

Un esempio tipico dell'utilizzo della Grid Extension nelle KAN è l'approssimazione di funzioni con picchi o discontinuità localizzate. In queste situazioni, la rete inizia con una rappresentazione generale della funzione e, durante l'addestramento, estende la griglia solo nelle aree in cui è necessaria una maggiore precisione. Questo approccio permette di ottenere un buon compromesso tra precisione e costo computazionale.

# 5. Implementazione delle reti KAN

L'implementazione delle reti Kolmogorov-Arnold richiede una serie di passaggi specifici per configurare e addestrare la rete in modo efficiente. Un aspetto distintivo delle KAN è che, a differenza delle MLP, l'addestramento non si concentra sull'ottimizzazione di pesi lineari, ma sui coefficienti delle B-spline che rappresentano le funzioni univariate.

## 5.1. Parametrizzazione delle Funzioni Univariate tramite B-Spline

Ogni funzione univariata $\varphi_{q,p}(x_p)$ nelle KAN è rappresentata come una combinazione lineare di B-spline:

$$\varphi_{q,p}(x_p) = \sum_i c_i B_{i,k}(x_p)$$

dove:

- $c_i$ sono i coefficienti delle B-spline da ottimizzare durante l'addestramento.
- $B_{i,k}$ sono le B-spline di grado k definite su una griglia di nodi.

L'ottimizzazione dei coefficienti $c_i$ avviene tramite backpropagation, un algoritmo di ottimizzazione che calcola i gradienti dell'errore rispetto ai parametri della rete e li utilizza per aggiornare i coefficienti delle B-spline.

## 5.2. Addestramento delle KAN

Il processo di addestramento delle KAN segue lo schema tradizionale delle reti neurali, con alcune differenze chiave. L'addestramento avviene attraverso i seguenti passaggi:

1. Propagazione in Avanti: Per ogni input, la rete calcola l'output utilizzando le funzioni univariate parametrizzate dalle B-spline. Questo output viene confrontato con l'output target per calcolare l'errore.
2. Calcolo dei Gradienti: L'errore viene retropropagato attraverso la rete, e i gradienti dell'errore rispetto ai coefficienti delle B-spline vengono calcolati utilizzando il metodo della backpropagation. Questo passaggio è simile a quello delle MLP, ma nelle KAN i gradienti vengono calcolati rispetto ai coefficienti delle spline, non rispetto ai pesi lineari.
3. Aggiornamento dei Coefficienti: I coefficienti delle B-spline vengono aggiornati utilizzando una regola di aggiornamento basata sul gradiente, come l'algoritmo di discesa del gradiente o sue varianti (es. Adam).
4. Grid Extension (Se Applicata): Se necessario, durante l'addestramento possono essere aggiunti nuovi nodi alla griglia per aumentare la precisione della rappresentazione delle funzioni univariate.

Questo processo di addestramento consente alle KAN di apprendere una rappresentazione precisa delle funzioni multivariate target, con un numero ridotto di parametri rispetto alle MLP.

# 6. Tecniche di Semplificazione e Interpretabilità: Sparsificazione, Visualizzazione, Pruning e Symbolification

Per rendere le reti KAN più efficienti e interpretabili, vengono utilizzate diverse tecniche di semplificazione e ottimizzazione. Queste tecniche aiutano a ridurre la complessità della rete senza sacrificare la sua capacità di approssimazione.

La sparsificazione è una tecnica che riduce il numero di attivazioni nella rete eliminando quelle che non contribuiscono significativamente all'output. Nelle KAN, la sparsificazione può essere applicata utilizzando la regolarizzazione L1, che penalizza i coefficienti delle funzioni univariate con valori molto piccoli, portando a una riduzione del numero di attivazioni attive. Questa tecnica aiuta a ridurre la complessità della rete, migliorando al contempo la sua capacità di generalizzazione.

Matematicamente, la regolarizzazione L1 per una funzione univariata $\varphi$ può essere scritta come:

$$|\varphi|_{L1} = \frac{1}{N_p}\sum_{g=1}^{N_p}\left|\varphi\left(x^{(g)}\right)\right|$$

dove $N_p$ è il numero di campioni e $\left(x^{(g)}\right)$ sono gli input.

La visualizzazione delle attivazioni, invece, è un altro strumento importante per l'interpretabilità delle KAN. Permette di visualizzare quali funzioni univariate contribuiscono maggiormente all'output della rete. In questo modo, le attivazioni meno importanti possono essere oscurate o eliminate, rendendo più chiaro il funzionamento interno del modello.

È importante ulteriormente anche la fase di *pruning*, cioè un processo di potatura della rete, in cui i nodi o le connessioni meno importanti vengono eliminati per ridurre ulteriormente la complessità del modello. Nelle KAN, il pruning può essere applicato eliminando i neuroni che hanno un impatto trascurabile sull'output finale, determinato dai punteggi di importanza di ingresso e uscita (vedi Figura 3.3).

I punteggi di importanza vengono calcolati utilizzando la magnitudine delle attivazioni delle funzioni univariate associate ai neuroni. Se un neurone ha un punteggio di importanza basso sia in ingresso che in uscita, viene considerato non essenziale e può essere eliminato senza influenzare negativamente le prestazioni della rete.

Infine, la *symbolification* è una tecnica che permette di trasformare le funzioni numeriche apprese dalla rete in funzioni simboliche. Questo è particolarmente utile per rendere il modello più interpretabile e comprensibile. Ad esempio, se durante l'addestramento una funzione univariata apprende un comportamento che può essere approssimato da una funzione matematica semplice, come sin(x) o log(x), è possibile sostituire la funzione numerica con la sua forma simbolica.

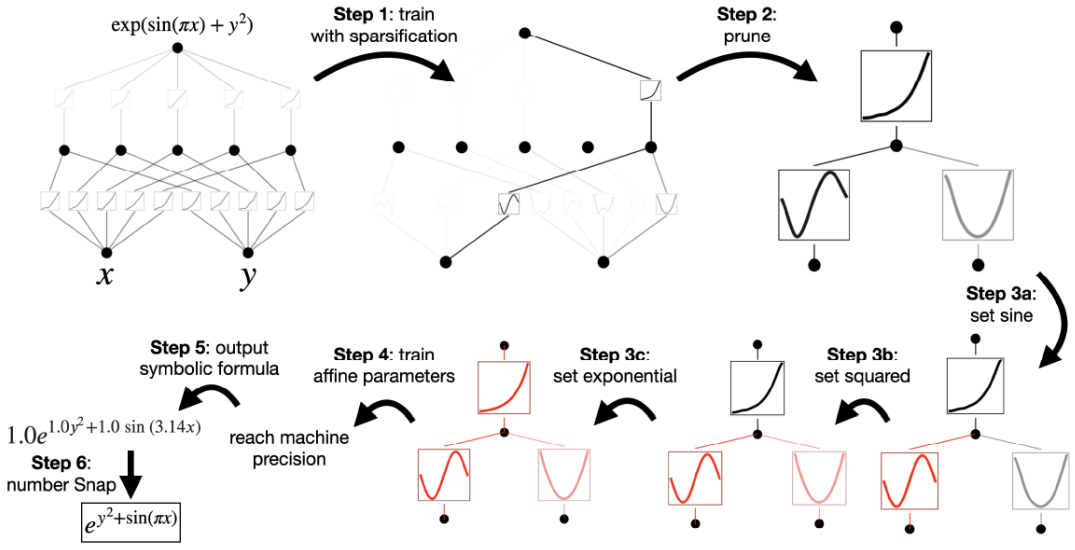



**Figura 3.3 – Step del Pruning del modello**

Questa trasformazione non solo migliora l'interpretebilità, ma può anche rendere la rete più efficiente dal punto di vista computazionale, poiché le funzioni simboliche sono spesso più veloci da calcolare rispetto alle loro controparti numeriche.

# 7. Legge di Scaling nelle Reti KAN

Una delle caratteristiche distintive delle reti KAN è la loro capacità di scalare in modo più efficiente rispetto alle MLP. In particolare, l'errore della rete KAN decresce più rapidamente con l'aumento del numero di parametri. Questo fenomeno è descritto dalla legge di scaling:

$$\ell \sim N^{-\alpha}$$

dove:

- $\ell$ rappresenta l'errore della rete (tipicamente misurato come errore quadratico medio, RMSE),
- N è il numero di parametri nella rete,
- $\alpha$ è un esponente di scaling che dipende dalla complessità della funzione target.

Questa legge mostra che le KAN sono in grado di ottenere una riduzione dell'errore più rapida rispetto alle MLP, con un numero inferiore di parametri. Questo

comportamento rende le KAN particolarmente vantaggiose in scenari in cui si desidera ottenere alte prestazioni con una complessità computazionale ridotta.

Nelle MLP, la riduzione dell'errore con l'aumento dei parametri è più lenta, poiché la rete necessita di un numero maggiore di neuroni e strati per rappresentare funzioni non lineari complesse. Le KAN, grazie alla decomposizione in funzioni univariate, riescono a migliorare le loro prestazioni in modo più efficiente.

# 8. Il processo del Continual Learning nelle Reti KAN

Un aspetto particolarmente interessante delle KAN è la loro capacità di gestire il *continual learning* (apprendimento continuo) in modo efficace, evitando il problema del *catastrophic forgetting* (oblio catastrofico), che affligge molte altre architetture neurali, come le MLP.

Nelle reti tradizionali, il catastrophic forgetting si verifica quando la rete, durante l'apprendimento di nuovi task, dimentica quanto appreso in precedenza. Questo accade perché i nuovi dati sovrascrivono i pesi e le attivazioni che erano stati appresi per i task precedenti. Questo fenomeno è particolarmente problematico nei contesti in cui i dati vengono presentati alla rete in modo sequenziale, come nell'apprendimento continuo.

Le KAN, grazie alla loro struttura basata su B-spline locali, riescono a preservare le informazioni apprese in passato. Quando nuovi dati vengono presentati alla rete, solo i coefficienti locali delle spline vengono aggiornati, lasciando intatte le altre parti della rete che non sono influenzate dai nuovi dati. Questo significa che la rete può apprendere nuove informazioni senza compromettere la conoscenza precedentemente acquisita.

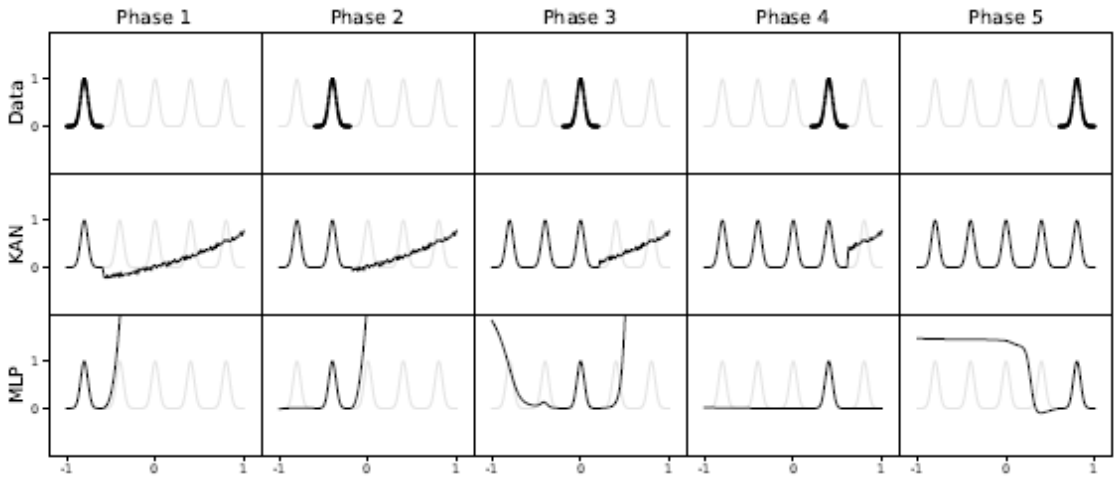



**Figura 3.4 Esempio di Continual Learning**

Questo approccio è reso possibile dalla località delle spline, che permette di aggiornare solo una parte della rete alla volta, mantenendo inalterati i coefficienti che descrivono le regioni del dominio che non vengono influenzate dai nuovi dati.

Un esempio di questo approccio si può osservare quando la rete viene addestrata su un task di regressione sequenziale con funzioni che variano nel tempo. Man mano che vengono introdotti nuovi task, le KAN possono adattarsi progressivamente senza dimenticare le soluzioni precedenti. Questo rende le KAN particolarmente adatte per applicazioni come la robotica o i sistemi di intelligenza artificiale adattativa, dove la capacità di apprendere continuamente nuove informazioni è fondamentale.

# 9. Risoluzione di Equazioni Differenziali Parziali (PDE) nelle Reti KAN

Le reti KAN si rivelano particolarmente efficaci nella risoluzione di equazioni differenziali parziali (PDE), un problema complesso che richiede la rappresentazione accurata di funzioni non lineari multivariate. Le equazioni differenziali parziali sono alla base di molti fenomeni fisici e trovano applicazioni in campi come la fluidodinamica, la termodinamica, e la modellazione dei sistemi dinamici.

Le KAN, grazie alla loro capacità di scomporre funzioni multivariate in funzioni univariate più semplici, riescono a rappresentare le soluzioni delle PDE in modo efficiente. L'uso delle B-spline permette di approssimare le soluzioni delle PDE con alta precisione, riducendo al contempo il numero di parametri necessari per ottenere una buona approssimazione.

Nella pratica, le reti KAN sono particolarmente utili per risolvere PDE non lineari, dove la complessità della soluzione rende difficile l'uso di metodi tradizionali. Le KAN offrono un modo per apprendere direttamente la soluzione della PDE utilizzando un approccio basato sui dati, riducendo il costo computazionale associato alla risoluzione numerica delle PDE.

# 10. Vantaggi e svantaggi delle reti KAN

Le reti KAN si caratterizzano per diversi vantaggi, tra i quali:

1. Efficienza Parametrica: Le KAN riescono a rappresentare funzioni multivariate complesse con un numero di parametri molto ridotto rispetto alle MLP. La decomposizione in funzioni univariate consente di ridurre drasticamente il numero di neuroni e strati necessari per rappresentare la funzione target.

2. Generalizzazione Migliorata: Grazie alla loro struttura, le KAN hanno una migliore capacità di generalizzazione rispetto alle MLP. La scomposizione in componenti univariate riduce il rischio di overfitting, poiché la rete non è costretta a imparare relazioni complesse tra tutte le variabili.

3. Efficienza Computazionale: Le KAN richiedono meno risorse computazionali rispetto alle MLP per ottenere la stessa precisione. Tecniche come la Grid Extension permettono di raffinare la rappresentazione della funzione senza dover riaddestrare completamente la rete.

4. Adattabilità al Continual Learning: Le KAN possono essere utilizzate in contesti di apprendimento continuo, dove i dati vengono presentati in modo sequenziale. La capacità di evitare il catastrophic forgetting rende le KAN particolarmente adatte per applicazioni che richiedono un apprendimento incrementale.

Tuttavia, si presentano anche delle limitazioni:

1. Implementazione Complessa: L'implementazione delle reti KAN richiede una parametrizzazione accurata delle spline e un'ottimizzazione efficace della griglia. La scelta della griglia iniziale e del grado delle spline può influenzare significativamente le prestazioni del modello.

2. Scalabilità Limitata: Sebbene le KAN siano particolarmente efficienti per problemi di bassa o media dimensionalità, la loro scalabilità verso problemi ad alta dimensionalità può essere una sfida. In alcuni casi, la necessità di estendere la griglia e ottimizzare un gran numero di coefficienti può comportare un aumento significativo del costo computazionale.

# CAPITOLO IV

# SVILUPPO E CONFRONTO DEI MODELLI KOLMOGOROV-ARNOLD (KAN) E MULTI-LAYER PERCEPTRON (MLP)

SOMMARIO**:** 1. Introduzione – 2. Un esempio pratico: Heart Disease – 3. Implementazione della rete KAN con 8 neuroni nello strato nascosto – 4. Strategia di confronto tra reti MLP e KAN – 5. Dataset utilizzati – 6. Risultati delle implementazioni.

# 1. Introduzione

Questo capitolo è dedicato all'implementazione delle reti neurali KAN (Kolmogorov-Arnold Network) e MLP (Multilayer Perceptron), con l'obiettivo di fornire una visione approfondita del loro funzionamento interno e dei meccanismi che le caratterizzano. In particolare, verrà posta maggiore enfasi sull'elevata accuratezza ottenuta dalle reti KAN rispetto alle MLP, evidenziando i vantaggi che le prime offrono in termini di performance e capacità di generalizzazione.

L'analisi condotta in questo lavoro è stata eseguita utilizzando il linguaggio di programmazione Python, ampiamente diffuso tra i ricercatori per l'apprendimento automatico e l'elaborazione dei dati. Le librerie principali utilizzate includono TensorFlow e PyTorch per la costruzione delle reti MLP, e il pacchetto PyKAN per l'implementazione delle reti KAN. Quest'ultimo è stato fornito dall'autore delle reti KAN ed è disponibile al seguente link GitHub:

https://github.com/KindXiaoming/pykan.

L'implementazione si articolerà in diversi scenari:

- Esempio applicativo iniziale: Si fornirà un esempio pratico dell'implementazione di entrambe le reti su uno dei dataset selezionati, esaminando gli output e le prestazioni fornite da ciascuna rete. Verrà messo in evidenza come le reti KAN raggiungano un'accuratezza superiore rispetto alle MLP, sottolineando i motivi di tale vantaggio.
- Confronto su sette dataset: Si confronteranno le classificazioni e le performance ottenute dai metodi KAN e MLP su sette dataset differenti, selezionati per analizzare l'efficacia di ciascun metodo di fronte a dati di input con diversa composizione e struttura. Attraverso questo confronto, si mostrerà come, in vari contesti, le reti KAN superino le MLP in termini di accuratezza e robustezza.

- Valutazione di interpretabilità ed efficienza computazionale: Si effettuerà un paragone tra l'interpretabilità e l'efficienza computazionale offerte dalla rete KAN rispetto all'approccio MLP, coinvolgendo dataset con caratteristiche diversificate per raggiungere risultati confrontabili. Saranno discussi i trade-off tra complessità del modello e prestazioni ottenute.

L'implementazione presentata in questo elaborato aspira al raggiungimento di diversi obiettivi:

- Dimostrare le superiori prestazioni delle reti KAN: Fornire al lettore una comprensione delle capacità delle reti KAN di raggiungere un'accuratezza superiore rispetto alle MLP, evidenziando i casi in cui le KAN mostrano un vantaggio significativo e analizzando le ragioni sottostanti a queste differenze.
- Validare l'affidabilità dei modelli: Verificare la capacità del modello KAN di superare le performance dell'MLP in termini di accuratezza di classificazione e importanza delle variabili, al fine di quantificare la sua affidabilità e mostrare il suo comportamento in diverse circostanze pratiche.
- Valutare punti di forza e debolezza: Analizzare le similitudini e le differenze tra i due metodi attraverso un approccio empirico, fornendo un criterio di scelta tra le due architetture di reti neurali basato sugli obiettivi dell'utente, sulla struttura dei dataset e sui punti di forza e di debolezza di ciascun metodo.

Inoltre, al fine di rendere i risultati riproducibili, nell'Appendice è possibile consultare il codice con cui è possibile replicare le analisi presentate di seguito, incluso l'utilizzo del pacchetto PyKAN disponibile su GitHub.

# 2. Un esempio pratico: Heart Disease

## 2.1. Descrizione del Dataset e delle Variabili

In questo paragrafo si presenta un'implementazione pratica delle reti neurali KAN sul dataset "Heart Disease" disponibile su Kaggle[25]. Il dataset è composto da 1.189 osservazioni e 11 caratteristiche, suddivise in 5 variabili numeriche e 6 categoriche. Il campione risulta bilanciato, con il 53% degli individui affetti da patologie cardiache.

L'obiettivo del modello predittivo sviluppato è fornire uno strumento di supporto per la diagnosi tempestiva di patologie cardiache, correlando specifiche variabili mediche alla presenza di tali condizioni. I predittori considerati sono i seguenti:

- Age: età dell'individuo;
- Sex: sesso (1 = maschio, 0 = femmina);
- Chest Pain Type: tipo di dolore toracico, una variabile categorica che può assumere i valori: angina tipica, angina atipica, dolore non correlato all'angina e asintomatico;
- Resting Blood Pressure: pressione arteriosa a riposo (in mm Hg);
- Cholesterol: livello di colesterolo nel sangue (in mg/dl);
- Fasting Blood Sugar: glicemia a digiuno > 120 mg/dl (1 = vero, 0 = falso);
- Resting ECG: risultati dell'elettrocardiogramma a riposo (normale, anormalità dell'onda ST-T, ipertrofia ventricolare sinistra);
- Max Heart Rate: frequenza cardiaca massima raggiunta durante l'esercizio;
- Exercise Induced Angina: angina indotta da esercizio fisico (1 = sì, 0 = no);

[25] www.kaggle.com

- Oldpeak: depressione del segmento ST indotta dall'esercizio rispetto al riposo;
- ST Slope: pendenza del segmento ST durante il picco dell'esercizio fisico (piatta, discendente, ascendente).

La variabile target è Heart Disease, che assume valore 1 se l'individuo è affetto da patologia cardiaca e 0 altrimenti.

## 2.2. Addestramento del Modello KAN

Considerate le limitate risorse computazionali a disposizione, è stata implementata una rete KAN con un singolo layer nascosto composto da 5 nodi (Figura 4.1). Nonostante questa restrizione, il modello ha ottenuto una percentuale di corretta classificazione dell'81% sul set di training e dell'77% sul set di test.

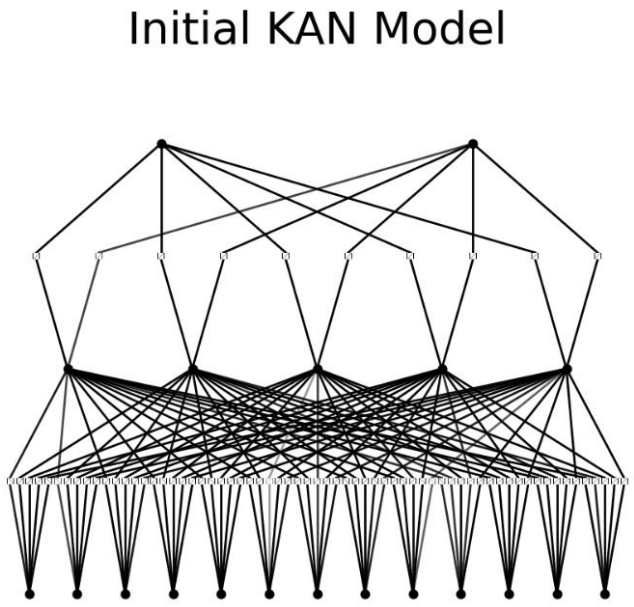



**Figura 4.1 – Modello KAN iniziallizzato**

### 2.2.1. Algoritmo di ottimizzazione

Per l'ottimizzazione dei pesi del modello è stato utilizzato l'algoritmo Limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS), un metodo di ottimizzazione quasi-Newtoniano particolarmente efficace per problemi di

apprendimento profondo con un numero moderato di parametri. LBFGS è noto per la sua capacità di convergere rapidamente verso un minimo locale della funzione di errore, grazie alla stima delle derivate utilizzando un numero limitato di iterazioni.

Motivazioni della Scelta di LBFGS:

- Convergenza Rapida: LBFGS è in grado di ridurre drasticamente il numero di iterazioni necessarie rispetto ai metodi di discesa del gradiente tradizionali.
- Ottimizzazione per Piccoli Batch: È particolarmente efficace quando si lavora con dataset di dimensioni medie e un numero limitato di epoche, come nel caso del nostro setup.
- Miglioramento della Stabilità: A differenza degli algoritmi di gradiente stocastico, LBFGS è meno suscettibile a fluttuazioni durante l'ottimizzazione, consentendo una stabilizzazione più rapida dei pesi.

### 2.2.2. Iperparametri utilizzati

Gli iperparametri del modello KAN sono stati mantenuti nelle impostazioni di default di LBFGS per garantire stabilità e velocità di convergenza. Gli iperparametri principali includono:

- Learning rate: Configurato secondo le impostazioni predefinite dell'algoritmo, che adatta automaticamente il passo di aggiornamento durante l'ottimizzazione.
- Tolleranza: Definita per controllare la precisione della convergenza; più il valore è basso, più è accurata la soluzione, ma a scapito del tempo di calcolo.

- Numero di Iterazioni: Determinato dinamicamente in base al criterio di convergenza dell'algoritmo LBFGS, che tende a fermarsi quando la riduzione dell'errore tra le iterazioni successive diventa minima.

## 2.3. Pruning del modello

Successivamente, è stata effettuata una potatura del modello al fine di semplificarlo e renderlo più interpretabile, eliminando tutti i collegamenti tra nodi aventi un peso inferiore a $1\times10^{-4}$. Questo processo, noto come pruning, ha permesso di ridurre la complessità della rete, mantenendo comunque prestazioni elevate. Dopo la potatura, il modello è stato riaddestrato per ottimizzare nuovamente i pesi rimanenti e assicurare che la struttura semplificata continuasse a fornire buone prestazioni predittive.

La figura sottostante illustra il modello dopo il pruning, evidenziando la riduzione delle connessioni più deboli.

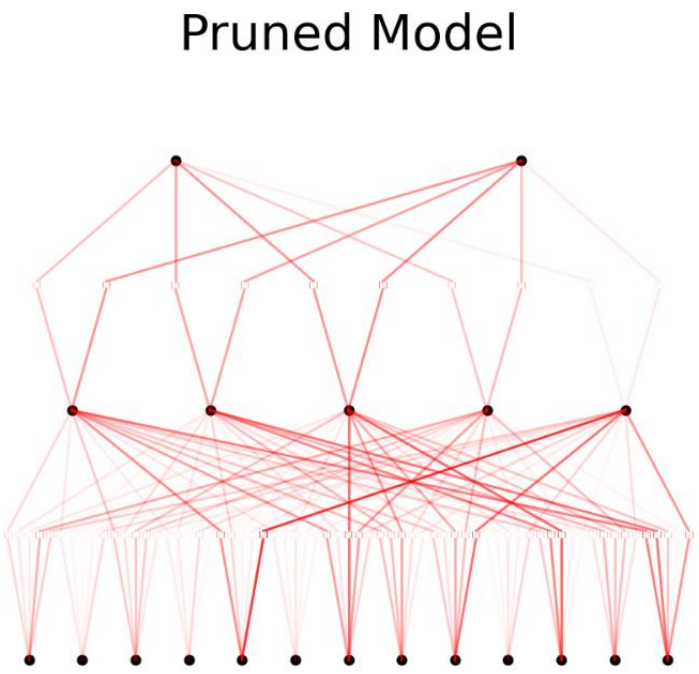



**Figura 4.2 – Modello KAN dopo la fase di Pruning**

Il pruning ha permesso di migliorare l'efficienza computazionale, riducendo i tempi di addestramento e rendendo il modello più robusto e meno incline all'overfitting.

## 2.4. Fix delle Funzioni di attivazione

Per rendere la rete maggiormente fluida e interpretabile, si è proceduto a fissare le funzioni spline utilizzate, sostituendo le funzioni di attivazione "create" con funzioni note e consolidate, quali: x, $x^3, x^4, exp, log, \sqrt{x}, tanh, \sin(x)$. Questo approccio ha permesso di generare formule simboliche che descrivono esplicitamente le relazioni apprese dalla rete.

## 2.5. Formula Simbolica

Le formule simboliche ottenute rappresentano una rappresentazione esplicita delle relazioni non lineari tra le variabili di input e l'output predetto dalla rete. In pratica, queste formule catturano la complessità del modello, evidenziando come le diverse variabili interagiscano per influenzare il risultato finale.

L'utilizzo di funzioni matematiche note, come sin, tanh, e tan, consente al modello di esprimere comportamenti complessi, quali oscillazioni e variazioni rapide, che riflettono le dinamiche sottostanti nei dati. Le formule simboliche permettono di comprendere quali trasformazioni sono applicate ai dati grezzi per produrre la previsione, rendendo il modello più trasparente e interpretabile.

Di seguito, le due formule simboliche ottenute dopo la sostituzione delle funzioni di attivazione:

- Formula 1:

```
formula1
```

$$1.83 \sin\Big(0.21(1 - 0.18x_{11})^3 + 0.08(1 - 0.46x_6)^4 - 0.16 \sin(1.39x_1 + 3.78) +$$

$$+ 0.16 \sin(0.8x_{10} - 8.58) + 0.16 \sin(0.6x_4 - 5.18) - 0.08 \sin(1.05x_8 - 6.37) +$$

$$+ 0.12 \tan(1.57x_3 - 0.42) - 0.09 \tanh(3.0x_5 + 2.98) - 0.05 |8.7x_7 + 0.03| + 7.74\Big) +$$

$$+ 0.2 \sin\Big(0.2(0.17 - x_7)^2 - 0.01(-0.63x_2 - 1)^4 - 0.02e^{3.0x_9} + 2.0 \sin(0.4x_1 + 8.0) +$$

$$+ 0.3 \sin(0.38x_4 - 9.8) - 1.33 \sin(0.59x_5 + 4.99) - 0.4 \tan(9.92x_6 + 0.8) +$$

$$+ 0.23 \tan(0.41x_8 - 6.18) + 0.75 \tanh(0.31x_{10} - 0.49) + 0.06 \tanh(10.0x_3 + 2.06) - 6.65 + 0.96e^{-0.64x_{11}}\Big) -$$

$$- 0.16 \sin\left(-1.06(0.47 - x_9)^4 + 0.4(1 - 0.25x_{11})^4 + 0.71(-0.7x_8 - 1)^2\right.$$

- Formula 2:

```
formula2
```

$$1.57 \sin\Big(0.23(1 - 0.18x_{11})^3 + 0.08(1 - 0.46x_6)^4 - 0.17 \sin(1.39x_1 + 3.78) +$$

$$+ 0.17 \sin(0.8x_{10} - 8.58) + 0.17 \sin(0.6x_4 - 5.18) - 0.09 \sin(1.05x_8 - 6.37) +$$

$$+ 0.13 \tan(1.57x_3 - 0.42) - 0.1 \tanh(3.0x_5 + 2.98) -$$

$$- 0.1 \tanh(3.0x_5 + 2.98) - 0.05 |8.7x_7 + 0.03| - 4.83\Big) +$$

$$+ 0.2 \sin\Big(0.2(0.17 - x_7)^2 - 0.01(-0.63x_2 - 1)^4 - 0.02e^{3.0x_9} +$$

$$+ 1.97 \sin(0.4x_1 + 8.0) + 0.3 \sin(0.38x_4 - 9.8) - 1.31 \sin(0.59x_5 + 4.99) -$$

$$- 0.4 \tan(9.92x_6 + 0.8) + 0.22 \tan(0.41x_8 - 6.18) + 0.73 \tanh(0.31x_{10} - 0.49) +$$

$$+ 0.06 \tanh(10.0x_3 + 2.06) - 6.57 + 0.95e^{-0.64x_{11}}\Big)$$

### 2.5.1. Interpretazione delle Formule Simboliche

Di seguito vengono esaminate alcune delle principali caratteristiche delle formule, evidenziando i meccanismi specifici attraverso cui queste interagiscono con le variabili d'ingresso per generare le previsioni del modello:

1. Relazioni Non Lineari e Complesse: Le formule esprimono come ogni variabile di input sia trasformata attraverso funzioni trigonometriche e iperboliche per modellare relazioni complesse. Ad esempio, termini come $\sin(0.21(1-0.18x_{11})^3)$ mostrano che esistono oscillazioni significative che

possono corrispondere a cambiamenti critici nei dati clinici, come variazioni nei risultati dell'ECG o nella pressione sanguigna.

2. Coefficiente di Peso: I coefficienti numerici accanto a ciascuna funzione (1.83, 0.08, etc.) determinano l'importanza relativa di ciascun termine nel contributo complessivo all'output. Un peso più elevato implica che quel particolare aspetto ha un impatto maggiore sulla previsione finale del modello.

3. Oscillazioni e Pendenze Variabili: Le funzioni sinusoidali e tangenziali (sin, tan) indicano che ci sono effetti ciclici e periodici che il modello riconosce come importanti per la previsione delle patologie cardiache. Questo può essere associato, ad esempio, a variabili come la frequenza cardiaca massima, che può avere un comportamento oscillatorio in risposta a diversi fattori.

4. Effetto di Potenza e Radice: I termini $(1-0.46x_6)^4$ e $|8.72x_7+0.03|$ mostrano come le variabili vengano potenziate o trattate con funzioni di valore assoluto per migliorare la rappresentazione delle relazioni sottostanti.

## 2.6. Grid Extension

Per migliorare ulteriormente le prestazioni del modello, si è sfruttata la proprietà delle B-spline estendendo la griglia da un valore iniziale di 3 a valori più fini di 5, 10, 20, 50 e 100. L'estensione della griglia ha portato a una riduzione dell'errore sia sul training set che sul test set (Figura 4.3), fino a raggiungere una griglia di dimensione 100. Con questa configurazione ottimizzata, si è ottenuta una percentuale di corretta classificazione del 86% sul training set e dell'83% sul test set.

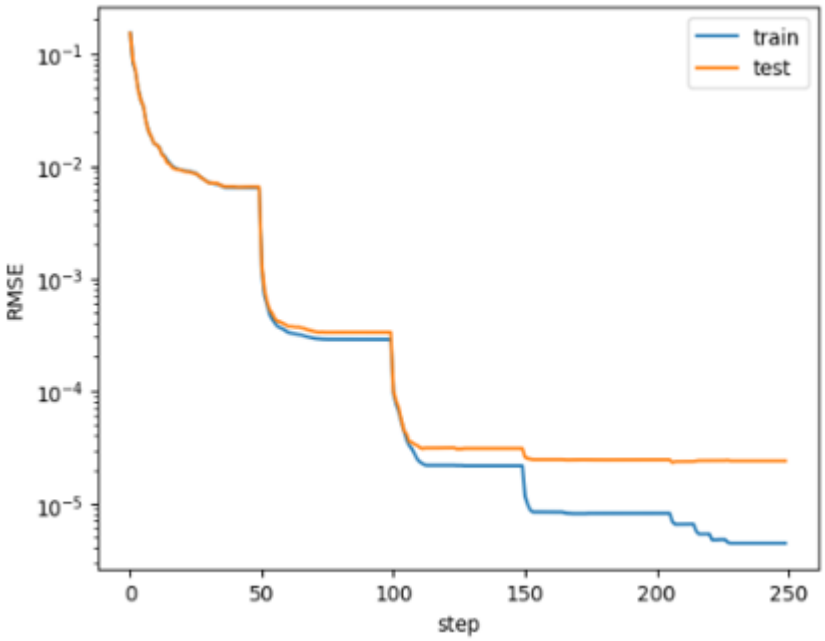



**Figura 4.3 – Espansione della griglia con riduzione dell'errore**

## 2.7. Analisi delle Caratteristiche Influenti

Per identificare le caratteristiche che influenzano maggiormente l'addestramento della rete, è stata condotta un'analisi tramite permutazione delle feature. Questa tecnica ha permesso di valutare l'importanza relativa di ciascuna variabile all'interno del modello. I risultati dell'analisi sono riportati nella tabella seguente:

| Rank | Feature | Importanza |
|---|---|---|
| 1 | Feature 2 | 0.1303 |
| 2 | Feature 4 | 0.1218 |
| 3 | Feature 10 | 0.1218 |
| 4 | Feature 3 | 0.0882 |
| 5 | Feature 6 | 0.0714 |

| Rank | Feature | Importanza |
|---|---|---|
| 6 | Feature 9 | 0.0714 |
| 7 | Feature 7 | 0.0672 |
| 8 | Feature 0 | 0.0630 |
| 9 | Feature 1 | 0.0588 |
| 10 | Feature 8 | 0.0588 |
| 11 | Feature 5 | 0.0504 |

**Tabella 4.1 – Permutation Importance in KAN**

Le tre caratteristiche più influenti risultano essere:

1. Feature 2 (Chest Pain Type): con un'importanza di 0.1303;
2. Feature 4 (Cholesterol): con un'importanza di 0.1218;
3. Feature 10 (ST Slope): anch'essa con un'importanza di 0.1218.

Questa analisi evidenzia come le variabili relative al tipo di dolore toracico, al livello di colesterolo e alla pendenza del segmento ST siano determinanti nella previsione delle patologie cardiache.

## 2.8. Implementazione della Rete MLP sul Dataset "Heart Disease"

Dopo aver valutato le prestazioni della rete KAN, è stata implementata una rete Multi-Layer Perceptron (MLP) sullo stesso dataset "Heart Disease" per poter confrontare le due architetture e determinare quale delle due offra le migliori capacità predittive. La scelta di utilizzare le stesse variabili impiegate dalla rete KAN consente un confronto diretto delle prestazioni, mettendo in luce le differenze tra i due approcci in termini di accuratezza e interpretabilità.

### 2.8.1. Ricerca della Struttura Ottimale tramite Trial and Error

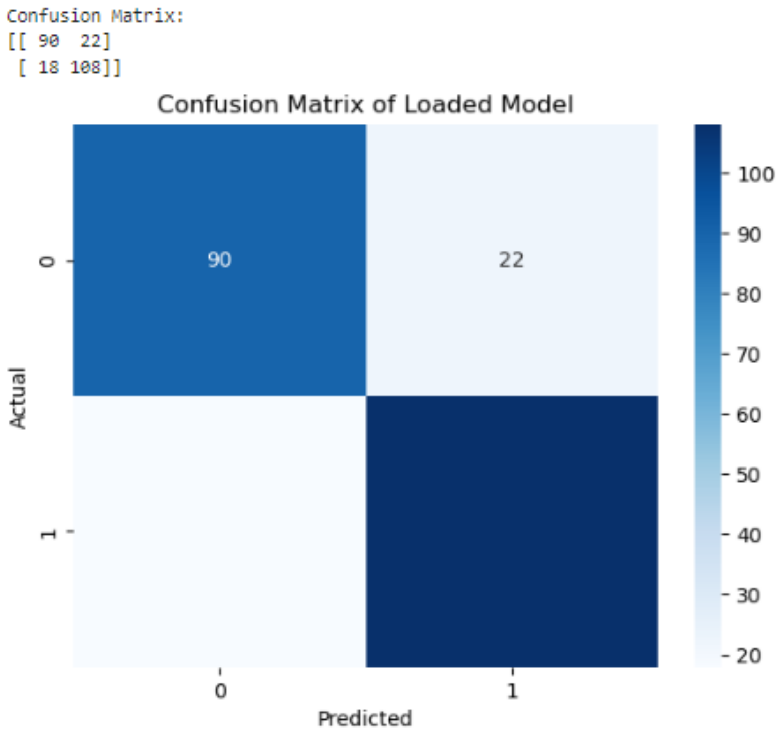
Per identificare la configurazione più efficiente della rete MLP, è stato adottato un approccio empirico di trial and error. Questo metodo ha previsto il test di diverse combinazioni di layer nascosti, numero di neuroni per layer e funzioni di attivazione. La sperimentazione ha permesso di esplorare una vasta gamma di architetture, calibrando il modello per trovare il miglior equilibrio tra complessità e accuratezza, evitando al contempo il rischio di overfitting.

- Strutture testate: Sono state esplorate configurazioni che variavano da reti semplici con un singolo layer nascosto e pochi neuroni, a strutture più complesse con due layer nascosti. Le architetture testate includevano:
    - Configurazioni con un solo layer nascosto, variando il numero di neuroni tra 5 e 64, come ad esempio [5], [10], [20], [30], [64].
    - Configurazioni con due layer nascosti, combinando un numero variabile di neuroni, ad esempio [5, 5], [10, 5], [20, 10], fino a strutture più elaborate come [64, 32]. Questa ampia gamma di strutture ha permesso di esplorare come la complessità della rete influenzasse le sue prestazioni.
- Funzioni di attivazione testate: Per ogni configurazione di layer nascosti, sono state sperimentate diverse funzioni di attivazione per determinare quale fosse la più efficace nel catturare le relazioni non lineari tra le variabili. Le funzioni di attivazione testate includevano:
    - ReLU, Sigmoid, Tanh, LeakyReLU, ELU, SELU, Softplus, Softsign, ReLU6, GELU, Hardtanh, LogSigmoid. L'obiettivo era identificare la funzione di attivazione che ottimizzasse il processo di apprendimento, migliorando la capacità della rete di adattarsi ai dati.

### 2.8.2. Struttura e Funzione di Attivazione Ottimale Selezionata

Dopo un'analisi approfondita dei risultati ottenuti da ciascuna configurazione testata, è stata selezionata una rete MLP caratterizzata da un layer nascosto composto da 8 neuroni e con la funzione di attivazione ReLU6. Questa configurazione è stata scelta perché rappresenta un compromesso ideale tra la capacità del modello di catturare relazioni complesse e il rischio di overfitting. La funzione di attivazione ReLU6, in particolare, limita l'output massimo a 6, prevenendo l'esplosione dei valori durante l'addestramento e contribuendo a stabilizzare il processo di ottimizzazione.

- Accuratezza del modello: La rete MLP selezionata ha raggiunto un'accuratezza dell'83%, un risultato solido che si posiziona in modo competitivo rispetto alla rete KAN, dimostrando che anche una struttura relativamente semplice può fornire prestazioni predittive efficaci.
- Matrice di Confusione: Per comprendere meglio le prestazioni del modello, è stata utilizzata la matrice di confusione, che illustra la capacità della rete MLP di distinguere correttamente tra individui con e senza patologie cardiache:




Questa matrice mostra che il modello ha correttamente identificato 90 individui sani e 108 affetti da patologie cardiache, commettendo 22 falsi positivi e 18 falsi negativi. Questi risultati suggeriscono che il modello MLP ha una buona capacità di generalizzare sui dati di test, pur mostrando qualche difficoltà nel classificare perfettamente alcuni individui.

### 2.8.3. Analisi dell'Importanza delle Caratteristiche tramite Permutazione

Come per la rete KAN, è stata condotta un'analisi dell'importanza delle caratteristiche tramite la tecnica della permutazione delle feature. Questo approccio ha permesso di valutare l'impatto di ciascuna variabile sull'addestramento del modello, fornendo una visione dettagliata di quali caratteristiche abbiano influito maggiormente sulle prestazioni della rete MLP.

| **Importanza delle Caratteristiche (Permutation Importance)**: | |
|---|---|
| **ST slope**: | 0.0546 |
| **Cholesterol**: | 0.0210 |
| **Age**: | 0.0210 |
| **Exercise angina**: | 0.0168 |
| **Chest pain type**: | 0.0168 |
| **Sex**: | 0.0168 |
| **Max heart rate**: | 0.0126 |
| **Fasting blood sugar**: | 0.0126 |
| **Oldpeak**: | 0.0042 |
| **Resting ECG**: | 0.0000 |
| **Resting BP S**: | 0.0000 |

**Tabella 4.2 - Permutation Importance in MLP**

Questa analisi ha evidenziato che le variabili ST slope e cholesterol sono le più influenti per il modello MLP, suggerendo che queste caratteristiche hanno un ruolo chiave nella predizione della presenza di patologie cardiache. Altri fattori rilevanti includono age, exercise angina, e chest pain type, mentre variabili come resting ECG e resting BP S non sembrano influire significativamente sulle prestazioni del modello. Questo potrebbe indicare che, per la MLP, alcune variabili considerate rilevanti da altri approcci, come KAN, potrebbero avere un peso minore.

## 2.9. Confronto delle Prestazioni tra Rete KAN e MLP

Nel confronto tra la rete KAN e la MLP, è emerso un risultato particolarmente interessante: entrambe le reti hanno ottenuto prestazioni simili in termini di accuratezza, nonostante la rete KAN utilizzi una struttura più semplice con un minor numero di neuroni nel layer nascosto. In particolare, la rete KAN ha raggiunto un'accuratezza comparabile con quella della MLP utilizzando solo 5 neuroni nel suo layer nascosto, rispetto agli 8 neuroni impiegati dalla MLP.

Questa differenza nella struttura implica che la rete KAN ha dovuto stimare un numero inferiore di parametri rispetto alla MLP. In altre parole, la KAN ha raggiunto risultati equivalenti utilizzando una configurazione più parsimoniosa e meno complessa. Questo rappresenta uno dei punti di forza fondamentali delle reti KAN, come evidenziato nella letteratura scientifica: la capacità di ottenere elevate prestazioni predittive con un'architettura più snella, riducendo il numero di parametri da stimare e, di conseguenza, semplificando il modello complessivo.

Dal punto di vista dei tempi di addestramento, entrambe le reti hanno mostrato durate simili, grazie alla struttura efficiente della rete KAN. Tuttavia, il vantaggio della KAN risiede nel fatto che, con meno parametri da ottimizzare, il processo di addestramento non solo è più leggero, ma anche potenzialmente più stabile, riducendo il rischio di overfitting rispetto a reti più complesse.

Alla luce di questi risultati, emerge l'importanza di valutare non solo l'accuratezza finale di un modello, ma anche l'efficienza con cui tale accuratezza viene raggiunta. Le reti KAN, grazie alla loro capacità di ridurre la complessità del modello mantenendo elevate prestazioni, rappresentano un'opzione particolarmente vantaggiosa, soprattutto in contesti in cui le risorse computazionali sono limitate o si richiede un'elevata interpretabilità del modello.

Per un confronto ancora più approfondito e per valutare l'impatto di una struttura più complessa sulla rete KAN, verrà ora implementata una nuova versione della rete KAN che replica esattamente la struttura della MLP ottimale, con un layer nascosto composto da 8 neuroni. Questo adattamento permetterà di confrontare le due architetture su una base ancora più equa, analizzando non solo le differenze in termini di accuratezza, ma anche le dinamiche di addestramento e la gestione dei parametri, al fine di trarre conclusioni più solide sulle potenzialità delle reti KAN rispetto alle tradizionali MLP.

## 3. Implementazione della Rete KAN con 8 Neuroni nello Strato Nascosto

Dopo aver confrontato le prestazioni tra la rete KAN e la MLP utilizzando strutture diverse, è stato deciso di implementare una nuova versione della rete KAN, allineando la sua struttura a quella della MLP selezionata come ottimale. In particolare, è stato introdotto un layer nascosto composto da 8 neuroni, replicando così la configurazione della MLP per consentire un confronto diretto delle due architetture in condizioni analoghe.

L'implementazione della rete KAN con 8 neuroni ha seguito lo stesso approccio metodologico precedentemente descritto per la rete KAN originale. Il modello è stato addestrato con l'algoritmo di ottimizzazione LBFGS, mantenendo gli stessi criteri di valutazione e le tecniche di gestione delle funzioni di attivazione. Il processo di addestramento ha portato a risultati particolarmente positivi, con un'accuratezza finale del 93,17% sul set di training e dell'88,66% sul set di test. Questi risultati sottolineano l'efficacia della rete KAN nel mantenere alte prestazioni predittive anche con una struttura più complessa, dimostrando la sua

capacità di adattarsi a configurazioni maggiormente articolate senza compromettere la sua robustezza.

La matrice di confusione sottostante evidenzia la capacità del modello di distinguere correttamente tra individui sani e affetti da patologie cardiache:

Confusion Matrix:

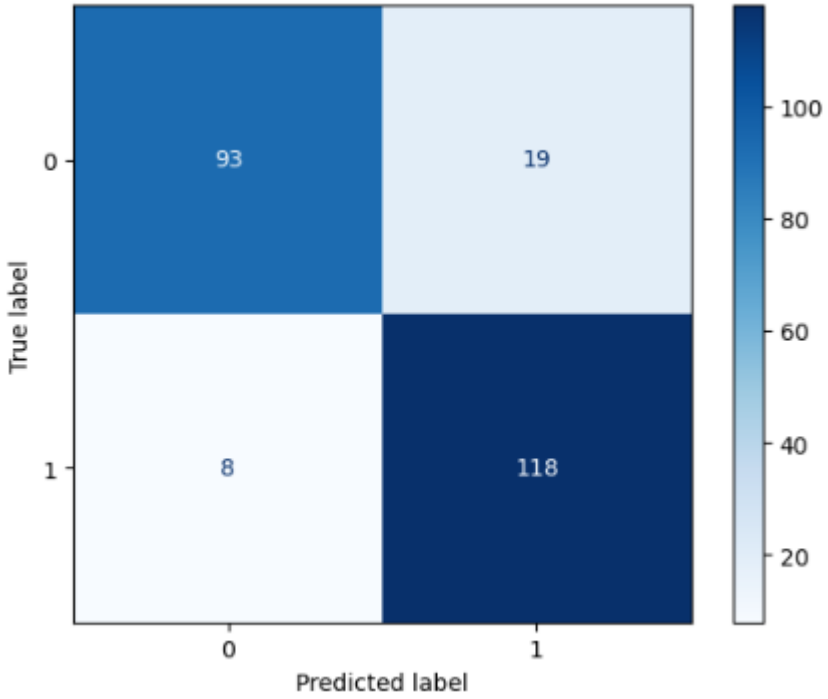



**Figura 4.3 – Matrice di Confusione**

La matrice mostra che la rete KAN ha classificato correttamente 93 individui sani e 118 con patologie cardiache, commettendo 19 falsi positivi e 8 falsi negativi. Questi risultati indicano una gestione efficace delle classi bilanciate, migliorando ulteriormente rispetto alla versione precedente della rete KAN, in particolare per quanto riguarda la riduzione degli errori nel set di test.

## 3.1. Valutazione della Grid Extension

Contrariamente a quanto osservato nella precedente implementazione della rete KAN, in questo caso l'applicazione della tecnica della Grid Extension si è rivelata meno vantaggiosa. Nella configurazione originale, l'estensione della griglia delle B-spline aveva permesso di affinare ulteriormente le prestazioni del modello, migliorando l'accuratezza sia nel training che nel test. Tuttavia, con l'architettura a 8 neuroni, il modello ha già raggiunto un livello di accuratezza elevato, rendendo marginali i benefici derivanti dall'ulteriore raffinamento della griglia (Figura 4.4).

Prove successive di estensione della griglia hanno infatti mostrato un miglioramento delle prestazioni esclusivamente sul training set, mentre l'accuratezza sul test set non ha evidenziato significativi progressi.

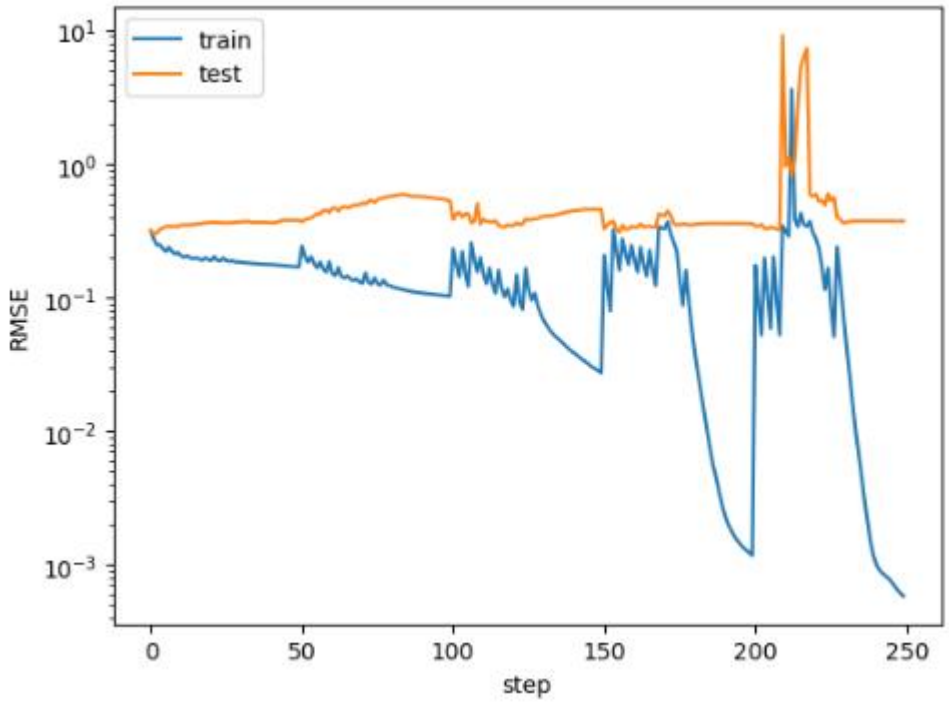



**Figura 4.4 – Estensione griglia (modello con 8 neuroni)**

Questo fenomeno suggerisce un inizio di overfitting, poiché il modello tendeva ad adattarsi troppo ai dati di training senza migliorare la sua capacità di

generalizzazione sui dati non visti. Di conseguenza, si è deciso di non estendere ulteriormente la griglia oltre i parametri standard, per evitare un deterioramento della capacità predittiva sul test set e preservare l'equilibrio tra accuratezza e robustezza del modello.

## 3.2. Analisi dell'Importanza delle Caratteristiche

Per quanto riguarda l'analisi dell'importanza delle caratteristiche, i risultati ottenuti sono rimasti invariati rispetto alla versione precedente della rete KAN. Questo indica che l'aumento del numero di neuroni non ha alterato significativamente le relazioni interne tra le variabili di input e l'output predetto dal modello. Le caratteristiche principali che influenzano la rete KAN continuano a essere le stesse, confermando la stabilità del modello rispetto alle modifiche strutturali apportate.

# 4. Strategia di Confronto tra Reti MLP e KAN

Per facilitare il confronto tra le reti MLP e KAN e garantire un'analisi rigorosa e consistente, da questo punto in avanti verrà adottata una metodologia standardizzata su tutti i dataset esaminati. In particolare, per ciascun dataset, verrà inizialmente implementata una rete MLP, ricercando la struttura ottimale tramite un approccio empirico di trial and error. Questo processo consentirà di individuare la configurazione che rappresenta il miglior compromesso tra complessità del modello e prestazioni predittive.

Una volta identificata la struttura ottimale della MLP, la stessa configurazione verrà adottata per la rete KAN, replicando il numero di layer nascosti e di neuroni, al fine di rendere il confronto il più equo possibile. Questo approccio permetterà di valutare direttamente le capacità di ciascuna rete nell'adattarsi ai diversi dataset,

analizzando le prestazioni in termini di accuratezza, efficienza di addestramento e interpretabilità dei modelli.

Questa metodologia assicura che le differenze nelle prestazioni non siano influenzate dalla struttura del modello, ma riflettano le effettive potenzialità delle due architetture. Il confronto diretto tra MLP e KAN su strutture equivalenti offrirà una panoramica chiara dei punti di forza e delle debolezze di ciascun approccio, permettendo di trarre conclusioni basate su dati empirici circa la loro applicabilità in vari contesti.

# 5. Dataset Utilizzati

I dataset utilizzati nell'analisi sono stati estratti dalle piattaforme Kaggle e UCI[26] Machine Learning Repository, note comunità online dedicate ai data scientist e ai professionisti dell'apprendimento automatico, dove vengono condivise matrici di dati utili per lo sviluppo e il test di algoritmi innovativi. I percorsi specifici per il reperimento delle singole matrici sono indicati all'interno del codice in Appendice.

Per valutare e confrontare le prestazioni delle reti neurali Multi-Layer Perceptron (MLP) e Kolmogorov-Arnold Networks (KAN), sono stati selezionati sette dataset con obiettivi di classificazione, sia binaria che multiclasse. La scelta dei dataset è stata guidata dalla necessità di testare i modelli su differenti tipologie di dati per esplorare la loro capacità di adattamento in diversi contesti.

I dataset differiscono in termini di numero di osservazioni, bilanciamento delle classi e composizione delle variabili di input (numeriche e categoriche). Questa diversificazione consente di testare la robustezza e la flessibilità delle reti MLP e

[26] https://archive.ics.uci.edu/

KAN nell'adattarsi a diverse strutture dei dati, valutando l'accuratezza predittiva e l'efficienza dell'addestramento in presenza di caratteristiche eterogenee.

**Tabella (4.1): descrizione dataset utilizzati.**

| Dataset | n | % classi | % var. numeriche | % var. categoriche |
|---|---|---|---|---|
| Heart Failure | 5000 | 69% Survived 31% Failure | 54% | 46% |
| Breast Cancer | 569 | 63% Benign 37% Malign | 100% | 0% |
| Iris | 150 | 33% Setosa 33% Versicolor 33%Virginica | 100% | 0% |
| Bank Marketing | 4521 | 88% no 12% yes | 37.50% | 62.50% |
| Heart Disease | 1190 | 47% Normal 53% Heart Disease | 41.67% | 58.33% |
| Fetal Health | 2126 | 78% Normal 14% Suspicious 8% Pathological | 100% | 0% |
| Cervical Cancer | 858 | 94% Healthy 6% Cancer | 61.54% | 38.46% |

Nella Tabella (4.1) sono riportate le caratteristiche dettagliate dei dataset utilizzati, evidenziando il numero di osservazioni, la percentuale delle classi nella variabile target, e la distribuzione delle variabili numeriche e categoriche.

## 5.1. Motivazioni della Selezione dei Dataset

L'utilizzo di dataset con diverse caratteristiche permette di evidenziare le potenzialità e i limiti delle reti KAN rispetto alle MLP. In particolare:

- Bilanciamento delle Classi: Alcuni dataset, come "Heart Failure" e "Breast Cancer", presentano classi bilanciate, mentre altri, come "Bank Marketing", sono caratterizzati da una forte asimmetria nelle classi della variabile target. Questo consente di osservare come le reti KAN e MLP si comportino in contesti di bilanciamento e sbilanciamento delle classi.
- Natura delle Variabili: La presenza di variabili numeriche e categoriche nei dataset consente di valutare l'abilità delle reti nel gestire diverse tipologie di dati. Ad esempio, i dataset "Fetal Health" e "Iris" sono costituiti esclusivamente da variabili numeriche, mentre altri, come "Heart Disease", contengono una combinazione di variabili numeriche e categoriche, mettendo alla prova la capacità dei modelli di integrare e analizzare informazioni eterogenee.

- Dimensione del Dataset: La variabilità nel numero di osservazioni, che spazia da dataset di grandi dimensioni come "Heart Failure" (5.000 osservazioni) a dataset più contenuti come "Iris" (150 osservazioni), permette di testare l'efficienza di addestramento delle reti e di analizzare come la dimensione del dataset influisca sulle prestazioni del modello.

Di seguito, si analizzeranno su questi dati i risultati delle due tipologie di reti neurali.

## 5.2. Performance Predittiva delle Reti KAN e MLP sui Dataset

L'implementazione delle reti KAN e MLP è stata effettuata sui sette dataset precedentemente descritti, con l'obiettivo di valutare la loro capacità predittiva in differenti contesti di classificazione, sia binaria che multiclasse. Per ogni dataset, è stata adottata una metodologia che prevedeva dapprima l'individuazione della struttura ottimale per le reti MLP attraverso un processo di trial and error, per poi applicare la medesima architettura alle reti KAN, al fine di ottenere un confronto equo e completo tra le due reti.

### 5.2.1. Metodologia di Addestramento e Validazione

I dataset sono stati suddivisi in training set e test set, con una proporzione dell'80% per l'addestramento e del 20% per la validazione, a eccezione dei dataset "Breast Cancer" e "Cervical Cancer", per i quali la proporzione è stata modificata al 70% per l'addestramento, al fine di ottimizzare le prestazioni e gestire la presenza di classi sbilanciate. Questa suddivisione ha permesso di stabilizzare i modelli e migliorare la capacità di generalizzazione dei risultati.

Per la valutazione delle prestazioni predittive delle reti, sono state calcolate diverse metriche di errore sia sui dati di addestramento che sui dati di validazione. Un'importante misura di valutazione è stata la matrice di confusione, che mostra la relazione tra le previsioni effettuate dal modello e le classificazioni reali presenti

nel dataset. Di seguito si presenta un esempio di matrice di confusione ottenuta per il dataset "Heart Failure" con la rete KAN.

Matrice di Confusion:

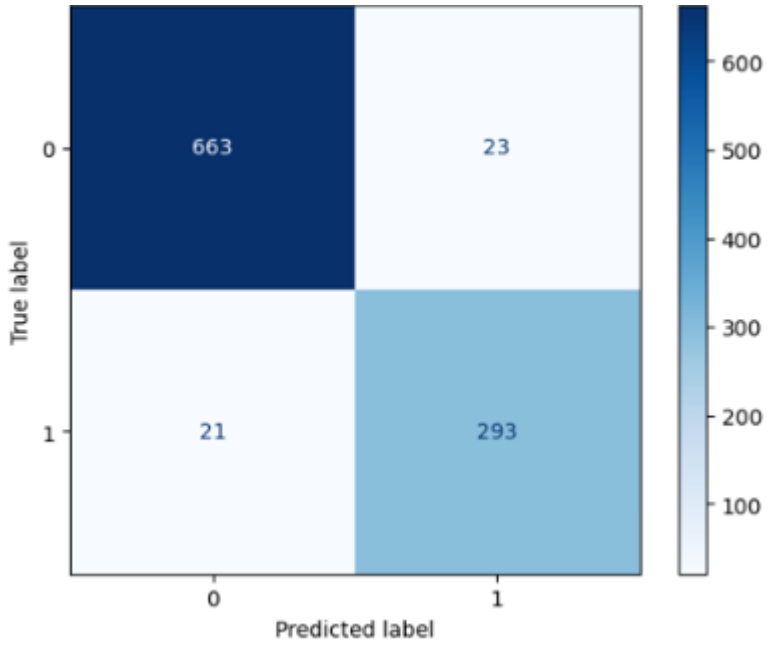



**Figura (4.2): Matrice di confusione del dataset Heart Failure con il modello KAN.**

In Figura (4.2) si può osservare la matrice di confusione generata dal modello KAN implementato sul dataset "Heart Failure". L'obiettivo di questo modello è quello di classificare correttamente gli individui colpiti da arresto cardiaco in base alle caratteristiche cliniche fornite. La matrice fornisce un'idea precisa delle capacità predittive del modello, mostrando il numero di previsioni corrette e gli errori di classificazione per ciascuna classe.

L'accuratezza del modello KAN per il dataset Heart Failure è stata calcolata come:

[(663+293)\1000]×100=95,60%

Questa percentuale indica che circa il 96% delle osservazioni sono state correttamente classificate. Tuttavia, è importante notare che, in presenza di classi

sbilanciate, l'accuratezza complessiva potrebbe non fornire una visione completa delle prestazioni del modello, motivo per cui è utile analizzare anche il tasso di errore per singola classe.

Il tasso di errore per la classe "Heart Failure" è stato calcolato come:

(21\314)×100=6,68%

Questo valore indica che circa il 7% degli individui con arresto cardiaco sono stati erroneamente classificati come sani. Il tasso di errore per le singole classi è un indicatore importante per valutare le difficoltà del modello nella gestione delle classi minoritarie.

# 6. Risultati delle Implementazioni

## Tabella 4.2:

### a) risultati implementazione reti MLP

| Dataset | Errori Classi (Test) | Accuratezza (Training) | Accuratezza (Test) |
|---|---|---|---|
| Heart Failure | 6,56% Normal - 38,36% Failure | 81,50% | 79,45% |
| Breast Cancer | 1,38% Benign - 13,28% Malign | 97,32% | 93,02% |
| Iris | 0 %Setosa - 70% Verticolor - 0% Virginica | 86,12% | 76,92% |
| Bank Marketing | 5,74% No - 51,91% Yes | 86,96% | 72,49% |
| Heart Disease | 19,64% Normal - 14,28% Heart Disease | 83,19% | 78,92% |
| Featal Health | 4,82% Normal - 40,68% Suspicious - 76,2% Pathological | 82,63% | 77,13% |
| Cervical Cancer | 0% Healthy - 100% Cancer | 93,60% | 85,22% |

### b) risultati implementazione reti KAN

| Dataset | Errori Classi (Test) | Accuratezza (Training) | Accuratezza (Test) |
|---|---|---|---|
| Heart Failure | 3,35% Normal - 6,68% Failure | 96,97% | 95,60% |
| Breast Cancer | 0% Benign - 9,52% Malign | 99,12% | 96,49% |
| Iris | 0 %Setosa - 5% Verticolor - 5% Virginica | 100% | 93,33% |
| Bank Marketing | 4,24% No - 30,76% Yes | 91,15% | 88,29% |
| Heart Disease | 16,96% Normal - 6,34% Heart Disease | 93,17% | 88,66% |
| Featal Health | 5,42% Normal - 25,4% Suspicious - 31,43% Pathological | 93,12% | 89,67% |
| Cervical Cancer | 0% - Healty - 92,3% Cancer | 94,33% | 87,46% |

Le Tabelle (4.2 a & b) riportano i risultati delle performance di classificazione ottenute dalle reti MLP e KAN sui sette dataset analizzati. I risultati dimostrano che entrambe le reti sono state in grado di raggiungere alte performance, ma con alcune differenze significative.

**Prestazioni delle Reti MLP:**

- Le reti MLP hanno ottenuto accuratezze elevate, generalmente comprese tra il 75% e il 90%, sia sul training set che sul test set. Tuttavia, si sono osservati problemi con le classi minoritarie, specialmente nei dataset sbilanciati come "Bank Marketing" e "Cervical Cancer", dove il tasso di errore per le classi meno rappresentate ha superato il 50% e il 100%, rispettivamente.
- Le prestazioni delle MLP tendono a essere influenzate da problemi di overfitting su dataset particolarmente sbilanciati, con una perdita di

accuratezza sul test set rispetto al training set. Ad esempio, nel dataset "Heart Disease", l'accuratezza scende al 78,92% nel test set, segnalando difficoltà nel mantenere una buona generalizzazione.

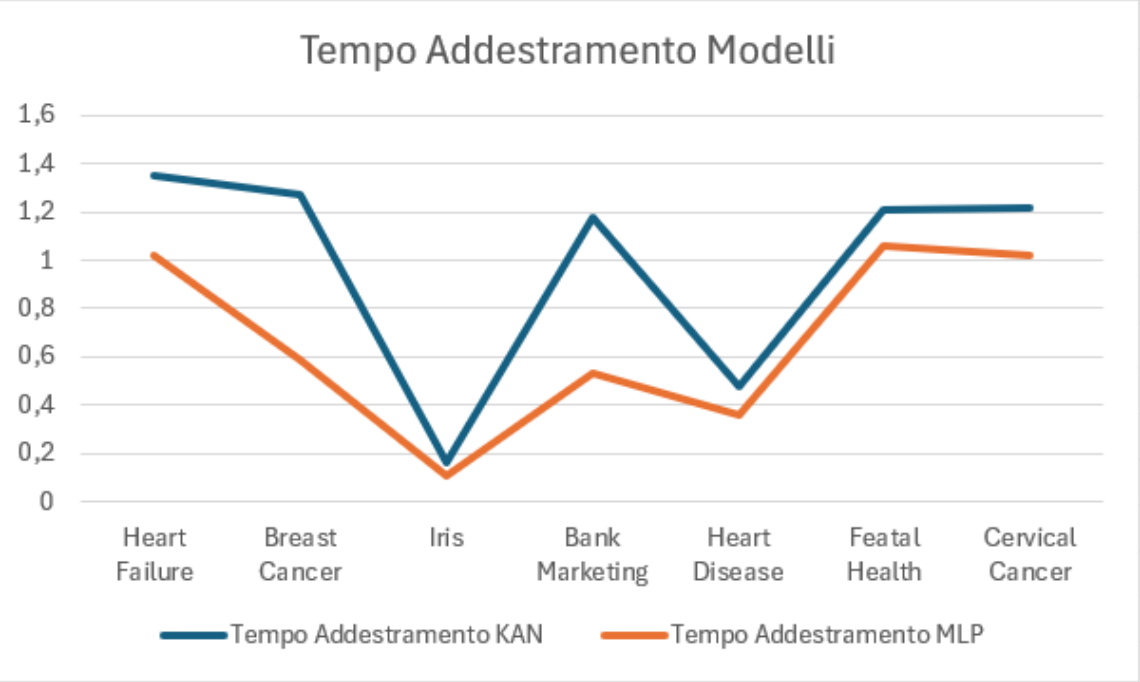
**Prestazioni delle Reti KAN:**

- Le reti KAN hanno dimostrato prestazioni eccellenti, con un'accuratezza media generalmente più alta rispetto alle MLP. In quasi tutti i dataset, le KAN hanno raggiunto un'accuratezza superiore al 90% nel test set, con riduzioni meno marcate rispetto al training set, il che indica una maggiore robustezza nella generalizzazione.
- Un notevole vantaggio delle KAN è stata la **riduzione degli errori di classificazione per le classi minoritarie**. Ad esempio, nel dataset "Bank Marketing",il quale presentava la variabile target sbilanciata, il tasso di errore per la classe è stato del 30%, contro il 52% ottenuto dalle MLP, evidenziando la capacità delle KAN di gestire meglio le situazioni di squilibrio tra le classi.

## 6.1. Confronto tra le Reti KAN e MLP: Tempo di Addestramento, Numero di Parametri e Accuratezza

Il confronto tra le reti KAN e MLP, condotto sui sette dataset precedentemente analizzati, evidenzia differenze significative su tre aspetti fondamentali: tempo di addestramento, numero di parametri e accuratezza. Questi tre fattori permettono di comprendere meglio le caratteristiche distintive delle due architetture e di valutarne l'applicabilità in diversi contesti.

### 6.1.1. Tempo di Addestramento




Uno dei principali punti di distinzione tra i modelli è il tempo di addestramento. Le KAN richiedono generalmente un tempo di addestramento superiore rispetto alle MLP. Questo aumento dei tempi è dovuto alla complessità computazionale delle KAN, in particolare per la gestione delle funzioni B-spline e delle trasformazioni non lineari. Anche se le MLP risultano più rapide nel convergere, le KAN, grazie alla loro capacità di catturare relazioni più complesse tra le variabili di input, offrono un vantaggio in termini di accuratezza e generalizzazione, giustificando il maggiore tempo di elaborazione.

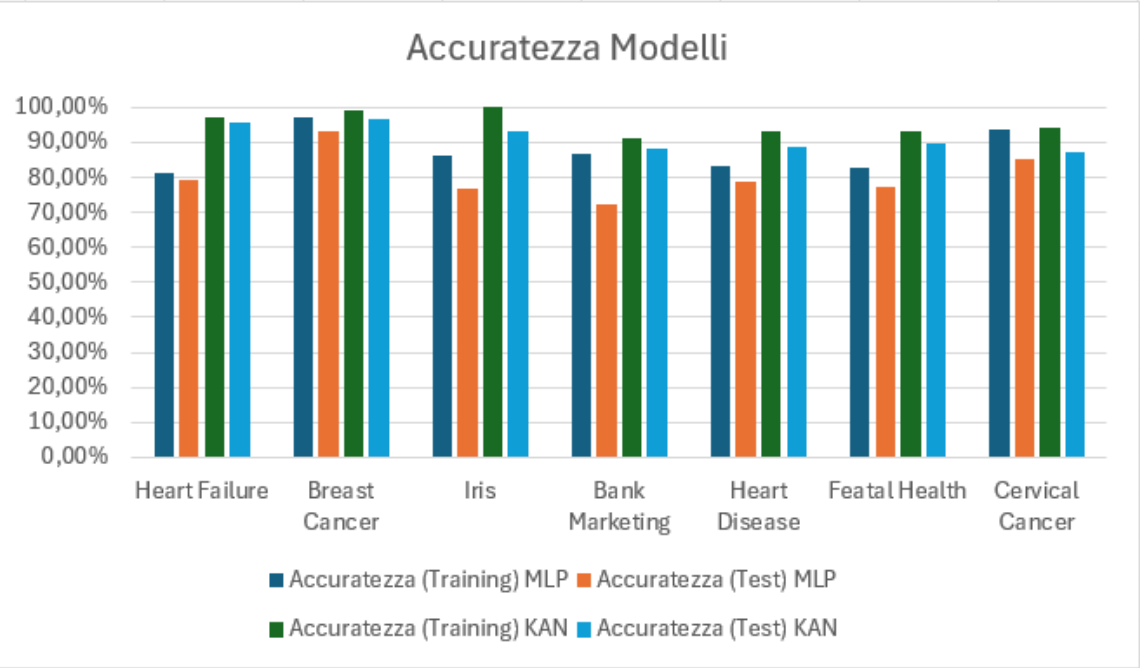
Le KAN richiedono più risorse computazionali soprattutto nei dataset che presentano variabili fortemente non lineari o classi sbilanciate. Tuttavia, va notato che la differenza nel tempo di addestramento, pur presente, rimane gestibile in termini pratici, soprattutto quando il miglioramento in termini di accuratezza compensa il costo computazionale aggiuntivo.

### 6.1.2. Numero di Parametri

Per quanto riguarda il numero di parametri, le reti KAN tendono a essere più complesse rispetto alle MLP. Il numero di parametri da ottimizzare nelle KAN risulta essere superiore, principalmente a causa dell'architettura che permette di modellare relazioni non lineari più raffinate attraverso funzioni spline. Tuttavia, questo aspetto può essere visto come un vantaggio. Sebbene le KAN richiedano un numero maggiore di parametri, si è osservato che un'architettura KAN più semplice può ottenere performance di accuratezza equivalenti a quelle delle MLP, con meno sforzi in termini di complessità del modello.

Ciò significa che le KAN offrono maggiore flessibilità e potenziale di ottimizzazione. Mentre le MLP tendono a richiedere configurazioni più complesse per migliorare l'accuratezza, le KAN possono raggiungere gli stessi risultati mantenendo un buon equilibrio tra complessità e prestazioni. Questa capacità di adattarsi a strutture più semplici senza compromettere le prestazioni rappresenta un punto di forza significativo per le KAN, soprattutto in applicazioni dove è importante ridurre la complessità del modello pur mantenendo elevate prestazioni predittive.

### 6.1.3. Accuratezza




Sul fronte dell'accuratezza, le KAN si dimostrano superiori alle MLP nella maggior parte dei dataset. In particolare, nei contesti con classi sbilanciate o relazioni fortemente non lineari, come ad esempio nei dataset "Bank Marketing" e "Heart Disease", le KAN hanno ridotto significativamente il tasso di errore rispetto alle MLP. Questo risultato riflette la capacità delle KAN di catturare relazioni complesse tra le variabili e di generalizzare meglio sui dati non visti.

Le MLP, pur raggiungendo buoni livelli di accuratezza, tendono a soffrire maggiormente di overfitting nei dataset più complessi. L'accuratezza della KAN, spesso superiore al 90%, dimostra una maggiore stabilità anche in presenza di dataset difficili da modellare, mantenendo prestazioni costanti sia sul training set che sul test set. Questo conferma che le KAN sono più adatte a gestire la complessità intrinseca dei dati senza perdere la capacità di generalizzare efficacemente.

# CONCLUSIONI

SOMMARIO: Conclusioni

# Conclusioni

Nel corso di questo lavoro di tesi, è stato condotto un confronto sistematico tra due tipologie di reti neurali: le Kolmogorov-Arnold Networks (KAN) e le Multi-Layer Perceptron (MLP). L'obiettivo principale è stato quello di valutare le prestazioni di queste due architetture in vari scenari di classificazione, considerando fattori chiave come l'accuratezza, il numero di parametri e il tempo di addestramento.

**Sintesi e Analisi dei Risultati**

Dall'analisi è emersa una chiara superiorità delle reti KAN rispetto alle MLP in termini di accuratezza predittiva, soprattutto in contesti caratterizzati da relazioni non lineari complesse tra le variabili. Le KAN, grazie alla loro capacità di scomporre funzioni multivariate in componenti univariate, hanno dimostrato di essere particolarmente efficaci nel gestire problemi complessi, raggiungendo livelli di precisione superiori.

Un elemento chiave che differenzia le KAN dalle MLP è la loro flessibilità strutturale. Nonostante le KAN tendano generalmente a utilizzare un numero maggiore di parametri, queste reti offrono una notevole capacità di generalizzazione e una ridotta tendenza all'overfitting. Infatti, anche con un aumento dei parametri, le KAN mantengono prestazioni elevate senza compromettere la generalizzazione sui dati di test. Al contrario, le MLP richiedono spesso architetture più complesse per migliorare l'accuratezza, esponendosi così al rischio di sovradattamento, soprattutto quando si opera su dataset meno equilibrati.

Tuttavia, uno svantaggio rilevante delle reti KAN riguarda il tempo di addestramento. La loro complessità computazionale, dovuta principalmente all'uso delle B-spline per modellare le relazioni non lineari, si traduce in tempi di addestramento significativamente più lunghi rispetto alle MLP. Questo può rappresentare una limitazione in applicazioni in cui l'efficienza temporale è critica.

Tuttavia, questa maggiore richiesta di risorse è compensata dall'elevata accuratezza che le KAN sono in grado di ottenere, rendendole la scelta preferibile in contesti in cui la precisione predittiva è fondamentale, come nel settore medico o finanziario.

Un aspetto particolarmente interessante riguarda l'utilizzo dei parametri nelle reti KAN. Sebbene queste reti tendano a richiedere un numero maggiore di parametri rispetto alle MLP, grazie alle proprietà delle B-spline è possibile semplificare l'architettura riducendo il numero di parametri senza compromettere l'accuratezza. Ciò consente di ottenere modelli più snelli e altrettanto performanti, dimostrando che la complessità delle KAN può essere modulata a seconda delle esigenze del problema.

**Implicazioni dei Risultati**

I risultati ottenuti suggeriscono che le Kolmogorov-Arnold Networks rappresentano una soluzione particolarmente promettente per affrontare problemi di apprendimento automatico caratterizzati da relazioni complesse tra variabili. La loro capacità di modellare accuratamente relazioni non lineari, combinata con la flessibilità nella gestione dei parametri, offre vantaggi tangibili rispetto alle MLP, soprattutto in contesti dove la precisione è cruciale.

Le MLP, pur essendo competitive per quanto riguarda i tempi di addestramento e la semplicità architetturale, mostrano limiti evidenti nei contesti in cui è necessaria una modellazione più sofisticata delle relazioni tra variabili. Le loro prestazioni risultano soddisfacenti nei problemi semplici o ben bilanciati, ma degradano quando la complessità aumenta o quando la non linearità delle relazioni diventa predominante.

## Prospettive Future e Possibili Sviluppi

Alla luce dei risultati ottenuti, si delineano diverse opportunità per migliorare ulteriormente le reti KAN. Un ambito di particolare interesse riguarda l'ottimizzazione del tempo di addestramento. Lo sviluppo di nuove tecniche che riducano la complessità computazionale senza compromettere l'efficacia predittiva potrebbe rendere le KAN applicabili in contesti in cui la rapidità di esecuzione è essenziale, come nei sistemi in tempo reale o con risorse computazionali limitate.

Un'altra direzione promettente è la riduzione del numero di parametri, mantenendo lo stesso livello di accuratezza, attraverso approcci di ottimizzazione come la potatura dei modelli o l'adozione di tecniche avanzate di ottimizzazione. Questo porterebbe a una riduzione della complessità computazionale e a un'applicabilità più ampia delle KAN.

Infine, un'area di ricerca futura potrebbe riguardare l'applicazione delle reti KAN a domini più complessi, come l'elaborazione di dati non strutturati (ad esempio, immagini, testo e dati temporali), dove finora le reti convoluzionali e le MLP hanno dominato. La capacità delle KAN di rappresentare relazioni non lineari potrebbe offrire nuovi strumenti per affrontare problemi in questi settori, spingendo i confini delle attuali tecnologie di apprendimento automatico.

## Conclusione Finale

In conclusione, il confronto tra le reti KAN e MLP ha dimostrato la superiorità delle KAN in termini di accuratezza e gestione della complessità, pur riconoscendo la necessità di ottimizzazioni dal punto di vista dell'efficienza computazionale. Le reti KAN rappresentano una valida alternativa alle MLP, soprattutto nei contesti complessi in cui la capacità di modellare relazioni non lineari è fondamentale. Questo lavoro di tesi contribuisce a porre le basi per ulteriori sviluppi nel campo

delle KAN, incoraggiando la ricerca e suggerendo potenziali miglioramenti per ampliare il loro utilizzo in ambito accademico e industriale.

# RIFERIMENTI


Agliari, E., Alessandrelli, A., Barra, A., Centonze, M. S., & Ricci-Tersenghi, F. (2024). Generalized hetero-associative neural networks. arXiv preprint arXiv:2409.08151.

Basheer, I. A., & Hajmeer, M. (2000). Artificial neural networks: fundamentals, computing, design, and application. Journal of microbiological methods, 43(1), 3-31.

Cilimkovic, M. (2015). Neural networks and back propagation algorithm. Institute of Technology Blanchardstown, Blanchardstown Road North Dublin, 15(1).

Fine, T. L. (2006). Feedforward neural network methodology. Springer Science & Business Media.

Günther, F., & Fritsch, S. (2010). Neuralnet: training of neural networks. R J., 2(1), 30.

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). Deep Machine Learning-A New Frontier in Artificial Intelligence Research. A fast learning algorithm for deep belief nets. Neural Computation, 18(7), 1527-1554.

Kelleher, J. D. (2019). Deep learning. MIT press.

Krenker, A., Bešter, J., & Kos, A. (2011). Introduction to the artificial neural networks. Artificial Neural Networks: Methodological Advances and Biomedical Applications. InTech, 1-18.

Liu, Z., Wang, Y., Vaidya, S., Ruehle, F., Halverson, J., Soljačić, M., ... & Tegmark, M. (2024). Kan: Kolmogorov-arnold networks. arXiv preprint arXiv:2404.19756.

Mercioni, M. A., & Holban, S. (2020, May). The most used activation functions: Classic versus current. In 2020 International Conference on Development and Application Systems (DAS) (pp. 141-145). IEEE.

Nishijima, T. (2021). Universal approximation theorem for neural networks. arXiv preprint arXiv:2102.10993.

Parhi, R., & Nowak, R. D. (2020). The role of neural network activation functions. IEEE Signal Processing Letters, 27, 1779-1783.

Popescu, M. C., Balas, V. E., Perescu-Popescu, L., & Mastorakis, N. (2009). Multilayer perceptron and neural networks. WSEAS Transactions on Circuits and Systems, 8(7), 579-588.

Prautzsch, H. (2002). Bézier and B-spline techniques.

Rajoub, B. (2020). Supervised and unsupervised learning. In Biomedical signal processing and artificial intelligence in healthcare (pp. 51-89). Academic Press.

Rosenblatt, F. (2021). The Perceptron: A Probabilistic Model for Information Storage and Organization (1958).

Schmidt-Hieber, J. (2021). The Kolmogorov–Arnold representation theorem revisited. Neural networks, 137, 119-126.

Sharma, S., Sharma, S., & Athaiya, A. (2017). Activation functions in neural networks. Towards Data Sci, 6(12), 310-316.

Wang, H., & Raj, B. (2017). On the origin of deep learning. arXiv preprint arXiv:1702.07800.

Wu, Y. C., & Feng, J. W. (2018). Development and application of artificial neural network. Wireless Personal Communications, 102, 1645-1656.

# SITOGRAFIA

https://www.ibm.com/it-it/topics/neural-networks

https://www.researchgate.net/figure/Multilayer-Perceptron-Advantages-and-Disadvantages_tbl4_338950098

https://github.com/KindXiaoming/pykan.

www.kaggle.com

https://archive.ics.uci.edu/

# APPENDICE

## Codice implementazione Kolmogorov – Arnold Network

```
import os
import sys
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from kan import KAN  # Import from the KAN package
import moviepy.video.io.ImageSequenceClip

# Cambia la directory di lavoro al percorso del progetto (modifica questo percorso se necessario)
os.chdir(r'C:\Users\Luca Ruocco\Desktop\pykan-master')

# Aggiungi il percorso del pacchetto alla lista di percorsi di sistema
sys.path.append(os.path.abspath(r'C:\Users\Luca Ruocco\Desktop\pykan-master'))

# Specifica il percorso del file CSV (dataset)
file_path = r'C:\Users\Luca Ruocco\Desktop\Dataset Tesi Luca\1. HEART\heart.csv'
```

```
# Leggi il dataset da CSV
df = pd.read_csv(file_path, sep=',')
print(df.head())  # Mostra le prime righe del dataset per controllo

# Informazioni sul dataset
df.info()  # Mostra il tipo di dati e se ci sono valori nulli
print(df.isnull().sum())  # Verifica la presenza di valori nulli

# Converti alcune colonne in tipo 'object' (categorie)
categorical_cols = ['sex', 'chest pain type', 'fasting blood sugar', 'resting ecg', 'target',
'exercise angina', 'ST slope']
df[categorical_cols] = df[categorical_cols].astype('object')

# Encoding delle variabili categoriali in numeriche
le = LabelEncoder()  # Inizializza il Label Encoder
for col in categorical_cols:
    df[col] = le.fit_transform(df[col])

# Separazione delle caratteristiche (X) e del target (y)
X = df.drop('target', axis=1)
y = df['target']

# Standardizzazione delle caratteristiche numeriche
numerical_features = X.select_dtypes(include=['number']).columns
scaler = StandardScaler()
```

```
X[numerical_features] = scaler.fit_transform(X[numerical_features])

# Suddivisione dei dati in train e test set, con stratificazione
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

# Conversione dei dati in tensori per PyTorch
dataset = {
    'train_input': torch.tensor(X_train.values.astype('float32')),
    'test_input': torch.tensor(X_test.values.astype('float32')),
    'train_label': torch.tensor(y_train.values.astype('float32')).view(-1, 1),
    'test_label': torch.tensor(y_test.values.astype('float32')).view(-1, 1)
}

# Conferma delle dimensioni dei tensori
print(dataset['train_input'].shape, dataset['train_label'].shape)
print(dataset['test_input'].shape, dataset['test_label'].shape)

### 1. Addestramento Iniziale ###

# Inizializzazione del modello KAN
model = KAN(width=[X_train.shape[1], 8, 2], grid=3, k=3)

# Funzioni per calcolare l'accuratezza
def train_acc():
    with torch.no_grad():
```

```
        return torch.mean((torch.round(model(dataset['train_input'])[:, 0]) ==
dataset['train_label'][:, 0]).float())


def test_acc():
    with torch.no_grad():
        return torch.mean((torch.round(model(dataset['test_input'])[:, 0]) ==
dataset['test_label'][:, 0]).float())


# Addestramento del modello utilizzando LBFGS
results = model.train(
    dataset,
    opt="LBFGS",
    steps=5,
    metrics=(train_acc, test_acc),
    save_fig=False,
    beta=7
)


# Stampa dell'accuratezza finale
print(f"Train Accuracy: {results['train_acc'][-1]:.4f}")
print(f"Test Accuracy: {results['test_acc'][-1]:.4f}")


# Plot del modello dopo l'addestramento
model.plot(beta=2, scale=1, title='Trained KAN Model')


### 2. Pruning ###
```

```
# Pruning del modello per rimuovere i pesi irrilevanti
model_pruned = model.prune(threshold=1e-4)

# Plot del modello potato
model_pruned.plot(scale=1, beta=10, title='Pruned Model')

### 3. Riaddestramento ###

# Riaddestramento del modello potato utilizzando LBFGS
results_pruned = model_pruned.train(
    dataset,
    opt="LBFGS",
    steps=5,
    metrics=(train_acc, test_acc),
    save_fig=False,
    beta=7
)

# Stampa dell'accuratezza dopo il riaddestramento
print(f"Train Accuracy after pruning: {results_pruned['train_acc'][-1]:.4f}")
print(f"Test Accuracy after pruning: {results_pruned['test_acc'][-1]:.4f}")

### 4. Fix delle Funzioni di Attivazione ###

# Libreria di funzioni di attivazione simboliche
lib = ['x', 'x^2', 'x^3', 'x^4', 'exp', 'log', 'sqrt', 'tanh', 'sin', 'tan', 'abs']
```

```
# Autotune delle funzioni di attivazione
model_pruned.auto_symbolic(lib=lib)

### 5. Calcolo Formula Simbolica ###

# Estrai le formule simboliche per il modello pruned
formula1, formula2 = model_pruned.symbolic_formula()[0]

# Funzione per calcolare l'accuratezza della formula simbolica
import sympy as sp
def acc(formula, X, y):
    batch = X.shape[0]
    correct = 0
    formula_func = sp.lambdify([sp.symbols(f'x_{i+1}') for i in range(X.shape[1])],
formula, "numpy")
    for i in range(batch):
        prediction = formula_func(*X[i])
        correct += (np.round(prediction) == y[i])
    return correct / batch

# Accuratezza delle formule simboliche su training e test set
train_accuracy = acc(formula1, dataset['train_input'].numpy(),
dataset['train_label'].numpy())
test_accuracy = acc(formula1, dataset['test_input'].numpy(),
dataset['test_label'].numpy())
```

```
train_accuracy2 = acc(formula2, dataset['train_input'].numpy(),
dataset['train_label'].numpy())

test_accuracy2 = acc(formula2, dataset['test_input'].numpy(),
dataset['test_label'].numpy())


print('Train accuracy of the formula 1: ', train_accuracy)

print('Test accuracy of the formula 1:', test_accuracy)

print('Train accuracy of the formula 2: ', train_accuracy2)

print('Test accuracy of the formula 2:', test_accuracy2)


# Visualizzazioni del modello con diverse beta per interpretazione delle funzioni
simboliche

model_pruned.plot(beta=3, scale=1, title='Low Beta Visualization')

model_pruned.plot(beta=100, scale=1, title='High Beta Visualization')

# Addestramento del modello con diverse griglie

grids = np.array([5, 10, 20, 50, 100])

train_losses = []

test_losses = []

steps = 50

k = 3


for i in range(grids.shape[0]):

   if i == 0:

      model = KAN(width=[X_train.shape[1], 8, 2], grid=grids[i], k=k)

   else:

      model = KAN(width=[X_train.shape[1], 8, 2], grid=grids[i],
k=k).initialize_from_another_model(model, dataset['train_input'])
```

```
    results = model.train(dataset, opt="LBFGS", steps=steps, stop_grid_update_step=30, metrics=(train_acc, test_acc))

    train_losses += results['train_loss']
    test_losses += results['test_loss']

# Plot delle perdite di addestramento e di test
plt.plot(train_losses)
plt.plot(test_losses)
plt.legend(['train', 'test'])
plt.ylabel('RMSE')
plt.xlabel('step')
plt.yscale('log')
plt.show()

# Plot del numero di parametri vs RMSE
n_params = 3 * grids
train_vs_G = train_losses[(steps-1)::steps]
test_vs_G = test_losses[(steps-1)::steps]
plt.plot(n_params, train_vs_G, marker="o")
plt.plot(n_params, test_vs_G, marker="o")
plt.plot(n_params, 100*n_params**(-4.), ls="--", color="black")
plt.xscale('log')
plt.yscale('log')
plt.legend(['train', 'test', r'$N^{-4}$'])
```

```
plt.xlabel('number of params')
plt.ylabel('RMSE')
plt.show()

# Calcolo dell'importanza delle feature
from sklearn.metrics import accuracy_score

def calculate_feature_importance(model, X_test, y_test, feature_names):
    baseline_preds = torch.round(model(X_test)[:, 0])
    baseline_acc = accuracy_score(y_test, baseline_preds.detach().numpy())

    feature_importances = {}

    for i in range(X_test.shape[1]):
        X_permuted = X_test.clone()
        X_permuted[:, i] = X_permuted[:, i][torch.randperm(X_permuted.size(0))]

        permuted_preds = torch.round(model(X_permuted)[:, 0])
        permuted_acc = accuracy_score(y_test, permuted_preds.detach().numpy())

        importance = baseline_acc - permuted_acc
        feature_importances[feature_names[i]] = importance

    sorted_importances = sorted(feature_importances.items(), key=lambda x: x[1],
reverse=True)
```

```
    return sorted_importances


# Definisci i nomi delle feature
feature_names = ['Feature_0','Feature_1', 'Feature_2', 'Feature_3', 'Feature_4',
'Feature_5','Feature_6','Feature_7','Feature_8','Feature_9','Feature_10']


# Calcola l'importanza delle feature
importances = calculate_feature_importance(model, dataset['test_input'],
dataset['test_label'], feature_names)


# Stampa delle feature ordinate per importanza
for feature, importance in importances:
    print(f'{feature}: {importance:.4f}')
```

## Codice implementazione Multi Layer Perceptron

```
import torch

import torch.nn as nn

import torch.optim as optim

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.metrics import accuracy_score, classification_report

import pandas as pd

import numpy as np


# Caricamento e preparazione del dataset

df = pd.read_csv(r'C:\Users\Luca Ruocco\Desktop\Dataset Tesi Luca\2. HEART FAILURE\heart_failure_clinical_records.csv')


# Separazione tra caratteristiche (X) e target (y)

X = df.drop('DEATH_EVENT', axis=1).values  # Rimuovi la colonna 'DEATH_EVENT' per ottenere X

y = df['DEATH_EVENT'].values  # Target variabile (DEATH_EVENT)

df.head()  # Mostra le prime righe del dataframe


# Controllo delle modalità nella variabile target

n_modalities = df['DEATH_EVENT'].nunique()  # Numero di modalità nella colonna target

print(f"La variabile 'DEATH_EVENT' ha {n_modalities} modalità.")


# Controllo dei valori mancanti
```

```
missing_values = df.isnull().sum()  # Conta il numero di valori mancanti in ogni colonna
print(missing_values)  # Mostra le colonne con il numero di valori mancanti

# Filtra solo le colonne che hanno valori mancanti (se presenti)
missing_values_filtered = missing_values[missing_values > 0]
print(missing_values_filtered)

# Suddivisione dei dati in training e test set, con stratificazione sulla variabile target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42,
stratify=y)

# Standardizzazione dei dati per avere media 0 e varianza 1
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)  # Applica fit e trasformazione ai dati di training
X_test = scaler.transform(X_test)  # Trasforma i dati di test con la stessa scala

# Conversione dei dati in tensori PyTorch
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_train_tensor = torch.tensor(y_train, dtype=torch.long)
y_test_tensor = torch.tensor(y_test, dtype=torch.long)

# Imposta il seed per garantire la riproducibilità dei risultati
seed = 42
torch.manual_seed(seed)
np.random.seed(seed)
```

```
random.seed(seed)


# Se stai usando una GPU, puoi impostare anche questo:
torch.cuda.manual_seed(seed)
torch.cuda.manual_seed_all(seed)  # Se usi più GPU


# Assicurati che il determinismo sia attivo (può rallentare l'esecuzione)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False


################################################################################
#####################################################
# Definizione della classe MLP (Multilayer Perceptron) con variabile funzione di attivazione
class MLP(nn.Module):
    def __init__(self, input_size, hidden_sizes, output_size, activation_fn):
        super(MLP, self).__init__()
        layers = []
        in_size = input_size  # Numero di input features
        for h in hidden_sizes:
            layers.append(nn.Linear(in_size, h))  # Aggiungi uno strato lineare
            layers.append(activation_fn())  # Aggiungi la funzione di attivazione selezionata
            in_size = h  # Aggiorna il numero di input per lo strato successivo
        layers.append(nn.Linear(in_size, output_size))  # Strato finale con output_size (numero di classi)
        self.network = nn.Sequential(*layers)  # Crea una sequenza di layer
```

```
    def forward(self, x):
        return self.network(x)  # Esegue il forward pass

# Funzione per addestrare e valutare il modello MLP
def train_and_evaluate_model(hidden_sizes, activation_fn):
    model = MLP(input_size=X_train.shape[1], hidden_sizes=hidden_sizes, output_size=len(np.unique(y)), activation_fn=activation_fn)
    criterion = nn.CrossEntropyLoss()  # Funzione di loss per classificazione multi-classe
    optimizer = optim.Adam(model.parameters(), lr=0.001)  # Ottimizzatore Adam con learning rate 0.001

    num_epochs = 50  # Numero di epoche di addestramento
    for epoch in range(num_epochs):
        model.train()  # Imposta il modello in modalità addestramento
        outputs = model(X_train_tensor)  # Forward pass
        loss = criterion(outputs, y_train_tensor)  # Calcola la loss
        optimizer.zero_grad()  # Azzera i gradienti
        loss.backward()  # Backpropagation
        optimizer.step()  # Aggiorna i pesi

    # Valutazione del modello sul test set
    model.eval()  # Imposta il modello in modalità valutazione
    with torch.no_grad():
        test_outputs = model(X_test_tensor)  # Forward pass sul test set
        _, test_predicted = torch.max(test_outputs.data, 1)  # Ottieni le predizioni
        test_accuracy = accuracy_score(y_test_tensor, test_predicted)  # Calcola l'accuratezza
```

```
    return test_accuracy, model  # Ritorna l'accuratezza e il modello addestrato

# Configurazioni di strutture di layer da testare
hidden_layer_structures = [
    [5], [6], [7], [8], [10], [12], [15], [18], [20], [25], [30],
    [5,5], [6,5], [6,6], [7,5], [7,6], [7,7],
    [8,5], [8,6], [8,7], [8,8],
    [10,5], [10,6], [10,7], [10,8], [10,10],
    [12,5], [12,6], [12,7], [12,8], [12,10], [12,12],
    [15,5], [15,6], [15,7], [15,8], [15,10], [15,12], [15,15],
    [20,5], [20,6], [20,7], [20,8], [20,10], [20,12], [20,15], [20,20],
    [32], [64], [32, 16], [64, 32]
]

# Funzioni di attivazione da testare
activation_functions = [
    nn.ReLU, nn.Sigmoid, nn.Tanh, nn.LeakyReLU, nn.ELU, nn.SELU,
    nn.Softplus, nn.Softsign, nn.ReLU6, nn.GELU, nn.Hardtanh, nn.LogSigmoid
]

# Dizionario per salvare i risultati
results = {}
selected_model = None
selected_config = None
```

```
# Loop su tutte le strutture e funzioni di attivazione da testare
for structure in hidden_layer_structures:
    for activation_fn in activation_functions:
        accuracy, model = train_and_evaluate_model(structure, activation_fn)
        config_name = f"Structure: {structure}, Activation: {activation_fn.__name__}"
        results[config_name] = accuracy
        print(f"{config_name} -> Test Accuracy = {accuracy:.4f}")

        # Salva il modello se soddisfa i criteri desiderati (ad esempio, struttura [10,5] e ReLU)
        if structure == [10,5] and activation_fn == nn.ReLU:
            selected_model = model  # Salva il modello corrente
            selected_config = config_name  # Salva la configurazione

# Se un modello è stato selezionato, salvalo su disco
if selected_model is not None:
    torch.save(selected_model.state_dict(), 'selected_model.pth')
    print(f"Modello selezionato salvato come 'selected_model.pth' con configurazione: {selected_config}")
else:
    print("Nessun modello corrisponde ai criteri specificati.")

# Caricamento del modello selezionato (opzionale)
loaded_model = MLP(input_size=X_train.shape[1], hidden_sizes=[10,5], output_size=len(np.unique(y)), activation_fn=nn.ReLU)
loaded_model.load_state_dict(torch.load('selected_model.pth'))  # Carica i pesi salvati nel modello
```

```
# Valutazione del modello caricato
loaded_model.eval()
with torch.no_grad():
    test_outputs = loaded_model(X_test_tensor)
    _, test_predicted = torch.max(test_outputs.data, 1)

# Calcolo delle metriche di valutazione
accuracy = accuracy_score(y_test_tensor.numpy(), test_predicted.numpy())
print(f"Accuracy of the loaded model: {accuracy:.4f}")

# Visualizzazione della matrice di confusione e altre metriche di valutazione
import seaborn as sns
from sklearn.metrics import confusion_matrix, classification_report

cm = confusion_matrix(y_test_tensor.numpy(), test_predicted.numpy())
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix of Loaded Model')
plt.show()

# Report di classificazione
print("\nClassification Report:\n", classification_report(y_test_tensor.numpy(),
test_predicted.numpy()))
```

```
# Visualizzazione della ROC curve
from sklearn.metrics import roc_curve, auc

with torch.no_grad():
    probabilities = torch.softmax(test_outputs, dim=1)[:, 1].numpy()  # Probabilità per la classe positiva

# Calcolo delle metriche ROC
fpr, tpr, thresholds = roc_curve(y_test_tensor.numpy(), probabilities)
roc_auc = auc(fpr, tpr)

# Plot della ROC curve
plt.figure()
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.4f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC)')
plt.legend(loc='lower right')
plt.show()

# Importanza delle feature tramite permutazione
from sklearn.utils import shuffle
```

```
# Funzione per calcolare l'importanza delle feature tramite permutazione
def permutation_importance(model, X_test, y_test):
    baseline_accuracy = accuracy_score(y_test, model(X_test).argmax(dim=1).numpy())
    importances = []

    for col in range(X_test.shape[1]):
        X_test_permuted = X_test.clone()  # Clona i dati di test
        X_test_permuted[:, col] = torch.tensor(shuffle(X_test[:, col].numpy()))  # Permuta i valori di una feature
        permuted_accuracy = accuracy_score(y_test, model(X_test_permuted).argmax(dim=1).numpy())
        importances.append(baseline_accuracy - permuted_accuracy)  # Calcola l'importanza come differenza di accuratezza

    return np.array(importances)

# Calcola l'importanza delle feature
feature_importances = permutation_importance(loaded_model, X_test_tensor, y_test_tensor)
sorted_idx = np.argsort(feature_importances)[::-1]

# Visualizzazione dell'importanza delle feature
feature_names = list(df.columns[:-1])  # Nomi delle feature
print("Feature Importance (Permutation):")
for idx in sorted_idx:
    print(f"Feature {feature_names[idx]}: {feature_importances[idx]:.4f}")
```

```
# Mappa di salienza
def saliency_map(model, X_sample, y_sample):
    model.eval()
    X_sample.requires_grad_()  # Richiede il gradiente per l'input
    output = model(X_sample)  # Passa il campione nel modello
    output_max = output[0, output.argmax(dim=1).item()]  # Seleziona l'output corrispondente alla classe predetta
    output_max.backward()  # Calcola il gradiente rispetto all'input
    saliency = X_sample.grad.abs().squeeze().detach().cpu().numpy()  # Ottieni la mappa di salienza
    return saliency

# Calcolo della mappa di salienza per un campione di test
X_sample = X_test_tensor[0:1]  # Prendi il primo campione dal test set
saliency = saliency_map(loaded_model, X_sample, y_test_tensor[0:1])

# Visualizzazione della mappa di salienza
plt.figure(figsize=(10, 8))
plt.barh(range(len(saliency)), saliency, color='skyblue')
plt.yticks(ticks=range(len(saliency)), labels=feature_names)
plt.xlabel('Saliency Value')
plt.ylabel('Feature')
plt.title('Saliency Map for the First Test Sample')
plt.gca().invert_yaxis()  # Inverte l'asse y per avere la feature più importante in alto
plt.show()

# Visualizzazione della rete neurale
```

```
import networkx as nx

def draw_neural_network(layer_sizes):
    G = nx.DiGraph()

    # Aggiungi i nodi dei layer
    for i, layer_size in enumerate(layer_sizes):
        for j in range(layer_size):
            G.add_node(f'Layer {i} - Node {j}', pos=(i, -j))

    # Aggiungi le connessioni tra i nodi
    for i in range(len(layer_sizes) - 1):
        for j in range(layer_sizes[i]):
            for k in range(layer_sizes[i + 1]):
                G.add_edge(f'Layer {i} - Node {j}', f'Layer {i + 1} - Node {k}')

    # Disegna il grafo
    pos = nx.get_node_attributes(G, 'pos')
    nx.draw(G, pos, with_labels=True, node_size=150, node_color='red', arrowsize=1)
    plt.show()

# Definisci le dimensioni dei layer della rete
layer_sizes = [X_train.shape[1], 10, 5, len(np.unique(y))]  # Input, hidden layers, output
draw_neural_network(layer_sizes)
```